Oracle® Machine Learning for Python

User's Guide

Release 2.1.1 for Oracle AI Database 26ai
G19940-18
May 2026

Oracle Machine Learning for Python User's Guide, Release 2.1.1 for Oracle AI Database 26ai

G19940-18

Copyright © 2024, 2026, Oracle and/or its affiliates.

Primary Author: Dhanish Kumar

Contributors: Andi Wang , Boriana Milenova , David McDermid , Feng Li , Mandeep Kaur , Mark Hornick, Qin Wang , Sherry Lamonica , Venkatanathan Varadarajan , Yu Xiang

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Audience i
Related Resources i
Conventions i
Documentation Accessibility ii

1 Changes in This Release

1.1 OML4Py 2.1.1 Updates 1
1.2 OML4Py 2.1 Updates 1
1.3 OML4Py 2.0.1 Updates 2
1.4 OML4Py 2.0 Updates 3
1.5 Deprecated Features 5

2 About Oracle Machine Learning for Python

2.1 What Is Oracle Machine Learning for Python 1
2.2 Advantages of Oracle Machine Learning for Python 2
2.3 Manipulate database tables and views using familiar Python functions and syntax 4
2.4 About the Python Components and Libraries in OML4Py 6

3 Install OML4Py Client for Linux for Use With Autonomous AI Database Serverless

4 Install OML4Py for On-Premises Databases

4.1 Database and System Requirements for OML4Py 1
4.2 Getting Started with OML4Py Installation 2
4.2.1 Install OML4Py 2.1.1 for On-Premises Databases 2
4.2.1.1 Build and Install Python for Linux for On-Premises Databases 3
4.2.1.2 Install the Required Supporting Packages for Linux for On-Premises Databases 6
4.2.1.3 Install OML4Py Server for On-Premises Oracle AI Database 9

Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.1.4 Install OML4Py Client for Linux for On-Premises Oracle AI Database 20
4.2.1.5 Install OML4Py Slim Client 27
4.2.2 Install OML4Py 2.1 for On-Premises Databases 29
4.2.2.1 Build and Install Python for Linux for On-Premises Databases 30
4.2.2.2 Install the Required Supporting Packages for Linux for On-Premises Databases 32
4.2.2.3 Install OML4Py Server for On-Premises Oracle Database 34
4.2.2.4 Install OML4Py Client for Linux for On-Premises Oracle AI Database 44
4.2.2.5 Install OML4Py Client for Windows for On-Premises and Autonomous Oracle AI Database 51
4.2.3 Install OML4Py 2.0.1 for On-Premises Databases 62
4.2.3.1 Build and Install Python for Linux for On-Premises Databases 62
4.2.3.2 Install the Required Supporting Packages for Linux for On-Premises Databases 65
4.2.3.3 Install OML4Py Server for On-Premises Oracle Database 66
4.2.3.4 Install OML4Py Client for On-Premises Oracle Database 76
4.2.4 Install OML4Py 2.0 for On-Premises Databases 83
4.2.4.1 Build and Install Python for Linux for On-Premises Databases 84
4.2.4.2 Install the Required Supporting Packages for Linux for On-Premises Databases 86
4.2.4.3 Install OML4Py Server for On-Premises Oracle Database 87
4.2.4.4 Install OML4Py Client for On-Premises Oracle Database 97

5 Install OML4Py Client Container

5.1 Configure Pooled Connections for AutoML 6

6 Install OML4Py on Exadata

6.1 About Oracle Machine Learning for Python on Exadata 1
6.2 Configure DCLI to install Python across Exadata compute nodes. 2
6.2.1 Install Python across Exadata compute nodes using DCLI 4
6.2.2 Install OML4Py across Exadata compute nodes using DCLI 5

7 Install Third-Party Packages

7.1 Conda Commands 1
7.2 Administrative Tasks for Creating and Saving a Conda Environment 9
7.3 OML User Tasks for Downloading an Available Conda Environment 13
7.4 Using Conda Environments with Embedded Python Execution 19
7.5 Conda Environment Best Practices 22

Copyright © 2024, 2026, Oracle and/or its affiliates.

8 Get Started with Oracle Machine Learning for Python

8.1 Use OML4Py with Oracle Autonomous AI Database 1
8.2 Use OML4Py with an On-Premises Oracle AI Database 1
8.2.1 About Connecting to an On-Premises Oracle AI Database 2
8.2.2 About Oracle Wallets 3
8.2.3 Connect to an Oracle AI Database 4
8.3 Move Data Between the Database and a Python Session 8
8.3.1 About Moving Data Between the Database and a Python Session 9
8.3.2 Push Local Python Data to the Database 9
8.3.3 Pull Data from the Database to a Local Python Session 11
8.3.4 Create a Python Proxy Object for a Database Object 13
8.3.5 Create a Persistent Database Table from a Python Data Set 15
8.4 Save Python Objects in the Database 19
8.4.1 About OML4Py Datastores 20
8.4.2 Save Objects to a Datastore 21
8.4.3 Load Saved Objects From a Datastore 24
8.4.4 Get Information About Datastores 25
8.4.5 Get Information About Datastore Objects 27
8.4.6 Delete Datastore Objects 28
8.4.7 Manage Access to Stored Objects 30
8.5 Import and Export Datastore 32
8.5.1 Export Datastore 33
8.5.2 Import Datastore 34

9 Prepare and Explore Data

9.1 Prepare Data 1
9.1.1 About Preparing Data in the Database 1
9.1.2 Select Data 4
9.1.3 Combine Data 8
9.1.4 Clean Data 13
9.1.5 Split Data 15
9.1.5.1 Define the Target Variable 17
9.2 Explore Data 18
9.2.1 About the Exploratory Data Analysis Methods 19
9.2.2 Correlate Data 21
9.2.3 Cross-Tabulate Data 23
9.2.4 Mutate Data 25
9.2.5 Sort Data 28
9.2.6 Summarize Data 30
9.2.7 Date, Time, and Integer Data 33

Copyright © 2024, 2026, Oracle and/or its affiliates.

9.3 Render Graphics 41

10 OML4Py Classes That Provide Access to In-Database Machine Learning Algorithms

10.1 About Machine Learning Classes and Algorithms 2
10.2 About Model Settings 5
10.3 Shared Settings 5
10.4 Export Oracle Machine Learning for Python Models 8
10.5 Automatic Data Preparation 12
10.6 Model Explainability 13
10.7 Attribute Importance 20
10.8 Association Rules 22
10.9 Decision Tree 28
10.10 Expectation Maximization 35
10.11 Explicit Semantic Analysis 49
10.12 Generalized Linear Model 55
10.13 k-Means 65
10.14 Naive Bayes 72
10.15 Neural Network 80
10.16 ONNX 90
10.17 Random Forest 93
10.18 Singular Value Decomposition 100
10.19 Support Vector Machine 106
10.20 Non-Negative Matrix Factorization 112
10.21 Exponential Smoothing Method 118
10.22 XGBoost 128

11 Automated Machine Learning

11.1 About Automated Machine Learning 1
11.2 Algorithm Selection 6
11.3 Feature Selection 8
11.4 Model Tuning 11
11.5 Model Selection 15
11.6 AutoML Pipeline 18

12 Import Pretrained Models in ONNX Format

12.1 Convert Pretrained Models to ONNX Model: End-to-End Instructions for Text Embedding 4
12.2 Python Classes to Convert Pretrained Models to ONNX Models 10
12.3 ONNX Pipeline Models: Text Embedding 18

Copyright © 2024, 2026, Oracle and/or its affiliates.

12.4 ONNX Pipeline Models: Image Embedding 24
12.5 ONNX Pipeline Models: CLIP Multi-Modal Embedding 26
12.6 ONNX Pipeline Models: Text Classification 27
12.7 ONNX Pipeline Models: Reranking Pipeline 30
12.8 Load Custom Models from The Local Filesystem 32
12.9 Support For Large ONNX Format Model 33
12.10 Model Summary 35

13 OML4Py Metrics

14 Embedded Python Execution

14.1 About Embedded Python Execution 2
14.2 Parallelism with OML4Py Embedded Python Execution 2
14.3 Datastores Supporting Embedded Python Execution 3
14.3.1 ALL_PYQ_DATASTORE_CONTENTS View 4
14.3.2 ALL_PYQ_DATASTORES View 5
14.3.3 USER_PYQ_DATASTORES View 6
14.4 Script repository for user-defined Python functions supporting EPE 7
14.4.1 ALL_PYQ_SCRIPTS View 7
14.4.2 USER_PYQ_SCRIPTS View 8
14.5 Python API for Embedded Python Execution 9
14.5.1 About Python API for Embedded Python Execution 9
14.5.2 Run a User-Defined Python Function 11
14.5.3 Run a User-Defined Python Function on the Specified Data 12
14.5.4 Run a Python Function on Data Grouped By Column Values 15
14.5.5 Run a User-Defined Python Function on Sets of Rows 19
14.5.6 Run a User-Defined Python Function Multiple Times 22
14.5.7 Save and Manage User-Defined Python Functions in the Script Repository 24
14.5.7.1 About the Script Repository 25
14.5.7.2 Create and Store a User-Defined Python Function 25
14.5.7.3 List Available User-Defined Python Functions 28
14.5.7.4 Load a User-Defined Python Function 30
14.5.7.5 Drop a User-Defined Python Function from the Repository 31
14.5.7.6 Export a User-Defined Python Function 32
14.5.7.7 Import a User-Defined Python Function 34
14.6 SQL API for Embedded Python Execution with On-premises AI Database 35
14.6.1 About the SQL API for Embedded Python Execution with On-Premises Database 36
14.6.2 pyqEval Function (On-Premises Database) 37
14.6.3 pyqTableEval Function (On-Premises Database) 40

Copyright © 2024, 2026, Oracle and/or its affiliates.

14.6.4 pyqRowEval Function (On-Premises Database) 43
14.6.5 pyqGroupEval Function (On-Premises Database) 47
14.6.6 pyqGrant Function (On-Premises Database) 50
14.6.7 pyqRevoke Function (On-Premises Database) 51
14.6.8 pyqScriptCreate Procedure (On-Premises Database) 52
14.6.9 pyqScriptDrop Procedure (On-Premises Database) 55
14.7 SQL API for Embedded Python Execution with Autonomous AI Database 56
14.7.1 Access and Authorization Procedures and Functions 56
14.7.1.1 pyqAppendHostACE Procedure 58
14.7.1.2 pyqGetHostACE Function 59
14.7.1.3 pyqRemoveHostACE Procedure 59
14.7.1.4 pyqSetAuthToken Procedure 60
14.7.1.5 pyqIsTokenSet Function 60
14.7.2 Embedded Python Execution Functions (Autonomous Database) 61
14.7.2.1 pyqListEnvs Function (Autonomous AI Database) 61
14.7.2.2 pyqEval Function (Autonomous AI Database) 62
14.7.2.3 pyqTableEval Function (Autonomous AI Database) 68
14.7.2.4 pyqRowEval Function (Autonomous AI Database) 72
14.7.2.5 pyqGroupEval Function (Autonomous AI Database) 80
14.7.2.6 pyqIndexEval Function (Autonomous AI Database) 89
14.7.2.7 pyqListAllJobs Function 112
14.7.2.8 pyqDeleteJob Function 113
14.7.2.9 pyqGrant Function (Autonomous AI Database) 113
14.7.2.10 pyqRevoke Function (Autonomous AI Database) 114
14.7.2.11 pyqScriptCreate Procedure (Autonomous AI Database) 115
14.7.2.12 pyqScriptDrop Procedure (Autonomous AI Database) 118
14.7.3 Asynchronous Jobs (Autonomous AI Database) 118
14.7.3.1 oml_async_flag Argument 119
14.7.3.2 pyqJobStatus Function 120
14.7.3.3 pyqJobResult Function 121
14.7.3.4 Asynchronous Job Example 122
14.7.4 Special Control Arguments (Autonomous AI Database) 127
14.7.5 Output Formats (Autonomous AI Database) 129

15 Manage System Processes in OML4Py Using psutil

16 Python Classes to Convert Pretrained Models to ONNX Models (Deprecated)

Copyright © 2024, 2026, Oracle and/or its affiliates.

Index

Copyright © 2024, 2026, Oracle and/or its affiliates.

Preface

This publication describes Oracle Machine Learning for Python (OML4Py) and how to use it.

.

- Audience
- Related Resources
- Conventions
- Documentation Accessibility

Audience

This document is intended for those who want to run Python commands for statistical, machine learning, and graphical analysis on data stored in or accessible through Oracle Autonomous Database or Oracle Database on premises using a Python API. Use of Oracle Machine Learning for Python requires knowledge of Python and of Oracle Autonomous Database or Oracle Database on premises.

Related Resources

Related documentation is in the following publications:

- *Oracle Machine Learning for Python API Reference*
- *Oracle Machine Learning for Python Known Issues*
- *Oracle Machine Learning for Python Licensing Information User Manual*
- *REST API for Embedded Python Execution*
- Get Started with Notebooks for Data Analysis and Data Visualization in *Using Oracle Machine Learning Notebooks*
- Oracle Machine Learning AutoML User Interface
- *REST API for Oracle Machine Learning Services*

For more information, see these Oracle resources:

- Oracle Machine Learning Technologies
- Oracle Autonomous Database

Conventions

The following text conventions are used in this document:

Copyright © 2024, 2026, Oracle and/or its affiliates.

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Copyright © 2024, 2026, Oracle and/or its affiliates.

1

Changes in This Release

The following sections outline the changes in Oracle Machine Learning for Python User's Guide for Oracle AI Database 26ai.

- OML4Py 2.1.1 Updates
  The following updates have been made in OML4Py User's Guide Release 2.1.1 in Oracle AI Database version 23.26.1.
- OML4Py 2.1 Updates
  The following updates have been made in OML4Py User's Guide Release 2.1 in 26ai.
- OML4Py 2.0.1 Updates
  The following updates have been made in OML4Py User's Guide Release 2.0.1 in 26ai.
- OML4Py 2.0 Updates
  The following updates have been made in OML4Py User's Guide Release 2.0 in 26ai.
- Deprecated Features
  The following features are deprecated, and may be desupported in a future release.

1.1 OML4Py 2.1.1 Updates

The following updates have been made in OML4Py User's Guide Release 2.1.1 in Oracle AI Database version 23.26.1.

Features	Description
OML4Py Client Container	Enables you to use the OML4Py Client Container, which provides OML4Py features including ONNX model conversion and datastore export and import. See Install OML4Py Client Container for more information.
Large ONNX Model Conversion	Supports converting large pretrained ONNX models for use with Database version 23.26.1 or later. See Support For Large ONNX Format Model for more information.
Export or Import Datastore	Enables you to export or import Datastores. Requires OML4Py Server 2.1. See Import and Export Datastore for more information.

1.2 OML4Py 2.1 Updates

The following updates have been made in OML4Py User's Guide Release 2.1 in 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Features	Description
Install OML4Py 2.1 Window Client	You can install and use OML4Py 2.1 on Windows. To learn more, see Install OML4Py 2.1 on Windows
Vector series data class	You can use `oml.vector` vector type to manipulate database objects using the OML4Py transparency layer. To learn more, see Oracle Machine Learning for Python API Reference.
New types of ONNX pipeline	Import Pretrained Models in ONNX Format provide methods for : • Text Embedding - ONNX Pipeline Models: Text Embedding • Image Embedding - ONNX Pipeline Models: Image Embedding • Multi-modal Embedding - ONNX Pipeline Models: CLIP Multi-Modal Embedding • Text Classification - ONNX Pipeline Models: Text Classification • Reranking - ONNX Pipeline Models: Reranking Pipeline
ONNX algorithm	Allows you to bring your own ONNX-format model, load it in the database and score using existing data in the database. To learn more see, ONNX.
Export or Import Python scripts	Allows you to export or import your Python scripts. To learn more, see Export a User-Defined Python Function or Import a User-Defined Python Function.

1.3 OML4Py 2.0.1 Updates

The following updates have been made in OML4Py User's Guide Release 2.0.1 in 26ai.

Additional Changes:

- AutoML Pipeline: It automates the three major stages of the machine learning pipeline:
  - algorithm selection
  - adaptive data reduction (feature and sample size selection)
  - hyperparameter optimization

  To learn more, seeAutoML Pipeline.
- OML4Py Metrics: It provides metrics to evaluate the performance of machine learning models. To learn more, see OML4Py Metrics.
- The OML4Py client has been updated to the latest version, 2.0.1. To learn more, see Install OML4Py Client for Linux for On-Premises Oracle AI Database.
- The option to load custom models from the local filesystem is now available. To learn more, see Load Custom Models from The Local Filesystem.

Copyright © 2024, 2026, Oracle and/or its affiliates.

1.4 OML4Py 2.0 Updates

The following updates have been made in OML4Py User's Guide Release 2.0 in 26ai.

Algorithm Enhancements

> **Note**
>
> New Algorithm Settings: You can find model settings and algorithm specific settings in *Oracle AI Database PL/SQL Packages and Types Reference* guide. See Oracle Database PL/SQL Packages and Types Reference guide.

- **GLM link functions**

  `GLMS_LINK_FUNCTION`: this setting enables the user to specify the link function for building a generalized linear model. The additional link functions are: Logit, Probit, Cloglog, and Cauchit. See Generalized Linear Model.

- **XGBoost**

  The following new settings are added for XGBoost support for constraints and survival analysis.

  > **Note**
  >
  > The XGBoost settings are case sensitive.

  - **Interaction and Monotonic Constraints**
    - `xgboost_interaction_constraints`
    - `xgboost_decrease_constraints`
    - `xgboost_increase_constraints`
  - **Support for Survival Analysis**
    - `objective: survival:aft`
    - `xgboost_aft_loss_distribution`
    - `xgboost_aft_loss_distribution_scale`
    - `xgboost_aft_right_bound_column_name`

  Oracle Machine Learning supports XGBoost features such as monotonic and interaction constraints, as well as the AFT model for survival analysis. See XGBoost.

- **Explicit Semantic Analysis (ESA)**

  The following settings are added to support generate embeddings through Explicit Semantic Analysis embeddings:

  - `ESAS_EMBEDDINGS`: when enabled, generates embeddings during scoring for feature extraction models.
  - `ESAS_EMBEDDING_SIZE`: specifies the size of the vectors representing embeddings.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Supports embeddings for the Explicit Semantic Analysis (ESA) algorithm. ESA embeddings enables you to utilize ESA models to generate embeddings for any text or other ESA input. This functionality is equivalent to doc2vec (document to vector representation). See Explicit Semantic Analysis.

- **Expectation Maximization**

  `EMCS_OUTLIER_RATE`: identifies the frequency of outliers in the training data. See Expectation Maximization.

- **Exponential Smoothing Model**

  New settings for Exponential Smoothing to support Time Series regression models and initial value optimization for model build:

  – **Multiple time series**

    `EXSM_SERIES_LIST`:setting enables you to forecast up to twenty predictor series in addition to the target series.

  – **Automated model type search**

    `EXSM_INITVL_OPTIMIZE`: determines whether initial values are optimized during model build.

  Exponential Smoothing is enhanced to support building of multiple time series models and time series regression is possible with the multi-series build. The behavior of Exponential Smoothing is modified such that it searches for an acceptable time series model automatically. Enables the algorithm to select the best model type automatically when you do not specify `EXSM_MODEL` setting. This leads to more accurate forecasting. For details, see Exponential Smoothing Method.

- ***K*-Means**

  `KMNS_WINSORIZE`: this setting restricts the data in a window size of six standard deviations around the mean. See *k*-Means.

General Enhancements

- **New shared settings**

  – `ODMS_BOXCOX`: this setting enables the Box-Cox variance-stabilization transformation.

  – `ODMS_EXPLOSION_MIN_SUPP`: introduced more efficient data driven encoding for high cardinality categorical columns. You can define minimum support required for the categorical values in explosion mapping.

  See Shared Settings.

- **Convert Pretrained Models to ONNX Format**
  OML4Py enables the use of text transformers from Hugging Face by converting them into ONNX format models. OML4Py also adds the necessary tokenization and post-processing. The resulting ONNX pipeline is then imported into the database and can be used to generate embeddings for AI Vector Search. See ONNX Pipeline Models: Text Embedding.

- **Model Includes Data Lineage**

  In-database ML models now record the query string that was run to specify the build data within the model's metadata. The `build_source` parameter in the `all/user/dba_mining_models` view enables users to know the data query used to produce the model. See `ALL_MINING_MODELS`.

- **Improved Performance of Partitioned Models**

Copyright © 2024, 2026, Oracle and/or its affiliates.

Performance of partitioned models with high number of partitions and dropping individual models within partition model is improved. To know more about partitioned models, see DDL in Partitioned model.

- **4k Columns in Table**:
  The database tables can now accommodate up to 4,096 columns. This functionality is referred to as Wide Tables. To enable or disable Wide Tables for your Oracle database, you can use the `MAX_COLUMNS` parameter. See MAX_COLUMNS.
- **Support for in-database Machine Learning Algorithms**:
  Support for in-database machine learning Non-Negative Matrix Factorization algorithm, Exponential Smoothing Method algorithm and XGBoost algorithm. The following functions are new in the package that use in-database algorithms:
  - `oml.nmf`, Non-Negative Matrix Factorization model
  - `oml.esm`, Exponential Smoothing Method model
  - `oml.xgb`, XGboost model
- **Support for Data Types**:
  Support for data types that enable you to manipulate date, time and integer. The following data types are supported by OML4Py:
  - `oml.Datetime`, to create date.
  - `oml.Timezone`, to create time and timezone that includes `hour, minute, second, microsecond`, and `tzone`.
  - `oml.Timedelta`, to perform simple arithmetic operations.
  - `oml.Integer` to represent the integer data type.
- **Convert Pretrained Models to ONNX Format**:
  Oracle Machine Learning for Python supports ONNX format models. To learn more, see ONNX Pipeline Models: Text Embedding.

1.5 Deprecated Features

The following features are deprecated, and may be desupported in a future release.

Starting with Oracle AI Database 26ai, Release Update 23.7, the python packages EmbeddingModel and EmbeddingModelConfig are deprecated. These packages are replaced with ONNXPipeline and ONNXPipelineConfig respectively.

- Details and utility of the deprecated package can be found here : Python Classes to Convert Pretrained Models to ONNX Models (Deprecated).
- Oracle recommends that you use the latest version. You can find the details of the updated python classes in Import Pretrained Models in ONNX Format.

Copyright © 2024, 2026, Oracle and/or its affiliates.

2

About Oracle Machine Learning for Python

The following topics describe Oracle Machine Learning for Python (OML4Py) and its advantages for the Python user.

- What Is Oracle Machine Learning for Python
- Advantages of Oracle Machine Learning for Python
  Oracle Machine Learning for Python (OML4Py) provides a Python API to Oracle's in-database machine learning algorithms, enabling scalable analysis and modeling directly where the data resides. It minimizes data movement, ensures security, and streamlines workflows with AutoML, embedded Python execution, and in-database model storage and scoring.
- Manipulate database tables and views using familiar Python functions and syntax
  With the transparency layer classes, you can manipulate database tables and views using familiar Python functions and syntax, For example, using DataFrame proxy objects that map to database data, users can invoke overloaded Pandas functions that transparently generate SQL that runs in the database, using the database as a high-performance compute engine.
- About the Python Components and Libraries in OML4Py
  Oracle Machine Learning for Python (OML4Py) requires an installation of Python, the specified Python libraries, as well as the OML4Py components.

2.1 What Is Oracle Machine Learning for Python

Oracle Machine Learning for Python (OML4Py) provides a Python API to Oracle's in-database machine learning algorithms, enabling scalable, high-performance model building and scoring directly where the data resides. In addition, OML4Py supports data transformations and statistical, machine learning, and graphical analysis on data stored in or accessible through Oracle AI Database. You can run user-defined Python functions through database-spawned and controlled Python engines, with optional built-in data-parallelism and task-parallelism. This embedded execution functionality enables calling user-defined functions from SQL, and on Autonomous Database, from REST. OML4Py also supports Automated Machine Learning (AutoML) for algorithm and feature selection, and model tuning and selection. You can further augment the built-in functionality with third-party packages from the Python ecosystem. OML4Py is a Python module that enables Python users to manipulate data in database tables and views using Python syntax. OML4Py functions and methods transparently translate a select set of Python functions into SQL for in-database execution.

OML4Py is available in the following Oracle AI Database environments:

- OML4Py is available in the Python interpreter in Oracle Machine Learning Notebooks in your Oracle Autonomous Database. For more information, see Use the Python Interpreter in a Notebook Paragraph in *Using Oracle Machine Learning Notebooks*.
- An OML4Py client connection to OML4Py in an on-premises Oracle AI Database instance.

  For this environment, you must install Python, the required Python libraries, and the OML4Py server components in the database, and you must install the OML4Py client. See Install OML4Py for On-Premises Databases.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Designed for problems involving both large and small volumes of data, OML4Py integrates Python with the database. With OML4Py, you can do the following:

- Run overloaded Python functions and use native Python syntax to manipulate in-database data, without having to learn SQL.
- Use Automated Machine Learning (AutoML) to enhance user productivity and machine learning results through automated algorithm and feature selection, as well as model tuning and selection.
- Use Embedded Python Execution to run user-defined Python functions in Python engines spawned and managed by the database environment. The user-defined functions and data are automatically loaded to the engines as required, and when data-parallel and task-parallel execution is enabled. Develop, refine, and deploy user-defined Python functions and machine learning models that leverage the parallelism and scalability of the database to automate data preparation and machine learning.
- Use a natural Python interface to build in-database machine learning models.

2.2 Advantages of Oracle Machine Learning for Python

Oracle Machine Learning for Python (OML4Py) provides a Python API to Oracle's in-database machine learning algorithms, enabling scalable analysis and modeling directly where the data resides. It minimizes data movement, ensures security, and streamlines workflows with AutoML, embedded Python execution, and in-database model storage and scoring.

With OML4Py, you can do the following:

- **Operate on database data without using SQL**

  OML4Py transparently translates many standard Python functions into SQL. With OML4Py, you can create Python proxy objects that access, analyze, and manipulate data that resides in the database. OML4Py can automatically optimize the SQL by taking advantage of column indexes, query optimization, table partitioning, and database parallelism.

  OML4Py overloaded functions are available for many commonly used Python functions, including those on Pandas data frames for in-database execution.

  See Also: Manipulate database tables and views using familiar Python functions and syntax

- **Automate common machine learning tasks**

  By using Oracle's advanced Automated Machine Learning (AutoML) technology, both data scientists and beginner machine learning users can automate common machine learning modeling tasks such as algorithm selection and feature selection, and model tuning and selection, all of which leverage the parallel processing and scalability of the database.

  See Also: About Automated Machine Learning

- **Minimize data movement**

  By keeping data in the database whenever possible, you eliminate the time involved in transferring the data to your client Python engine and the need to store the data locally. You also eliminate the need to manage the locally stored data, which includes tasks such as distributing the data files to the appropriate locations, synchronizing the data with changes that are made in the production database, and so on.

  See Also: About Moving Data Between the Database and a Python Session

- **Keep data secure**

Copyright © 2024, 2026, Oracle and/or its affiliates.

By keeping the data in the database, you have the security, scalability, reliability, and backup features of the database for managing the data.

- **Use the power of the database**

  By operating directly on data in the database, you can use the memory and processing power of the database and avoid the memory constraints of your client Python engine.

- **Use current data**

  As data is refreshed in the database, you have immediate access to current data.

- **Save Python objects to a datastore in the database**

  You can save Python objects to an OML4Py datastore for future use and for use by others.

  See Also: About OML4Py Datastores

- **Build and store native Python models in the database**

  Using Embedded Python Execution, you can build native Python models and store and manage them in an OML4Py datastore.

  You can also build in-database models, with, for example, an `oml` class such as the Decision Tree class `oml.dt`. These in-database models have proxy objects that reference the actual models. Keeping with normal Python behavior, when the Python engine terminates, all in-memory objects, including models, are lost. To prevent an in-database model created using OML4Py from being deleted when the database connection is terminated, you must store its proxy object in a datastore.

  See Also: About Machine Learning Classes and Algorithms

- **Score data**

  For most of the OML4Py machine learning classes, you can use the `predict` and `predict_proba` methods of the model object to score new data.

  For these OML4Py in-database models, you can also use the SQL `PREDICTION` function on the model proxy objects, which scores directly in the database. You can use in-database models directly from SQL if you prepare the data properly. For open source models, you can use Embedded Python Execution and enable data-parallel execution for performance and scalability.

- **Run user-defined Python functions in embedded Python engines**

  Using OML4Py Embedded Python Execution, you can store user-defined Python functions in the OML4Py script repository, and run those functions in Python engines spawned by the database environment. When a user-defined Python function runs, the database starts, controls, and manages one or more Python engines that can run in parallel. With the Embedded Python Execution functionality, you can do the following:

  – Use a select set of Python packages in user-defined functions that run in embedded Python engines

  – Use other Python packages and third-party package in user-defined Python functions that run in embedded Python engines

  – Operationalize user-defined Python functions for use in production applications and eliminate porting Python code and models into SQL, and on ADB, REST; avoid reinventing code to integrate Python results into existing applications

  – Seamlessly leverage your Oracle AI Database as a high-performance computing environment for user-defined Python functions, providing data parallelism and resource management

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Perform parallel simulations, for example, Monte Carlo analysis, using the `oml.index_apply` function
- Generate JSON images, PNG images and XML representations of both structured and image data, which can be used by Python clients and SQL-based applications. PNG images and structured data can be used for Python clients and applications that use REST APIs.

See Also: About Embedded Python Execution

2.3 Manipulate database tables and views using familiar Python functions and syntax

With the transparency layer classes, you can manipulate database tables and views using familiar Python functions and syntax, For example, using DataFrame proxy objects that map to database data, users can invoke overloaded Pandas functions that transparently generate SQL that runs in the database, using the database as a high-performance compute engine.

The OML4Py transparency layer does the following:

- Enables creating tables and views from `pandas.DataFrame` and getting proxy objects to tables and views.
- Overloads specific Python functions that transparently translate functionality to SQL
- Leverages proxy objects for database data
- Uses familiar Python syntax to manipulate database data

The following table lists the transparency layer functions for getting and creating proxy objects and tables/views.

Table 2-1 Transparency Layer Functions for getting and creating proxy objects and tables/views

Function	Description
`oml.create`	Creates a table in a the database schema from a Python data set.
`oml_object``.pull`	Creates a local Python object that contains a copy of data fetched from database object referenced by the `oml` object.
`oml.push`	Pushes data from a Python session into an object in a database schema.
`oml.sync`	Creates a `DataFrame` proxy object in Python that represents a database table or view.
`oml.dir`	Return the names of `oml` objects in the Python session workspace.
`oml.drop`	Drops a persistent database table, view or in-database model.

Transparency layer proxy classes map SQL data types or objects to corresponding Python types. The classes provide Python functions and operators that are the same as those on the mapped Python types. The following table lists the transparency layer data type classes.

Table 2-2 Transparency Layer Data Type Classes

Class	Description
`oml.Boolean`	A boolean series data class that represents a single column of 0, 1, and NULL values in database data.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 2-2 (Cont.) Transparency Layer Data Type Classes

Class	Description
`oml.Bytes`	A binary series data class that represents a single column of `RAW` or `BLOB` database data types.
`oml.Float`	A numeric series data class that represents a single column of `NUMBER`, `BINARY_DOUBLE`, or `BINARY_FLOAT` database data types.
`oml.String`	A character series data class that represents a single column of `VARCHAR2`, `CHAR`, or `CLOB` database data types.
`oml.DataFrame`	A tabular `DataFrame` class that represents multiple columns of `oml.Boolean`, `oml.Bytes`, `oml.Float`, `oml.String`, and `oml.Vector` data.
`oml.Integer`	A data class that represents a single column of `NUMBER(*,0)` data in the database.
`oml.Datetime`	A series date class that represents a single column of `TIMESTAMP` or `TIMESTAMP WITH TIME ZONE` in Oracle Database. `oml.Timezone` A time class that is used with `oml.Datetime` to support `TIME STAMP WITH TIME ZONE`. `oml.Timedelta` A time class that represents a single column series of differences between two dates or times, or `INTERVAL DAY TO SECOND` in Oracle Database.
`oml.Timezone`	A time class that is used with `oml.Datetime` to support `TIME STAMP WITH TIME ZONE`.
`oml.Timedelta`	A time class that represents a single column series of differences between two dates or times, or `INTERVAL DAY TO SECOND` in Oracle Database.
`oml.Vector`	A vector series data class that represents a single column of `VECTOR` data in Oracle Database.

The following table lists the mappings of Python data types for both the reading and writing of data between Python and the database.

Table 2-3 Python and SQL Data Type Equivalencies

Database Read	Python Data Types	Database Write
N/A	`Bool`	If `oranumber == True`, then `NUMBER` (the default), else `BINARY_DOUBLE`.
`BLOB`	`bytes`	`BLOB`
`RAW`		`RAW`
`BINARY_DOUBLE`	`float`	If `oranumber == True`, then `NUMBER` (the default), else `BINARY_DOUBLE`.
`BINARY_FLOAT`		
`NUMBER`		
`CHAR`	`str`	`CHAR`
`CLOB`		`CLOB`
`VARCHAR2`		`VARCHAR2`
`NUMBER(*,0)`	`int`	`NUMBER(*,0)`
`TIMESTAMP` or `TIMESTAMP WITH TIME ZONE`	`datetime.datetime`	`TIMESTAMP` or `TIMESTAMP WITH TIME ZONE`
`TIMESTAMP WITH TIME ZONE`	`datetime.timezone`	`TIMESTAMP WITH TIME ZONE`
`INTERVAL DAY TO SECOND`	`datetime.timedelta`	`INTERVAL DAY TO SECOND`

Copyright © 2024, 2026, Oracle and/or its affiliates.

2.4 About the Python Components and Libraries in OML4Py

Oracle Machine Learning for Python (OML4Py) requires an installation of Python, the specified Python libraries, as well as the OML4Py components.

- In Oracle Autonomous Database, OML4Py is already installed. The OML4Py installation includes Python, additional required Python libraries, and the OML4Py server components. A Python interpreter is included with Oracle Machine Learning Notebooks in Autonomous Database.
- You can install third-party Python libraries in a conda environment through a conda interpreter for use within OML Notebooks sessions and OML4Py embedded execution invocations.
- You can install OML4Py in an on-premises Oracle AI Database. In this case, you must install Python, the additional required Python libraries, the OML4Py server components, and an OML4Py client. See Install OML4Py for On-Premises Databases.

Python Version in Current Release of OML4Py

To determine the current version of python used by OML4Py, run the following command:

```
import sys
sys.version
```

This version is in the current release of Oracle Autonomous Database.

Required Python Libraries

The following Python libraries must be included.

- `oracledb 2.2.0`
- `matplotlib 3.8.4`
- `numpy 1.26.4`
- `pandas 2.1.1`
- `scikit-learn 1.4.1.post1`
- `scipy 1.12.0`
- `six 1.13.0`

All the above libraries are included with Python in the current release of Oracle Autonomous Database.

For an installation of OML4Py in an on-premises Oracle AI Database, you must install Python and additionally the libraries listed here. See Install OML4Py for On-Premises Databases.

Copyright © 2024, 2026, Oracle and/or its affiliates.

3

Install OML4Py Client for Linux for Use With Autonomous AI Database Serverless

You can install and use the OML4Py client for Linux to work with OML4Py in an Oracle Autonomous AI Database on Serverless Exadata infrastructure.

OML4Py on premises runs on 64-bit platforms only. For supported platforms see Database and System Requirements for OML4Py .

The following instructions tell you how to download install Python, configure your environment, install manage your client credentials, install Oracle Instant Client, and install the OML4Py client:

1. Download the Python 3.12.0 source and untar it:

   ```
   wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tar.xz
   tar xvf Python-3.12.0.tar.xz
   ```

2. OML4Py requires the presence of the `perl-Env`, `libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel`, `libuuid-devel` and `ncurses-devel` libraries. Install these packages as sudo or root user:

   ```
   sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-
   devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel
   ```

3. To build Python, enter the following commands, where `PREFIX` is the directory in which you installed Python-3.12.0. Use `make altinstall` to avoid overriding the system default's Python installation.

   ```
   export PREFIX=`pwd`/Python-3.12.0
   cd $PREFIX
   ./configure --prefix=$PREFIX --enable-shared

   make clean; make
   make altinstall
   ```

4. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

   ```
   export PYTHONHOME=$PREFIX
   export PATH=$PYTHONHOME/bin:$PATH
   export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
   ```

   Create a symbolic link in your `$PYTHONHOME/bin` directory. You need to link it to yourPython 3.12.0 executable, which you can do with the following commands:

   ```
   cd $PYTHONHOME/bin
   ln -s python3.12 python3
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

You can now start Python with the `python3` script:

```
python3
```

pip will return warnings during package installation if the latest version is not installed. You can upgrade the version of pip to avoid these warnings:

```
python3 -m pip install --upgrade pip
```

5. Install the Oracle Instant Client for Autonomous AI Database, as follows:

   Download the Oracle Instant Client for your system. Go to the Oracle Instant Client Downloads page and select **Instant Client for Linux x86-64**. For more instruction see Install Oracle Instant Client for Linux for On-Premises Databases.

   For instruction on installing the Oracle instant client for on-premises see Install OML4Py Client for Linux for On-Premises Oracle AI Database.

   If you have root access to install an RPM on the client system. Alternatively, you can also download the zip file installer, unzip the file, and add the location of the unzipped file to LD_LIBRARY_PATH as done in next section.

   ```
   wget https://download.oracle.com/otn_software/linux/instantclient/1914000/
   oracle-instantclient19.14-basic-19.14.0.0.0-1.x86_64.rpm
   rpm -ivh oracle-instantclient19.14-basic-19.14.0.0.0-1.x86_64.rpm
   export LD_LIBRARY_PATH=/usr/lib/oracle/19.14/client64/lib:$LD_LIBRARY_PATH
   ```

   If you do not have root access to install an RPM on the client system.

   ```
   https://download.oracle.com/otn_software/linux/instantclient/2340000/
   instantclient-basic-linux.x64-23.4.0.24.05dbru.zip
   instantclient-basic-linux.x64-23.4.0.24.05dbru.zip
   export LD_LIBRARY_PATH=/path/to/instantclient_23_4:$LD_LIBRARY_PATH
   ```

6. Download the client credentials (wallet) from your Autonomous AI Database. Create a directory for the Wallet contents. Unzip the wallet zip file to the newly created directory:

   > **Note**
   >
   > An mTLS connection using the client Wallet is required. TLS connections are not currently supported.

   ```
   mkdir -p mywalletdir
   unzip Wallet.name.zip -d mywalletdir
   cd mywalletdir/
   ls
   ```

   ```
   README       ewallet.p12   ojdbc.properties  tnsnames.ora
   cwallet.sso  keystore.jks  sqlnet.ora        truststore.jks
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

7. Update `sqlnet.ora` with the wallet location. If you're working behind a proxy firewall, set the `SQLNET.USE_HTTPS_PROXY` environment variable to `on`:

```
WALLET_LOCATION = (SOURCE = (METHOD = file) (METHOD_DATA =
(DIRECTORY="mywalletdir")))
SSL_SERVER_DN_MATCH=yes
SQLNET.USE_HTTPS_PROXY=on
```

8. Add proxy address information to all service levels in `tnsnames.ora`, and add the connection pools for all service levels. If you are behind a firewall, enter the proxy address and port number to all service levels in `tnsnames.ora`. You will also need to add three new entries for the AutoML connection pools as shown below.

> **Note**
>
> If the proxy server contains a firewall to terminate connections within a set time period, the database connection will also be terminated.

For example, `myadb_medium_pool` is another alias for the connection string with `SERVER=POOLED` added to the corresponding one for `myadb_medium`.

```
myadb_low = (description= (retry_count=20)(retry_delay=3)
(address=(https_proxy=your proxy address here)(https_proxy_port=80)
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com))
(security=(ssl_server_cert_dn="CN=adb.us-sanjose-1.oraclecloud.com,OU=Oracle
ADB SANJOSE,O=Oracle Corporation,L=Redwood City,ST=California,C=US")))

myadb_medium = (description= (retry_count=20)(retry_delay=3)
(address=(https_proxy=your proxy address here)(https_proxy_port=80)
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com))
(security=(ssl_server_cert_dn="CN=adb.us-sanjose-1.oraclecloud.com,OU=Oracle
ADB SANJOSE,O=Oracle Corporation,L=Redwood City,ST=California,C=US")))

myadb_high = (description= (retry_count=20)(retry_delay=3)
(address=(https_proxy=your proxy address here)(https_proxy_port=80)
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com))
(security=(ssl_server_cert_dn="CN=adb.us-sanjose-1.oraclecloud.com,OU=Oracle
ADB SANJOSE,O=Oracle Corporation,L=Redwood City,ST=California,C=US")))

myadb_low_pool = (description= (retry_count=20)(retry_delay=3)
(address=(https_proxy=your proxy address here)(https_proxy_port=80)
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
City,ST=California,C=US")))

myadb_medium_pool = (description= (retry_count=20)(retry_delay=3)
(address=(https_proxy=your proxy address here)(https_proxy_port=80)
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
City,ST=California,C=US")))

myadb_high_pool = (description= (retry_count=20)(retry_delay=3)
(address=(https_proxy=your proxy address here)(https_proxy_port=80)
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
City,ST=California,C=US")))
```

9. Set `TNS_ADMIN` environment variable to the wallet directory:

   ```
   export TNS_ADMIN=mywalletdir
   ```

10. Install OML4Py library dependencies. The versions listed here are the versions Oracle has tested and supports. Create a file named `requirements.txt` containing the following content:

    ```
    oracledb 2.2.0
    matplotlib 3.8.4
    numpy 1.26.4
    pandas 2.1.1
    scikit-learn 1.4.1.post1
    scipy 1.12.0
    six 1.13.0
    ```

11. Install OML4Py client:

    Download OML4Py client installation zip file, go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network. For more instruction see Install OML4Py Client for Linux for On-Premises Databases

    ```
    unzip oml4py-client-linux-x86_64-2.0.zip
    perl -Iclient client/client.pl


    Oracle Machine Learning for Python 2.0 Client.

    Copyright (c) 2018, 2022 Oracle and/or its affiliates. All rights reserved.
    Checking platform .................. Pass
    Checking Python .................... Pass
    Checking dependencies .............. Pass
    Checking OML4P version ............. Pass
    Current configuration
      Python Version ................... 3.12.0
      PYTHONHOME ....................... /opt/Python-3.12.0
      Existing OML4P module version .... None

      Operation ........................ Install/Upgrade

    Proceed? [yes]
    Processing ./client/oml-2.0-cp312-cp312-linux_x86_64.whl
    Installing collected packages: oml
    Successfully installed oml-2.0
    ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Done
```

12. Start Python and load the `oml` library:

    ```
    python3
    import oml
    ```

13. Create a database connection. The OML client connects using the wallet. Set the `dsn` and `automl` arguments to the `tnsnames` alias in the wallet:

    ```
    oml.connect(user="oml_user", password="oml_user_password",
    dsn="myadb_medium", automl="myadb_medium_pool")
    ```

    To provide empty strings for the user and password parameters to connect without exposing your Oracle Machine Learning user credentials in clear text:

    ```
    oml.connect(user="", password="", dsn="myadb_medium",
    automl="myadb_medium_pool")
    ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

4

Install OML4Py for On-Premises Databases

The following topics explain the database and system requirements and the installation instructions for OML4Py with an on-premises Oracle Database.

Topics:

- Database and System Requirements for OML4Py
  Lists the Oracle AI Database and system requirements for Oracle Machine Learning for Python.
- Getting Started with OML4Py Installation
  The following topics provide step-by-step installation instructions for OML4Py and its different versions with an on-premises Oracle Database.

4.1 Database and System Requirements for OML4Py

Lists the Oracle AI Database and system requirements for Oracle Machine Learning for Python.

Table 4-1 Database and Systems Requirements for OML4Py

OML4Py Version	Oracle Autonomous Database Serverless		Oracle AI Database		
	Python Version	Oracle Autonomous AI Database Version	Python Version	Oracle AI Database Version	Operating System
2.1.1	3.13.5	26ai	3.13.5	26ai	Linux X64 (OL8, OL9)
2.1	3.12.6	26ai	3.12.6	26ai	• Linux X64 (OL8) • Windows Server 2022 x64 Standard • Windows 11 x64 Pro
2.0.1	3.12.3	26ai	3.12.3	26ai	Linux X64 (OL8)
2.0	NA	NA	3.12.9	26ai	Linux X64(OL10)
	3.12.0	26ai	3.12.0	26ai	Linux X64 (OL8)
	3.12.0	21c, 19c	3.12.0	21c, 19c	Linux X64 (OL7, OL8)
1.0	NA	NA	3.9.5	21c, 19c	Linux X64 (OL7, OL8)

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> Plugin violations have been observed after upgrading Oracle AI Database from version 21c to 26ai with the OML4Py configurations.

4.2 Getting Started with OML4Py Installation

The following topics provide step-by-step installation instructions for OML4Py and its different versions with an on-premises Oracle Database.

> **Note**
>
> The steps and third-party packages version differ between OML4Py version.

Topics:

- Install OML4Py 2.1.1 for On-Premises Databases
  The following topics tell how to install and uninstall the server and client components required for using OML4Py with an on-premisesOracle AI Database.
- Install OML4Py 2.1 for On-Premises Databases
  The following topics tell how to install and uninstall the server and client components required for using OML4Py 2.1 with an on-premises Oracle Database.
- Install OML4Py 2.0.1 for On-Premises Databases
  The following topics tell how to install and uninstall the server and client components required for using OML4Py 2.0.1 with an on-premises Oracle Database.
- Install OML4Py 2.0 for On-Premises Databases
  The following topics tell how to install and uninstall the server and client components required for using OML4Py 2.0 with an on-premises Oracle Database.

4.2.1 Install OML4Py 2.1.1 for On-Premises Databases

The following topics tell how to install and uninstall the server and client components required for using OML4Py with an on-premisesOracle AI Database.

Topics:

- Build and Install Python for Linux for On-Premises Databases
  Instructions for installing Python for Linux for an on-premises Oracle AI Database.
- Install the Required Supporting Packages for Linux for On-Premises Databases
  Both the OML4Py server and client installations for an on-premises Oracle AI Database require that you also install a set of supporting Python packages, as described below.
- Install OML4Py Server for On-Premises Oracle AI Database
  These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.
- Install OML4Py Client for Linux for On-Premises Oracle AI Database
  Instructions for installing and uninstalling the on-premises OML4Py client.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Install OML4Py Slim Client
  The OML4Py slim client is a slim version of the OML4Py client that includes only the ONNX model conversion feature. The following topics explain how to install the OML4Py slim client and verify a successful installation.

4.2.1.1 Build and Install Python for Linux for On-Premises Databases

Instructions for installing Python for Linux for an on-premises Oracle AI Database.

For 26ai, building Python is only necessary for the client. The database has Python 3.13.10 in `$ORACLE_HOME/python`.

> **Note**
>
> Since Python 3.13.10 is already installed in `$ORACLE_HOME/python`, you may skip steps 1 through 4.

> **Note**
>
> The feature supporting text transformers from Hugging Face by converting them into ONNX format models will only work on the OML4Py client. It is not supported on the OML4Py server.

Python 3.13.5 is required to install and use OML4Py 2.1.1 client.

These steps describe building and installing Python 3.13.5 for Linux.

1. Go to the Python website and download the **Gzipped source tarball**. The downloaded file name is `Python-3.13.5.tgz`

   ```
   wget https://www.python.org/ftp/python/3.13.5/Python-3.13.5.tgz
   ```

2. Create a directory, such as `$HOME/python`, and extract the contents to this directory:

   ```
   mkdir -p $HOME/python
   tar -xvzf Python-3.13.5.tgz --strip-components=1 -C $HOME/python
   ```

   The contents of the Gzipped source tarball will be copied directly to `$HOME/python`.

3. OML4Py requires the presence of the `perl-Env`, `libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel`, `libuuid-devel`, `ncurses-devel`, `openblas`, and `libsqlite3-dev` libraries.
   Install these packages as sudo or root user:

   ```
   sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel tcl-
   devel \
   xz-devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel
   openblas libsqlite3-dev
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

4. To build Python 3.13.5, navigate to the `$HOME/python` directory where Python-3.13.5 will be installed. Enter the following commands on the Oracle Machine Learning for Python server:

```
cd $HOME/python
./configure --enable-shared --prefix=$HOME/python

make clean; make
make altinstall
```

> **Note**
>
> Be sure to use the `--enable-shared` flag if you are going to use Embedded Python Execution; otherwise, using an Embedded Python Execution function results in an `extproc` error.

> **Note**
>
> Be sure to invoke `make altinstall` instead of `make install` to avoid overwriting the system Python.

5. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

```
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

> **Note**
>
> In order to use Python for OML4Py, the variables must be set, and these variables must appear before system Python in PATH and LD_LIBRARY_PATH.

You can now start Python by running the command `python3.13`.

```
python3.13


Python 3.13.5 (main, Dec 17 2025, 19:37:22) [GCC 8.5.0 20210514 (Red Hat
8.5.0-28.0.1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

To verify the directory where Python is installed, use the `sys.executable` command from the sys package. For example:

```
import sys
print(sys.executable)


/home/<user>/python/bin/python3
```

This returns the absolute path of the Python executable binary.

Topics:

- Commands Summary for Building and Installing Python for Linux for On-Premises Databases
  The commands used to build and install Python for Linux for On-Premises Databases are listed in the following example.

4.2.1.1.1 Commands Summary for Building and Installing Python for Linux for On-Premises Databases

The commands used to build and install Python for Linux for On-Premises Databases are listed in the following example.

Example 4-1 Build and Install Python for Linux for On-Premises Databases

```
wget https://www.python.org/ftp/python/3.12.6/Python-3.12.6.tgz

mkdir -p $HOME/python
tar -xvzf Python-3.12.6.tgz --strip-components=1 -C $HOME/python

sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-
devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel

cd $HOME/python
./configure --enable-shared --prefix=$HOME/python

make clean; make
make altinstall

export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$ORACLE_HOME/oml4py/modules

cd $HOME/python/bin
ln -s python3.12 python3
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.1.2 Install the Required Supporting Packages for Linux for On-Premises Databases

Both the OML4Py server and client installations for an on-premises Oracle AI Database require that you also install a set of supporting Python packages, as described below.

Installing required packages on OML4Py client machine

> **Note**
>
> `onnxruntime`, `onnxruntime-extensions`, `onnx`, `torch`, `transformers`, and `sentencepiece` are to be installed on the client only. `onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` support the ONNX conversion feature on the OML4Py client and should installed on the client only. All other packages are installed on both the client and server.

> **Note**
>
> `oracledb 2.2.0` or later version is required to invoke vector database SQL APIs in 26ai.

These steps outline how to install the required Python packages for an on-premises OML4Py client:

1. There are two options for installing the dependent supporting packages for the OML4Py client. The steps for each option are outlined below:
   - Option 1: Install Packages from oml4py-supporting-linux-x86_64-2.1.1.zip

> **Note**
>
> The `torch-2.9.0` package bundled within `oml4py-supporting-linux-x86_64-2.1.1.zip` requires the GNU C++ standard library (**libstdc++.so.6**) on your system to be built with **GLIBCXX_3.4.29** or newer.If this version is not available, you may encounter an error such as:
>
> ```
> ImportError: /lib64/libstdc++.so.6: version `GLIBCXX_3.4.29' not
> found (required by .../torch/lib/libtorch_python.so)
> ```
>
> On Linux, you can check which `GLIBCXX` versions are supported by running:
>
> ```
> nm -D /lib64/libstdc++.so.6 | grep GLIBCXX
> ```
>
> If your system does not include `GLIBCXX_3.4.29` or newer, resolve the issue by copying a compatible `libstdc++.so.6` to `$PYTHONHOME/lib`.The `torch-2.9.0` package includes a suitable version in `torch/lib/libstdc++.so.6`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

a. Download the file oml4py-supporting-linux-x86_64-2.1.1.zip. Unzip `oml4py-supporting-linux-x86_64-2.1.1.zip` into a folder named `supporting/`.

b. Create a file named `requirements.txt` with the following content:

```
pandas==2.2.3
setuptools==80.8.0
scipy==1.14.1
matplotlib==3.10.0
oracledb==3.3.0
scikit-learn==1.6.1
numpy==2.1.0
pyarrow==19.0.0
onnxruntime==1.20.0
onnxruntime-extensions==0.12.0
onnx==1.17.0
torch==2.9.0
transformers==4.56.1
sentencepiece==0.2.0
```

c. Install the supporting packages listed in `requirements.txt` by running:

```
pip3.13 install --target <install folder> --no-index --find-
links=supporting/ --upgrade -r requirements.txt
```

Replace `<install folder>` with your target directory.

- Option 2: Install Packages from PyPI Repository

a. Create a file named `requirements.txt` with the following content:

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.2.3
setuptools==80.8.0
scipy==1.14.1
matplotlib==3.10.0
oracledb==3.3.0
scikit-learn==1.6.1
numpy==2.1.0
pyarrow==19.0.0
onnxruntime==1.20.0
onnxruntime-extensions==0.14.0
onnx==1.18.0
torch==2.9.0
transformers==4.56.1
sentencepiece==0.2.1
```

b. Install the supporting packages listed in `requirements.txt` by running:

```
pip3.13 install --target <install folder> --upgrade -r
requirements.txt --only-binary=:all:
```

Replace `<install folder>` with your target directory.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Installing required packages on OML4Py server machine

On the OML4Py server machine, all these packages must be installed into `$ORACLE_HOME/oml4py/modules` so they can be detected by the Embedded Python Execution process.

These steps outline how to install the required Python packages for an on-premises OML4Py server:

1. Create a file named `requirements2.txt` containing the following content.

   ```
   pandas==2.2.3
   setuptools==70.0.0
   scipy==1.14.1
   matplotlib==3.10.0
   oracledb==2.4.1
   joblib==1.4.2
   scikit-learn==1.6.1
   numpy==2.1.0
   ```

2. Install the packages with `requirements2.txt`. Run the following command, specifying the target directory, `$ORACLE_HOME/oml4py/modules`:

   ```
   pip3.12 install -r requirements2.txt --target=$ORACLE_HOME/oml4py/modules
   ```

Verify the Package Installation

Start Python and then load the packages below to ensure they have been installed successfully:

```
python3.13
```

```
Python 3.13.5 (main, Dec 17 2025, 19:37:22) [GCC 8.5.0 20210514 (Red Hat
8.5.0-28.0.1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Load the below packages:

> **Note**
>
> The libraries `onnx`, `torch`, `onnxruntime-extensions`, `transformers`, and `sentencepiece` are installed on the client machine only.

```
import numpy
import pandas
import scipy
import matplotlib
import oracledb
import sklearn
import onnx
import torch
import onnxruntime_extensions
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
import transformers
import sentencepiece
```

If all the packages are installed successfully, no errors will be returned.

- Commands Summary for Installing Required Supporting Packages for Linux for On-Premises Databases
  The commands used for installing required supporting packages for linux for On-Premises databases are listed in the following example.

4.2.1.2.1 Commands Summary for Installing Required Supporting Packages for Linux for On-Premises Databases

The commands used for installing required supporting packages for linux for On-Premises databases are listed in the following example.

Example 4-2 Install required packages on OML4Py client machine

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.2.2
setuptools==70.0.0
scipy==1.14.0
matplotlib==3.8.4
oracledb==2.4.1
scikit-learn==1.5.1
numpy==2.0.1
onnxruntime==1.20.0
onnxruntime-extensions==0.12.0
onnx==1.17.0
torch==2.6.0
transformers==4.49.0
sentencepiece==0.2.0
```

Example 4-3 Install required packages on OML4Py server machine

```
pandas==2.2.2
setuptools==70.0.0
scipy==1.14.0
matplotlib==3.8.4
oracledb==2.4.1
joblib==1.3.2
scikit-learn==1.5.1
numpy==2.0.1
```

4.2.1.3 Install OML4Py Server for On-Premises Oracle AI Database

These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.

- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Upgrade the OML4Py Client for On-Premises Databases
  Instructions for upgrading the OML4Py client.
- Downgrade OML4Py Server for Linux on On-Premises Database
  To downgrade OML4Py server to version 2.0 on your on-premises Oracle AI Database 26ai, follow the steps below.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

4.2.1.3.1 Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai

Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.

You can install OML4Py by using a Python script included in your 26ai database or by using the Database Configuration Assistant (DBCA).

Install OML4Py Using a SQL Script

To install the on-premises OML4Py server, the following are required:

- A connection to the internet.
- OML4Py supporting packages. For instructions on installing the required supporting packages see Install the Required Supporting Packages for Linux for On-Premises Databases.
- Perl 5.8 or higher installed on your system.

> **Note**
>
> Perl requires the `perl-Env` package. You can install the package as root with the command `yum install perl-Env` .

To check for the existence of `perl-Env`, run the following command. The version will vary depending on your Operating System and version:

```
rpm -qa perl-Env
perl-Env-1.04-395.el8.noarch
```

- Write permission on the directories to which you download and install the server components.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> The following environment variables must be set up.

- Set environment variables: Set `PYTHONHOME` and add it to your `PATH`
- Set `ORACLE_HOME` and add it to your `PATH`
- Set `LD_LIBRARY_PATH`

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

To install the OML4Py server for Linux for an on-premises Oracle AI Database 26ai, run the server installation SQL script `pyqcfg.sql`.

> **Note**
>
> The OML4Py server needs to be installed on `CDB$ROOT` first, followed by installation on a PDB for Oracle AI Database 26ai.

1. The `pyqcfg.sql` script is under `$ORACLE_HOME/oml4py/server`. Change directory to `$ORACLE_HOME/oml4py/server`.

   ```
   cd $ORACLE_HOME/oml4py/server
   ```

2. At your operating system prompt, start SQL*Plus, connect to the `CDB$ROOT` and run the `pyqcfg.sql` script.

   ```
   sqlplus / as sysdba


   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
   Version 23.4.0.24.05

   Copyright (c) 1982, 2024, Oracle.  All rights reserved.


   Connected to:
   Oracle AI
                                      Database 26ai Enterprise Edition Release
   23.0.0.0.0 - Production
   Version 23.4.0.24.05
   ```

   To capture the log, spool the installation steps to an external file `install_root.txt`.

   ```
   SQL> spool install_root.txt
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Verify that you are currently connected to the CDB$ROOT container.

```
SQL> show con_name



CON_NAME
------------------------------
CDB$ROOT
```

Run the pyqcfg.sql script to install OML4Py server in the CDB$ROOT first.

```
SQL> @pyqcfg.sql SYSAUX TEMP

SQL> spool off;
```

where

- SYSAUX is the permanent tablespace for the PYQSYS schema.
- TEMP is the temporary tablespace for the PYQSYS schema.

Open the install_root.txt file to see if any errors occurred.

3. At your operating system prompt, start SQL*Plus, connect to your PDB and run the pyqcfg.sql script. To capture the log, spool the installation steps to an external file install_pdb.txt. The following example uses the PDB ORCLPDB and gives example values for the script arguments.

```
sqlplus / as sysdba

SQL> spool install_pdb.txt
```

Log into a PDB where you want to install OML4Py server.

```
SQL> alter session set container=ORCLPDB;
```

Run the pyqcfg.sql script to install OML4Py server in the PDB.

```
SQL> @pyqcfg.sql SYSAUX TEMP

SQL> spool off;
```

where

- SYSAUX is the permanent tablespace for the PYQSYS schema.
- TEMP is the temporary tablespace for the PYQSYS schema.

Open the install_pdb.txt file to see if any errors occurred

Verify the Server Installation

You can verify the database configuration of OML4Py as oracle user by doing the following:

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. On the OML4Py server database instance, start SQL*Plus as the OML user logging into the PDB, in this example, PDB1.

```
sqlplus oml_user/oml_user_password$ORCLPDB
```

2. Run the following command:

```
SELECT * FROM sys.pyq_config;
```

The expected output is as follows:

```
$ sqlplus / as sysdba;
SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 16:23:35 2024


Copyright (c) 1982, 2024, Oracle.  All rights reserved.


Connected to:
Oracle AI
                                    Database 26ai Enterprise Edition Release
23.0.0.0.0 - Production
Version 23.4.0.24.05

SQL> alter session set container=ORCLPDB;

Session altered.

SQL> select * from sys.pyq_config;

NAME
--------------------------------------------------------------------------------
-----
VALUE
--------------------------------------------------------------------------------
-----
PYTHONHOME
/u01/app/oracle/product/23.4.0.0/dbhome_1/python

PYTHONPATH
/u01/app/oracle/product/23.4.0.0/dbhome_1/oml4y/modules

VERSION
2.0


NAME
--------------------------------------------------------------------------------
-----
VALUE
--------------------------------------------------------------------------------
-----
PLATFORM
ODB
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
DSWLIST
oml.*;pandas.*;numpy.*;matplotlib.*;sklearn.*
```

3. To verify the installation of the OML4Py server for an on-premises database see Verify OML4Py Installation for On-Premises Database.

- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

4.2.1.3.1.1 Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases

The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

Example 4-4 Install OML4Py Server for On-Premises Oracle Databases

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH

cd $ORACLE_HOME/oml4py/server

$ sqlplus / as sysdba
SQL> spool install_root.txt
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;

SQL> alter session set container=ORCLPDB;
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;
```

4.2.1.3.2 Verify OML4Py Installation for On-Premises Database

Verify the installation of the OML4Py server and client components for an on-premises database.

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, service_name='myservice')
```

2. Create a user-defined Python function and store it in the OML4Py script repository.

```
oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
```

3. Call the user-defined function, using the `oml.do_eval` function.

```
res = oml.do_eval(func='TEST')
res


2
```

4. When you are finished testing, you can drop the test.

```
oml.script.drop("TEST")
```

4.2.1.3.3 Grant Users the Required Privileges for On-Premises Database

Instructions for granting the privileges required for using OML4Py with an on-premises database.

To use OML4Py (OML4Py), a user must have certain database privileges. To store and manage user-defined Python functions in the OML4Py script repository, a user must also have the PYQADMIN database role.

User Privileges

After installing the OML4Py server on an on-premises Oracle AI Database server, grant the following privileges to any OML4Py user.

- CREATE SESSION
- CREATE TABLE
- CREATE VIEW
- CREATE PROCEDURE
- CREATE MINING MODEL
- EXECUTE ON CTXSYS.CTX_DDL (required for using Oracle Text Processing capability in the algorithm classes in the `oml.algo` package)

To grant all of these privileges, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is the OML4Py user:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE,
CREATE MINING MODEL, EXECUTE ON CTXSYS.CTX_DDL to oml_user;
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Script Repository and Datastore Management

The OML4Py script repository stores user-defined Python functions that a user can invoke in an Embedded Python Execution function. An OML4Py datastore stores Python objects that can be used in subsequent Python sessions. A user-defined Python function in the script repository or a datastore can be available to any user or can be restricted for use by the owner only or by those granted access to it.

The OML4Py server installation script creates the PYQADMIN role in the database. A user must have that role to do the following:

- Store user-defined Python functions in the script repository.
- Drop user-defined Python function from the repository
- Grant or revoke permission to use a user-defined Python function in the script repository.
- Grant or revoke permission to use the objects in a datastore.

To grant this role to a user, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is your OML4Py user:

```
GRANT PYQADMIN to oml_user;
```

4.2.1.3.4 Create New Users for On-Premises Oracle AI Database

The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.

About the pyquser.sql Script

The `pyquser.sql` script is a component of the on-premises OML4Py server installation. The script is in the `server` directory of the installation. The sysdba privilege is required to run the script.

The `pyquser.sql` script grants the new user the required on-premises Oracle AI Database privileges and, optionally, grants the PYQADMIN database role. The PYQADMIN role is required for creating and managing scripts in the OML4Py script repository for use in Embedded Python Execution.

The `pyquser.sql` script takes the following five positional arguments:

- Username
- User's permanent tablespace
- User's temporary tablespace
- Permanent tablespace quota
- PYQADMIN role

When you run the script, it prompts you for a password for the user.

Create a New User

To use the `pyquser.sql` script, go the `server` subdirectory of the directory that contains the extracted OML4Py server installation files. Run the script as a database administrator.

The following examples use SQL*Plus and the sysdba user to run the `pyquser.sql` script.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 4-5 Creating New Users

This example creates the user oml_user with the permanent tablespace USERS with an unlimited quota, the temporary tablespace TEMP, and grants the PYQADMIN role to the oml_user.

```
sqlplus / as sysdba
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin



Enter value for password: <type your password>
```

For a pluggable database:

```
sqlplus / as sysdba
alter session set container=<PDBNAME>
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin
```

The output is similar to the following:

```
SQL> @pyquser.sql oml_user USERS TEMP unlimited pyqadmin
Enter value for password: welcome1
old   1: create user &&1 identified by &password
new   1: create user oml_user identified by welcome1
old   2: default tablespace &&2
new   2: default tablespace USERS
old   3: temporary tablespace &&3
new   3: temporary tablespace TEMP
old   4: quota &&4 on &&2
new   4: quota unlimited on USERS

User created.

old   4:     'create procedure, create mining model to &&1';
new   4:     'create procedure, create mining model to pyquser';
old   6:   IF lower('&&5') = 'pyqadmin' THEN
new   6:   IF lower('pyqadmin') = 'pyqadmin' THEN
old   7:     execute immediate 'grant PYQADMIN to &&1';
new   7:     execute immediate 'grant PYQADMIN to pyquser';

PL/SQL procedure successfully completed.
```

This example creates the user oml_user2 with 20 megabyte quota on the USERS tablespace, the temporary tablespace TEMP, and without the PYQADMIN role.

```
sqlplus / as sysdba
@pyquser.sql oml_user2 USERS TEMP 20M FALSE


Enter value for password: <type your password>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.1.3.5 Upgrade the OML4Py Client for On-Premises Databases

Instructions for upgrading the OML4Py client.

If you already have a previous version of the OML4Py client installed and would like to upgrade to the 2.1.1 client, run the following command:

```
perl -Iclient client/client.pl -i --dep-pkgs supporting
```

This script will automatically uninstall the old version of the OML4Py client, install any required supporting packages, and then install the OML4Py 2.1.1 client.

If you already have OML4Py 2.1 client installed and would like to upgrade to 2.1.1 client, do the following:

1. Uninstall OML4Py 2.1 client by following the instructions in Uninstall the OML4Py Client for On-Premises Databases.
2. Install OML4Py 2.1.1 client by following the instructions in Install OML4Py Client for Linux for On-Premises Databases.

4.2.1.3.6 Downgrade OML4Py Server for Linux on On-Premises Database

To downgrade OML4Py server to version 2.0 on your on-premises Oracle AI Database 26ai, follow the steps below.

To downgrade the OML4Py server to version 2.0 on your on-premises Oracle AI Database 26ai, follow the steps below.

1. Download and Extract the OML4Py Server Package: Download the oml4py-server-linux-x86_64-2.1.zip package. Navigate to the directory where the zip file was downloaded, then unzip the package:

   ```
   unzip oml4py-server-linux-x86_64-2.1.zip
   ```

2. Downgrade OML4Py Server Using a SQL Script: Locate the downgrade script at `server/pyqu0102000.sql`. Connect to your database as a privileged user (such as SYSDBA), and run the script in the `CDB$ROOT` container first:

   ```
   SQL> @@pyqd0201000.sql
   ```

   Next, log into the Pluggable Database (PDB) where you need to downgrade OML4Py Server. For example, if your PDB is named ORCLPDB:

   ```
   SQL> alter session set container=ORCLPDB;
   SQL> @@pyqd0201000.sql
   ```

3. Verify the Downgrade: Run the following SQL command to confirm that OML4Py Server has been downgraded to version 2.0:

   ```
   SQL> select VALUE from sys.pyq_config where NAME = 'VERSION';
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

The output should display:

```
VALUE
--------------------------------------------------------------------------------
-----
2.0
```

4.2.1.3.7 Uninstall the OML4Py Server from an On-Premises Database 26ai

Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

Uninstall the On-Premises OML4Py Server for Linux

To uninstall the on-premises OML4Py server for Linux, do the following:

1. Verify that the `PYTHONHOME` environment variable is set to the Python3.12 directory.

   ```
   echo $PYTHONHOME
   ```

2. Verify that `PYTHONPATH` environment variable is set to the directory in which the `oml` modules are installed.

   ```
   echo $PYTHONPATH
   ```

   If it is not set to the proper directory, set it.

   ```
   export PYTHONPATH=$ORACLE_HOME/oml4py/modules
   ```

3. Change directories to the directory containing the server installation zip file.

   ```
   cd $ORACLE_HOME/oml4py
   ```

4. Run the server uninstall script `pyquncfg.sql`, the script is under `$ORACLE_HOME/oml4py/server`.

   ```
   cd $ORACLE_HOME/oml4py/server
   ```

   Start SQL*Plus, connect to the PDB.

   ```
   $ sqlplus / as sysdba;


   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
   Version 23.4.0.24.05

   Copyright (c) 1982, 2024, Oracle. All rights reserved.


   Connected to:
   Oracle AI
                                       Database 26ai Enterprise Edition Release
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
23.0.0.0.0 - Production
Version 23.4.0.24.05
```

Run the `pyquncfg.sql` script to record the log, spool, and the installation steps to an external file.

```
SQL> spool install.txt
SQL> alter session set container=ORCLPDB;
SQL> @pyquncfg.sql
```

4.2.1.4 Install OML4Py Client for Linux for On-Premises Oracle AI Database

Instructions for installing and uninstalling the on-premises OML4Py client.

For instructions on installing the OML4Py client on Oracle Autonomous AI Database, see Install OML4Py Client for Linux for Use With Autonomous AI Database Serverless.

To use the OML4Py client, users will need to install the following:

> **Note**
>
> The supporting packgaes need to be installed prior to the OML4Py client installation.

1. Install Python from source. See Build and Install Python for Linux for On-Premises Databases.
2. Install the Oracle Instant Client. See Install Oracle Instant Client for Linux for On-Premises Databases.
3. Install the OML4Py supporting packages. See Install the Required Supporting Packages for Linux for On-Premises Databases.
4. Install the OML4Py client. See Install OML4Py Client for Linux for On-Premises Databases.

Topics:

- Install Oracle Instant Client and the OML4Py Client for Linux
  Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.
- Verify OML4Py Client Installation for On-Premises Databases
  Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.
- Uninstall the OML4Py Client for On-Premises Databases
  Instructions for uninstalling the OML4Py client.
- Upgrade the OML4Py Client for On-Premises Databases
  Instructions for upgrading the OML4Py client.

4.2.1.4.1 Install Oracle Instant Client and the OML4Py Client for Linux

Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.

Copyright © 2024, 2026, Oracle and/or its affiliates.

To connect the OML4Py client for Linux to an on-premises Oracle AI Database, you must have Oracle Instant Client installed on your local system.

- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

4.2.1.4.1.1 Install Oracle Instant Client for Linux for On-Premises Databases

Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.

The OML4Py client requires Oracle Instant Client to connect to an Oracle AI Database.

To install Oracle Instant Client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.

To install Oracle Instant Client, do the following:

1. Download the Oracle Instant Client for your system. Go to the Oracle Instant Client Downloads page and select Instant Client for Linux x86-64.

Copyright © 2024, 2026, Oracle and/or its affiliates.

2. Locate the section for your version of Oracle AI Database. These instructions use Version 23.26.0.0.0.
3. In the Base section, in the Download column, click the zip file for the Basic Package and save the file in an accessible directory on your system or use `wget` to download the file:

   ```
   wget https://download.oracle.com/otn_software/linux/instantclient/2326000/
   instantclient-basic-linux.x64-23.26.0.0.0.zip
   ```

4. Unzip the package:

   ```
   unzip instantclient-basic-linux.x64-23.26.0.0.0.zip
   ```

   Extracting the package creates the subdirectory `instantclient_23_26`, which contains the Oracle Instant Client files.
5. To install the libaio package with sudo or as the root user, run the following command:

   ```
   sudo yum install libaio
   ```

   > **Note**
   >
   > In some Linux distributions, this package is called `libaio1`.

6. Add the directory that contains the Oracle Instant Client files to the beginning of your `LD_LIBRARY_PATH` environment variable:

   ```
   export LD_LIBRARY_PATH=/path/to/dir/instantclient_23_26:$LD_LIBRARY_PATH
   ```

4.2.1.4.1.2 Install OML4Py Client for Linux for On-Premises Databases

Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.

Prerequisites

To download and install the on-premises OML4Py client, the following are required:

- Write permission on the directory in which you are installing the client.
- Perl 5.8 or higher installed on your system.
- Python 3.13.5. To know more about downloading and installing Python 3.13.5, see Build and Install Python for Linux for On-Premises Databases
- The OML4Py supporting packages. To know more about OML4Py the supporting packages, see Install the Required Supporting Packages for Linux for On-Premises Databases.

To use the OML4Py client to connect to an on-premises Oracle AI Database, the following are required:

- Oracle Instant Client must be installed on the client machine.
- The OML4Py server must be installed on the on-premises database server.

Download and Extract the OML4Py Client Installation File

Copyright © 2024, 2026, Oracle and/or its affiliates.

To download and extract the OML4Py client installation file, do the following:

1. Download the client installation zip file.
   a. Go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network.
   b. Accept the license agreement and select Oracle Machine Learning for Python Downloads (v2.1.1).
   c. Select Oracle Machine Learning for Python Client Install for Oracle AI Database on Linux 64 bit.
   d. Save the zip file to an accessible directory.
2. Go to the directory to which you downloaded the zip file and unzip the file.

   ```
   oml4py-client-linux-x86_64-2.1.1.zip
   ```

   The contents are extracted to a subdirectory named `client`, which contains these five files:

   - `client.pl`
   - `OML4PInstallShared.pm`
   - `client/ oml-2.1.1-cp313-cp313-linux_x86_64.whl`
   - `client/slim/oml-2.1.1-cp313-cp313-linux_x86_64.whl`
   - `client/oml4py.ver`
3. Run the following command to install the OML4Py client package:

   ```
   pip3 install client/oml-2.1.1-cp313-cp313-linux_x86_64.whl
   ```

View the Optional Arguments to the Client Installation Perl Script

In the directory above the unzipped client folder, run the client installation Perl script with the `--help` option to display the arguments to the client installation Perl script.

The following command displays the available installation options:

```
perl -Iclient client/client.pl --help

Oracle Machine Learning for Python 2.1.1 Client.

Copyright (c) 2018, 2024 Oracle and/or its affiliates. All rights reserved.
Usage: client.pl [OPTION]...
Install, upgrade, or uninstall OML4P Client.

  -i, --install install or upgrade (default)
  -u, --uninstall uninstall
  -y never prompt
  --ask interactive mode (default)
  --no-embed do not install embedded python functionality
  --no-deps turn off dependencies checking
  --target <dir> install client into <dir>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

By default, the installation script installs the Embedded Python Execution modules. If you don't want to install this module, then you can use the `--no-embed` flag.

Also by default, the installation script checks for the existence and version of each of the supporting packages that the OML4Py client requires. If a required package is missing or does not meet the version requirement, the installation script displays an error message and exits. You can skip the dependency checking in the client installation by using the `--no-deps` flag. However, to use the `oml` module, you need to have installed acceptable versions of all of the supporting packages.

For a list of the required dependencies, see Install the Required Supporting Packages for Linux for On-Premises Databases.

Run the OML4Py Client Installation Script

To install the OML4Py client, do the following:

1. In the directory above the unzipped client folder, run the script. The following command runs the Perl script in the current directory:

   ```
   perl -Iclient client/client.pl
   ```

2. To install the OML4Py Client for Linux, first check that you have already installed the required dependencies (`numpy 2.1.0`, `pandas 2.2.3`, `scipy 1.14.1`, `matplotlib 3.10.0`, `oracledb 3.3.0`, `scikit_learn 1.6.1`, and `packaging 24.2`) . If these are installed, run:

   ```
   perl -Iclient client/client.pl -i
   ```

   If you need to install both OML4Py and its dependencies, unzip `oml4py-supporting-linux-x86_64-2.1.1.zip` into a folder named `supporting`, then run:

   ```
   perl -Iclient client/client.pl -i --dep-pkgs supporting
   ```

3. To verify that `oml` modules are successfully installed and are ready to use, start Python and import `oml`. At the Linux prompt, enter `python3.13`.

   ```
   python3.13
   ```

   At the Python prompt, enter `import oml` and check the client version.

   ```
   import oml
   oml.__version__
   ```

   The output is similar to the following:

   ```
   $ python3.13
   Python 3.13.5 (main, Dec 17 2025,19:37:22)
   [GCC 8.5.0 20210514 (Red Hat 8.5.0-28.0.1)] on linuxType "help",
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
"copyright",
"credits" or "license" for more information.


import oml
oml.__version__


'2.1.1'
```

4. Display the location of the installation directory.
   If you didn't use the `--target <dir>` argument, then the installed `oml` modules are stored under `$PYTHONHOME/lib/python3.13/site-packages/`. Again, you must have write permission for the target directory.

   In Python, after importing the `oml` module, you can display the directory in which the client is installed. At the Python prompt, enter:

   ```
   oml.__path__
   ```

Connect to the OML4Py Server

Start Python, import `oml`, and create a connection to your OML4Py server using an appropriate password, hostname, and system identifier. The following example uses oml_user as the user and has example argument values. Replace the username and other argument values with the values for your user and database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host=myhost,
            port=1521, service_name=myservicename)
```

After connecting, you can run any of the examples in this publication. For example, you could run Example 8-8.

> **Note**
>
> To use the Embedded Python Execution examples, you must have installed the OML4Py client with the Embedded Python Execution option enabled.
> To use the Automatic Machine Learning (AutoML) examples, you must specify a running connection pool on the server in the `automl` argument in an `oml.connect` invocation.

4.2.1.4.1.3 Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database

Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.

Example 4-6 Command Summary for Installing Oracle Instant Client

```
cd /opt/oracle
unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
sudo yum install libaio
export LD_LIBRARY_PATH=/opt/oracle/instantclient_23_04:$LD_LIBRARY_PATH
```

Example 4-7 Command Summary for Installing OML4Py Client

```
unzip v1048628-01.zip
perl -Iclient client/client.pl
```

4.2.1.4.2 Verify OML4Py Client Installation for On-Premises Databases

Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

   ```
   import oml
   oml.connect(user='oml_user', password='oml_user_password', host='myhost',
               port=1521, service_name=myservicename)
   ```

2. Create a user-defined Python function and store it in the OML4Py script repository.

   ```
   oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
   ```

3. Call the user-defined function, using the `oml.do_eval` function.

   ```
   res = oml.do_eval(func='TEST')
   res
   ```

   ```
   2
   ```

4. When you are finished testing, you can drop the test.

   ```
   oml.script.drop("TEST")
   ```

4.2.1.4.3 Uninstall the OML4Py Client for On-Premises Databases

Instructions for uninstalling the OML4Py client.

Uninstall the On-Premises OML4Py Client for Linux

To uninstall the on-premises OML4Py client for Linux, from the directory containing the client installation zip file, run the client installation Perl script with the `-u` argument:

```
perl -Iclient client/client.pl -u
```

When the script displays `Proceed?`, enter `y` or `yes`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

If the client is successfully uninstalled, you'll see the following message:

```
Uninstalling oml-2.1:
  Successfully uninstalled oml-2.1
```

4.2.1.4.4 Upgrade the OML4Py Client for On-Premises Databases

Instructions for upgrading the OML4Py client.

If you already have a previous version of the OML4Py client installed and would like to upgrade to the 2.1.1 client, run the following command:

```
perl -Iclient client/client.pl -i --dep-pkgs supporting
```

This script will automatically uninstall the old version of the OML4Py client, install any required supporting packages, and then install the OML4Py 2.1.1 client.

If you already have OML4Py 2.1 client installed and would like to upgrade to 2.1.1 client, do the following:

1. Uninstall OML4Py 2.1 client by following the instructions in Uninstall the OML4Py Client for On-Premises Databases.
2. Install OML4Py 2.1.1 client by following the instructions in Install OML4Py Client for Linux for On-Premises Databases.

4.2.1.5 Install OML4Py Slim Client

The OML4Py slim client is a slim version of the OML4Py client that includes only the ONNX model conversion feature. The following topics explain how to install the OML4Py slim client and verify a successful installation.

Topics:

- Install OML4Py Slim Client Modules
  Install the OML4Py Slim Client Modules to access only the ONNX model conversion feature. This lightweight client is ideal when you need ONNX model conversion without the full set of OML4Py functionalities.
- Verify the Installation of OML4Py Slim Client
  Verify the installation of the OML4Py slim client.

4.2.1.5.1 Install OML4Py Slim Client Modules

Install the OML4Py Slim Client Modules to access only the ONNX model conversion feature. This lightweight client is ideal when you need ONNX model conversion without the full set of OML4Py functionalities.

Starting with version 2.1.1, OML4Py offers a lightweight OML4Py client that supports only ONNX model conversion.

> **Note**
>
> If you plan to install both the full and slim versions of the OML4Py client, make sure to use separate `<install_folder>` directories for each version. Set the `PYTHONPATH` environment variable so that only one version is loaded at a time.

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. There are two options for installing the supporting packages for the OML4Py Slim Client:
   - Option 1: Install from the **oml4py-supporting-linux-x86_64-2.1.1.zip** archive.

     > **Note**
     >
     > The `torch-2.9.0` package bundled within `oml4py-supporting-linux-x86_64-2.1.1.zip` requires the GNU C++ standard library (**libstdc++.so.6**) on your system to be built with **GLIBCXX_3.4.29** or newer.If this version is not available, you may encounter an error such as:
     >
     > ```
     > ImportError: /lib64/libstdc++.so.6: version `GLIBCXX_3.4.29' not
     > found (required by .../torch/lib/libtorch_python.so)
     > ```
     >
     > On Linux, you can check which `GLIBCXX` versions are supported by running:
     >
     > ```
     > nm -D /lib64/libstdc++.so.6 | grep GLIBCXX
     > ```
     >
     > If your system does not include `GLIBCXX_3.4.29` or newer, resolve the issue by copying a compatible `libstdc++.so.6` to `$PYTHONHOME/lib`.The `torch-2.9.0` package includes a suitable version in `torch/lib/libstdc++.so.6`.

     Create a file named `requirements-slim.txt` with the following content:

     ```
     numpy==2.1.0
     pillow==10.4.0
     onnxruntime==1.20.0
     onnxruntime-extensions==0.12.0
     onnx==1.17.0
     torch==2.9.0
     transformers==4.56.1
     sentencepiece==0.2.0
     ```

     After unzipping the archive, install the supporting packages listed in `requirements-slim.txt` by running:

     ```
     pip3.13 install --target <install_folder> --no-index --find-
     links=supporting/ --upgrade -r requirements-slim.txt
     ```

     Replace `<install_folder>` with your installation directory.

   - Option 2: Install from the PyPI repository.
     Create a file named `requirements-slim.txt` with the following contents:

     ```
     --extra-index-url https://download.pytorch.org/whl/cpu
     numpy==2.1.0
     pillow==10.4.0
     onnxruntime==1.20.0
     onnxruntime-extensions==0.14.0
     onnx==1.18.0
     torch==2.9.0
     ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
transformers==4.56.1
sentencepiece==0.2.1
```

Install the supporting packages with `requirements-slim.txt`.

```
pip3.13 install --target <install folder> --upgrade -r requirements-
slim.txt –-only-binary:all:
```

2. Next, install the OML4Py Slim Client:

   Use the following wheel file from the **oml4py-client-linux-x86_64-2.1.1.zip** archive:

   `client/slim/oml-2.1.1-cp313-cp313-linux_x86_64.whl`
   Run the following command to install:

   ```
   pip3.13 install --target <install_folder> --no-index --upgrade --no-deps
   client/slim/oml-2.1.1-cp313-cp313-linux_x86_64.whl
   ```

   Replace `<install_folder>` with your desired installation directory.

4.2.1.5.2 Verify the Installation of OML4Py Slim Client

Verify the installation of the OML4Py slim client.

To verify that the OML4Py slim client is installed correctly, you can test its key features—such as loading ONNX model conversion utilities.

The following example demonstrates how to load ONNX model conversion methods from `oml.utils`, configure a pipeline, and export models as ONNX files:

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig, MiningFunction

# Initialize the pipeline with a pre-trained model
em = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2")
em.export2file("your_preconfig_file_name", output_dir="my_dir")

# Configure the pipeline using a template and export a different model
config = ONNXPipelineConfig.from_template("text", max_seq_length=512)
em = ONNXPipeline(model_name="intfloat/e5-small-v2", config=config)
em.export2file("your_template_file_name", output_dir="my_dir")
```

If you are able to run the code successfully, your OML4Py slim client installation is verified for ONNX model conversion.

4.2.2 Install OML4Py 2.1 for On-Premises Databases

The following topics tell how to install and uninstall the server and client components required for using OML4Py 2.1 with an on-premises Oracle Database.

Topics:

- Build and Install Python for Linux for On-Premises Databases
  Instructions for installing Python for Linux for an on-premises Oracle AI Database.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Install the Required Supporting Packages for Linux for On-Premises Databases
  Both the OML4Py server and client installations for an on-premises Oracle AI Database require that you also install a set of supporting Python packages, as described below.
- Install OML4Py Server for On-Premises Oracle Database
  These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.
- Install OML4Py Client for Linux for On-Premises Oracle AI Database
  Instructions for installing and uninstalling the on-premises OML4Py client.
- Install OML4Py Client for Windows for On-Premises and Autonomous Oracle AI Database
  Install OML4Py client 2.1 components required for using OML4Py Client on Windows.

4.2.2.1 Build and Install Python for Linux for On-Premises Databases

Instructions for installing Python for Linux for an on-premises Oracle AI Database.

For 26ai, building Python is only necessary for the client. The database has Python 3.12.6 in `$ORACLE_HOME/python`.

> **Note**
>
> Since Python 3.12.6 is already installed in `$ORACLE_HOME/python`, you may skip steps 1 through 4.

> **Note**
>
> The feature supporting text transformers from Hugging Face by converting them into ONNX format models will only work on the OML4Py client. It is not supported on the OML4Py server.

Python 3.12.6 is required to install and use OML4Py 2.1 client.

These steps describe building and installing Python 3.12.6 for Linux.

1. Go to the Python website and download the **Gzipped source tarball**. The downloaded file name is `Python-3.12.6.tgz`

   ```
   wget https://www.python.org/ftp/python/3.12.6/Python-3.12.6.tgz
   ```

2. Create a directory, such as `$HOME/python`, and extract the contents to this directory:

   ```
   mkdir -p $HOME/python
   tar -xvzf Python-3.12.6.tgz --strip-components=1 -C $HOME/python
   ```

   The contents of the Gzipped source tarball will be copied directly to `$HOME/python`.

3. OML4Py requires the presence of the `perl-Env`, `libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel`, `libuuid-devel` and `ncurses-devel` libraries.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Install these packages as sudo or root user:

```
sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-
devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel
```

4. To build Python 3.12.6, navigate to the `$HOME/python` directory where Python-3.12.6 will be installed. Enter the following commands on the Oracle Machine Learning for Python server:

```
cd $HOME/python
./configure --enable-shared --prefix=$HOME/python

make clean; make
make altinstall
```

> **Note**
>
> Be sure to use the `--enable-shared` flag if you are going to use Embedded Python Execution; otherwise, using an Embedded Python Execution function results in an `extproc` error.

> **Note**
>
> Be sure to invoke `make altinstall` instead of `make install` to avoid overwriting the system Python.

5. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

```
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

> **Note**
>
> In order to use Python for OML4Py, the variables must be set, and these variables must appear before system Python in PATH and LD_LIBRARY_PATH.

6. Create a symbolic link in your `$HOME/python/bin` directory to link to your python3.12 executable, which you can do with the following commands:

```
cd $HOME/python/bin
ln -s python3.12 python3
```

pip will return warnings during package installation if the latest version is not installed. You can upgrade the version of pip to avoid these warnings:

```
python3 -m pip install --upgrade pip
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

You can now start Python by running the command `python3`. To verify the directory where Python is installed, use the `sys.executable` command from the `sys` package. For example:

```
python3


Python 3.12.6 (main, Aug 14 2024, 15:13:51) [GCC 8.5.0 20210514 (Red Hat
8.5.0-18.0.6)] on linux
Type "help", "copyright", "credits" or "license" for more information.


import sys
print(sys.executable)


/home/<user>/python/bin/python3
```

This returns the absolute path of the Python executable binary.

If you run the command `python3` and you get the error `command not found`, then that means the system cannot find an executable named `python3` in `$PYTHONHOME/bin`. A symlink is required for the OML4Py server installation components. So, in that case, you need to create a symbolic link in your `$HOME/python/bin` directory to link to your `python3.12` executable as described in Step 7.

Topics:

4.2.2.2 Install the Required Supporting Packages for Linux for On-Premises Databases

Both the OML4Py server and client installations for an on-premises Oracle AI Database require that you also install a set of supporting Python packages, as described below.

Installing required packages on OML4Py client machine

> **Note**
>
> `onnxruntime`, `onnxruntime-extensions`, `onnx`, `torch`, `transformers`, and `sentencepiece` are to be installed on the client only. `onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` support the ONNX conversion feature on the OML4Py client and should installed on the client only. All other packages are installed on both the client and server.

> **Note**
>
> `oracledb 2.2.0` or later version is required to invoke vector database SQL APIs in 23ai.

These steps outline how to install the required Python packages for an on-premises OML4Py client:

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. Create a file named `requirements.txt` containing the following content:

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.2.2
setuptools==70.0.0
scipy==1.14.0
matplotlib==3.8.4
oracledb==2.4.1
scikit-learn==1.5.1
numpy==2.0.1
onnxruntime==1.20.0
onnxruntime-extensions==0.12.0
onnx==1.17.0
torch==2.6.0
transformers==4.49.0
sentencepiece==0.2.0
```

2. Install the packages with `requirements.txt`.

```
pip3.12 install -r requirements.txt
```

Installing required packages on OML4Py server machine

On the OML4Py server machine, all these packages must be installed into `$ORACLE_HOME/oml4py/modules` so they can be detected by the Embedded Python Execution process.

These steps outline how to install the required Python packages for an on-premises OML4Py server:

1. Create a file named `requirements2.txt` containing the following content.

```
pandas==2.2.2
setuptools==70.0.0
scipy==1.14.0
matplotlib==3.8.4
oracledb==2.4.1
joblib==1.3.2
scikit-learn==1.5.1
numpy==2.0.1
```

2. Install the packages with `requirements2.txt`. Run the following command, specifying the target directory, `$ORACLE_HOME/oml4py/modules`:

```
pip3.12 install -r requirements2.txt --target=$ORACLE_HOME/oml4py/modules
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Verify the Package Installation

Start Python and then load the packages below to ensure they have been installed successfully:

```
python3
```

```
Python 3.12.6 (main, Aug 14 2024, 15:13:51) [GCC 8.5.0 20210514 (Red
Hat 8.5.0-18.0.6)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Load the below packages:

> **Note**
>
> The libraries `onnx`, `torch`, `onnxruntime-extensions`, `transformers`, and `sentencepiece` are installed on the client machine only.

```
import numpy
import pandas
import scipy
import matplotlib
import oracledb
import sklearn
import onnx
import torch
import onnxruntime_extensions
import transformers
import sentencepiece
```

If all the packages are installed successfully, no errors will be returned.

4.2.2.3 Install OML4Py Server for On-Premises Oracle Database

These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.

- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.
- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.
- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

4.2.2.3.1 Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai

Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.

You can install OML4Py by using a Python script included in your 26ai database or by using the Database Configuration Assistant (DBCA).

Install OML4Py Using a SQL Script

To install the on-premises OML4Py server, the following are required:

- A connection to the internet.
- OML4Py supporting packages. For instructions on installing the required supporting packages see Install the Required Supporting Packages for Linux for On-Premises Databases.
- Perl 5.8 or higher installed on your system.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> Perl requires the `perl-Env` package. You can install the package as root with the command `yum install perl-Env` .

To check for the existence of `perl-Env`, run the following command. The version will vary depending on your Operating System and version:

```
rpm -qa perl-Env
perl-Env-1.04-395.el8.noarch
```

- Write permission on the directories to which you download and install the server components.

> **Note**
>
> The following environment variables must be set up.

- Set environment variables: Set `PYTHONHOME` and add it to your `PATH`
- Set `ORACLE_HOME` and add it to your `PATH`
- Set `LD_LIBRARY_PATH`

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

To install the OML4Py server for Linux for an on-premises Oracle AI Database 26ai, run the server installation SQL script `pyqcfg.sql`.

> **Note**
>
> The OML4Py server needs to be installed on `CDB$ROOT` first, followed by installation on a PDB for Oracle AI Database 26ai.

1. The `pyqcfg.sql` script is under `$ORACLE_HOME/oml4py/server`. Change directory to `$ORACLE_HOME/oml4py/server`.

   ```
   cd $ORACLE_HOME/oml4py/server
   ```

2. At your operating system prompt, start SQL*Plus, connect to the `CDB$ROOT` and run the `pyqcfg.sql` script.

   ```
   sqlplus / as sysdba


   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
   Version 23.4.0.24.05
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Copyright (c) 1982, 2024, Oracle.  All rights reserved.


Connected to:
Oracle AI
                                    Database 26ai Enterprise Edition Release
23.0.0.0.0 - Production
Version 23.4.0.24.05
```

To capture the log, spool the installation steps to an external file `install_root.txt`.

```
SQL> spool install_root.txt
```

Verify that you are currently connected to the `CDB$ROOT` container.

```
SQL> show con_name



CON_NAME
------------------------------
CDB$ROOT
```

Run the `pyqcfg.sql` script to install OML4Py server in the `CDB$ROOT` first.

```
SQL> @pyqcfg.sql SYSAUX TEMP

SQL> spool off;
```

where

- `SYSAUX` is the permanent tablespace for the PYQSYS schema.
- `TEMP` is the temporary tablespace for the PYQSYS schema.

Open the `install_root.txt` file to see if any errors occurred.

3. At your operating system prompt, start SQL*Plus, connect to your PDB and run the `pyqcfg.sql script`. To capture the log, spool the installation steps to an external file `install_pdb.txt`. The following example uses the PDB `ORCLPDB` and gives example values for the script arguments.

```
sqlplus / as sysdba

SQL> spool install_pdb.txt
```

Log into a PDB where you want to install OML4Py server.

```
SQL> alter session set container=ORCLPDB;
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Run the `pyqcfg.sql` script to install OML4Py server in the PDB.

```
SQL> @pyqcfg.sql SYSAUX TEMP

SQL> spool off;
```

where

- `SYSAUX` is the permanent tablespace for the PYQSYS schema.
- `TEMP` is the temporary tablespace for the PYQSYS schema.

Open the `install_pdb.txt` file to see if any errors occurred

Verify the Server Installation

You can verify the database configuration of OML4Py as oracle user by doing the following:

1. On the OML4Py server database instance, start SQL*Plus as the OML user logging into the PDB, in this example, PDB1.

   ```
   sqlplus oml_user/oml_user_password$ORCLPDB
   ```

2. Run the following command:

   ```
   SELECT * FROM sys.pyq_config;
   ```

   The expected output is as follows:

   ```
   $ sqlplus / as sysdba;
   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 16:23:35 2024


   Copyright (c) 1982, 2024, Oracle.  All rights reserved.


   Connected to:
   Oracle AI
                                      Database 26ai Enterprise Edition Release
   23.0.0.0.0 - Production
   Version 23.4.0.24.05

   SQL> alter session set container=ORCLPDB;

   Session altered.

   SQL> select * from sys.pyq_config;

   NAME
   ------------------------------------------------------------------------------
   -----
   VALUE
   ------------------------------------------------------------------------------
   -----
   PYTHONHOME
   /u01/app/oracle/product/23.4.0.0/dbhome_1/python
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
PYTHONPATH
/u01/app/oracle/product/23.4.0.0/dbhome_1/oml4y/modules

VERSION
2.0


NAME
--------------------------------------------------------------------------------
-----
VALUE
--------------------------------------------------------------------------------
-----
PLATFORM
ODB

DSWLIST
oml.*;pandas.*;numpy.*;matplotlib.*;sklearn.*
```

3. To verify the installation of the OML4Py server for an on-premises database see Verify OML4Py Installation for On-Premises Database.

- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

4.2.2.3.1.1 Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases

The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

Example 4-8 Install OML4Py Server for On-Premises Oracle Databases

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH

cd $ORACLE_HOME/oml4py/server

$ sqlplus / as sysdba
SQL> spool install_root.txt
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
SQL> alter session set container=ORCLPDB;
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;
```

4.2.2.3.2 Verify OML4Py Installation for On-Premises Database

Verify the installation of the OML4Py server and client components for an on-premises database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

   ```
   import oml
   oml.connect(user='oml_user', password='oml_user_password', host='myhost',
               port=1521, service_name='myservice')
   ```

2. Create a user-defined Python function and store it in the OML4Py script repository.

   ```
   oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
   ```

3. Call the user-defined function, using the `oml.do_eval` function.

   ```
   res = oml.do_eval(func='TEST')
   res


   2
   ```

4. When you are finished testing, you can drop the test.

   ```
   oml.script.drop("TEST")
   ```

4.2.2.3.3 Grant Users the Required Privileges for On-Premises Database

Instructions for granting the privileges required for using OML4Py with an on-premises database.

To use OML4Py (OML4Py), a user must have certain database privileges. To store and manage user-defined Python functions in the OML4Py script repository, a user must also have the PYQADMIN database role.

User Privileges

After installing the OML4Py server on an on-premises Oracle AI Database server, grant the following privileges to any OML4Py user.

- CREATE SESSION
- CREATE TABLE
- CREATE VIEW
- CREATE PROCEDURE
- CREATE MINING MODEL

Copyright © 2024, 2026, Oracle and/or its affiliates.

- EXECUTE ON CTXSYS.CTX_DDL (required for using Oracle Text Processing capability in the algorithm classes in the `oml.algo` package)

To grant all of these privileges, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is the OML4Py user:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE,
CREATE MINING MODEL, EXECUTE ON CTXSYS.CTX_DDL to oml_user;
```

Script Repository and Datastore Management

The OML4Py script repository stores user-defined Python functions that a user can invoke in an Embedded Python Execution function. An OML4Py datastore stores Python objects that can be used in subsequent Python sessions. A user-defined Python function in the script repository or a datastore can be available to any user or can be restricted for use by the owner only or by those granted access to it.

The OML4Py server installation script creates the PYQADMIN role in the database. A user must have that role to do the following:

- Store user-defined Python functions in the script repository.
- Drop user-defined Python function from the repository
- Grant or revoke permission to use a user-defined Python function in the script repository.
- Grant or revoke permission to use the objects in a datastore.

To grant this role to a user, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is your OML4Py user:

```
GRANT PYQADMIN to oml_user;
```

4.2.2.3.4 Create New Users for On-Premises Oracle AI Database

The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.

About the pyquser.sql Script

The `pyquser.sql` script is a component of the on-premises OML4Py server installation. The script is in the `server` directory of the installation. The sysdba privilege is required to run the script.

The `pyquser.sql` script grants the new user the required on-premises Oracle AI Database privileges and, optionally, grants the PYQADMIN database role. The PYQADMIN role is required for creating and managing scripts in the OML4Py script repository for use in Embedded Python Execution.

The `pyquser.sql` script takes the following five positional arguments:

- Username
- User's permanent tablespace
- User's temporary tablespace
- Permanent tablespace quota

Copyright © 2024, 2026, Oracle and/or its affiliates.

- PYQADMIN role

When you run the script, it prompts you for a password for the user.

Create a New User

To use the `pyquser.sql` script, go the `server` subdirectory of the directory that contains the extracted OML4Py server installation files. Run the script as a database administrator.

The following examples use SQL*Plus and the sysdba user to run the `pyquser.sql` script.

Example 4-9 Creating New Users

This example creates the user oml_user with the permanent tablespace USERS with an unlimited quota, the temporary tablespace TEMP, and grants the PYQADMIN role to the oml_user.

```
sqlplus / as sysdba
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin



Enter value for password: <type your password>
```

For a pluggable database:

```
sqlplus / as sysdba
alter session set container=<PDBNAME>
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin
```

The output is similar to the following:

```
SQL> @pyquser.sql oml_user USERS TEMP unlimited pyqadmin
Enter value for password: welcome1
old   1: create user &&1 identified by &password
new   1: create user oml_user identified by welcome1
old   2: default tablespace &&2
new   2: default tablespace USERS
old   3: temporary tablespace &&3
new   3: temporary tablespace TEMP
old   4: quota &&4 on &&2
new   4: quota unlimited on USERS

User created.

old   4:     'create procedure, create mining model to &&1';
new   4:     'create procedure, create mining model to pyquser';
old   6:   IF lower('&&5') = 'pyqadmin' THEN
new   6:   IF lower('pyqadmin') = 'pyqadmin' THEN
old   7:     execute immediate 'grant PYQADMIN to &&1';
new   7:     execute immediate 'grant PYQADMIN to pyquser';

PL/SQL procedure successfully completed.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

This example creates the user oml_user2 with 20 megabyte quota on the USERS tablespace, the temporary tablespace TEMP, and without the PYQADMIN role.

```
sqlplus / as sysdba
@pyquser.sql oml_user2 USERS TEMP 20M FALSE


Enter value for password: <type your password>
```

4.2.2.3.5 Uninstall the OML4Py Server from an On-Premises Database 26ai

Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

Uninstall the On-Premises OML4Py Server for Linux

To uninstall the on-premises OML4Py server for Linux, do the following:

1. Verify that the `PYTHONHOME` environment variable is set to the Python3.12 directory.

   ```
   echo $PYTHONHOME
   ```

2. Verify that `PYTHONPATH` environment variable is set to the directory in which the `oml` modules are installed.

   ```
   echo $PYTHONPATH
   ```

   If it is not set to the proper directory, set it.

   ```
   export PYTHONPATH=$ORACLE_HOME/oml4py/modules
   ```

3. Change directories to the directory containing the server installation zip file.

   ```
   cd $ORACLE_HOME/oml4py
   ```

4. Run the server uninstall script `pyquncfg.sql`, the script is under `$ORACLE_HOME/oml4py/server`.

   ```
   cd $ORACLE_HOME/oml4py/server
   ```

   Start SQL*Plus, connect to the PDB.

   ```
   $ sqlplus / as sysdba;


   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
   Version 23.4.0.24.05

   Copyright (c) 1982, 2024, Oracle. All rights reserved.


   Connected to:
   Oracle AI
                                        Database 26ai Enterprise Edition Release
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
23.0.0.0.0 - Production
Version 23.4.0.24.05
```

Run the `pyquncfg.sql` script to record the log, spool, and the installation steps to an external file.

```
SQL> spool install.txt
SQL> alter session set container=ORCLPDB;
SQL> @pyquncfg.sql
```

4.2.2.4 Install OML4Py Client for Linux for On-Premises Oracle AI Database

Instructions for installing and uninstalling the on-premises OML4Py client.

For instructions on installing the OML4Py client on Oracle Autonomous AI Database, see Install OML4Py Client for Linux for Use With Autonomous AI Database Serverless.

To use the OML4Py client, users will need to install the following:

> **Note**
>
> The supporting packgaes need to be installed prior to the OML4Py client installation.

1. Install Python from source. See Build and Install Python for Linux for On-Premises Databases.
2. Install the Oracle Instant Client. See Install Oracle Instant Client for Linux for On-Premises Databases.
3. Install the OML4Py supporting packages. See Install the Required Supporting Packages for Linux for On-Premises Databases.
4. Install the OML4Py client. See Install OML4Py Client for Linux for On-Premises Databases.

Topics:

- Install Oracle Instant Client and the OML4Py Client for Linux
  Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.
- Verify OML4Py Client Installation for On-Premises Databases
  Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.
- Uninstall the OML4Py Client for On-Premises Databases
  Instructions for uninstalling the OML4Py client.
- Upgrade the OML4Py Client for On-Premises Databases
  Instructions for upgrading the OML4Py client.

4.2.2.4.1 Install Oracle Instant Client and the OML4Py Client for Linux

Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.

Copyright © 2024, 2026, Oracle and/or its affiliates.

To connect the OML4Py client for Linux to an on-premises Oracle AI Database, you must have Oracle Instant Client installed on your local system.

- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

4.2.2.4.1.1 Install Oracle Instant Client for Linux for On-Premises Databases

Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.

The OML4Py client requires Oracle Instant Client to connect to an Oracle AI Database.

To install Oracle Instant Client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.

To install Oracle Instant Client, do the following:

1. Download the Oracle Instant Client for your system. Go to the Oracle Instant Client Downloads page and select Instant Client for Linux x86-64.

Copyright © 2024, 2026, Oracle and/or its affiliates.

2. Locate the section for your version of Oracle AI Database. These instructions use Version 23.26.0.0.0.
3. In the Base section, in the Download column, click the zip file for the Basic Package and save the file in an accessible directory on your system or use `wget` to download the file:

   ```
   wget https://download.oracle.com/otn_software/linux/instantclient/2326000/
   instantclient-basic-linux.x64-23.26.0.0.0.zip
   ```

4. Unzip the package:

   ```
   unzip instantclient-basic-linux.x64-23.26.0.0.0.zip
   ```

   Extracting the package creates the subdirectory `instantclient_23_26`, which contains the Oracle Instant Client files.
5. To install the libaio package with sudo or as the root user, run the following command:

   ```
   sudo yum install libaio
   ```

   > **Note**
   >
   > In some Linux distributions, this package is called `libaio1`.

6. Add the directory that contains the Oracle Instant Client files to the beginning of your `LD_LIBRARY_PATH` environment variable:

   ```
   export LD_LIBRARY_PATH=/path/to/dir/instantclient_23_26:$LD_LIBRARY_PATH
   ```

4.2.2.4.1.2 Install OML4Py Client for Linux for On-Premises Databases

Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.

Prerequisites

To download and install the on-premises OML4Py client, the following are required:

- Write permission on the directory in which you are installing the client.
- Perl 5.8 or higher installed on your system.
- Python 3.13.5. To know more about downloading and installing Python 3.13.5, see Build and Install Python for Linux for On-Premises Databases
- The OML4Py supporting packages. To know more about OML4Py the supporting packages, see Install the Required Supporting Packages for Linux for On-Premises Databases.

To use the OML4Py client to connect to an on-premises Oracle AI Database, the following are required:

- Oracle Instant Client must be installed on the client machine.
- The OML4Py server must be installed on the on-premises database server.

Download and Extract the OML4Py Client Installation File

Copyright © 2024, 2026, Oracle and/or its affiliates.

To download and extract the OML4Py client installation file, do the following:

1. Download the client installation zip file.
   a. Go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network.
   b. Accept the license agreement and select Oracle Machine Learning for Python Downloads (v2.1.1).
   c. Select Oracle Machine Learning for Python Client Install for Oracle AI Database on Linux 64 bit.
   d. Save the zip file to an accessible directory.
2. Go to the directory to which you downloaded the zip file and unzip the file.

   ```
   oml4py-client-linux-x86_64-2.1.1.zip
   ```

   The contents are extracted to a subdirectory named `client`, which contains these five files:

   - `client.pl`
   - `OML4PInstallShared.pm`
   - `client/ oml-2.1.1-cp313-cp313-linux_x86_64.whl`
   - `client/slim/oml-2.1.1-cp313-cp313-linux_x86_64.whl`
   - `client/oml4py.ver`
3. Run the following command to install the OML4Py client package:

   ```
   pip3 install client/oml-2.1.1-cp313-cp313-linux_x86_64.whl
   ```

View the Optional Arguments to the Client Installation Perl Script

In the directory above the unzipped client folder, run the client installation Perl script with the `--help` option to display the arguments to the client installation Perl script.

The following command displays the available installation options:

```
perl -Iclient client/client.pl --help

Oracle Machine Learning for Python 2.1.1 Client.

Copyright (c) 2018, 2024 Oracle and/or its affiliates. All rights reserved.
Usage: client.pl [OPTION]...
Install, upgrade, or uninstall OML4P Client.

  -i, --install install or upgrade (default)
  -u, --uninstall uninstall
  -y never prompt
  --ask interactive mode (default)
  --no-embed do not install embedded python functionality
  --no-deps turn off dependencies checking
  --target <dir> install client into <dir>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

By default, the installation script installs the Embedded Python Execution modules. If you don't want to install this module, then you can use the `--no-embed` flag.

Also by default, the installation script checks for the existence and version of each of the supporting packages that the OML4Py client requires. If a required package is missing or does not meet the version requirement, the installation script displays an error message and exits. You can skip the dependency checking in the client installation by using the `--no-deps` flag. However, to use the `oml` module, you need to have installed acceptable versions of all of the supporting packages.

For a list of the required dependencies, see Install the Required Supporting Packages for Linux for On-Premises Databases.

Run the OML4Py Client Installation Script

To install the OML4Py client, do the following:

1. In the directory above the unzipped client folder, run the script. The following command runs the Perl script in the current directory:

   ```
   perl -Iclient client/client.pl
   ```

2. To install the OML4Py Client for Linux, first check that you have already installed the required dependencies (`numpy 2.1.0`, `pandas 2.2.3`, `scipy 1.14.1`, `matplotlib 3.10.0`, `oracledb 3.3.0`, `scikit_learn 1.6.1`, and `packaging 24.2`) . If these are installed, run:

   ```
   perl -Iclient client/client.pl -i
   ```

   If you need to install both OML4Py and its dependencies, unzip `oml4py-supporting-linux-x86_64-2.1.1.zip` into a folder named `supporting`, then run:

   ```
   perl -Iclient client/client.pl -i --dep-pkgs supporting
   ```

3. To verify that `oml` modules are successfully installed and are ready to use, start Python and import `oml`. At the Linux prompt, enter `python3.13`.

   ```
   python3.13
   ```

   At the Python prompt, enter `import oml` and check the client version.

   ```
   import oml
   oml.__version__
   ```

   The output is similar to the following:

   ```
   $ python3.13
   Python 3.13.5 (main, Dec 17 2025,19:37:22)
   [GCC 8.5.0 20210514 (Red Hat 8.5.0-28.0.1)] on linuxType "help",
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
"copyright",
"credits" or "license" for more information.


import oml
oml.__version__


'2.1.1'
```

4. Display the location of the installation directory.
   If you didn't use the `--target <dir>` argument, then the installed `oml` modules are stored under `$PYTHONHOME/lib/python3.13/site-packages/`. Again, you must have write permission for the target directory.

   In Python, after importing the `oml` module, you can display the directory in which the client is installed. At the Python prompt, enter:

   ```
   oml.__path__
   ```

Connect to the OML4Py Server

Start Python, import `oml`, and create a connection to your OML4Py server using an appropriate password, hostname, and system identifier. The following example uses oml_user as the user and has example argument values. Replace the username and other argument values with the values for your user and database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host=myhost,
            port=1521, service_name=myservicename)
```

After connecting, you can run any of the examples in this publication. For example, you could run Example 8-8.

> **Note**
>
> To use the Embedded Python Execution examples, you must have installed the OML4Py client with the Embedded Python Execution option enabled.
> To use the Automatic Machine Learning (AutoML) examples, you must specify a running connection pool on the server in the `automl` argument in an `oml.connect` invocation.

4.2.2.4.1.3 Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database

Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.

Example 4-10 Command Summary for Installing Oracle Instant Client

```
cd /opt/oracle
unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
sudo yum install libaio
export LD_LIBRARY_PATH=/opt/oracle/instantclient_23_04:$LD_LIBRARY_PATH
```

Example 4-11 Command Summary for Installing OML4Py Client

```
unzip v1048628-01.zip
perl -Iclient client/client.pl
```

4.2.2.4.2 Verify OML4Py Client Installation for On-Premises Databases

Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

   ```
   import oml
   oml.connect(user='oml_user', password='oml_user_password', host='myhost',
               port=1521, service_name=myservicename)
   ```

2. Create a user-defined Python function and store it in the OML4Py script repository.

   ```
   oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
   ```

3. Call the user-defined function, using the `oml.do_eval` function.

   ```
   res = oml.do_eval(func='TEST')
   res
   ```

   ```
   2
   ```

4. When you are finished testing, you can drop the test.

   ```
   oml.script.drop("TEST")
   ```

4.2.2.4.3 Uninstall the OML4Py Client for On-Premises Databases

Instructions for uninstalling the OML4Py client.

Uninstall the On-Premises OML4Py Client for Linux

To uninstall the on-premises OML4Py client for Linux, from the directory containing the client installation zip file, run the client installation Perl script with the `-u` argument:

```
perl -Iclient client/client.pl -u
```

When the script displays `Proceed?`, enter `y` or `yes`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

If the client is successfully uninstalled, you'll see the following message:

```
Uninstalling oml-2.1:
  Successfully uninstalled oml-2.1
```

4.2.2.4.4 Upgrade the OML4Py Client for On-Premises Databases

Instructions for upgrading the OML4Py client.

If you already have OML4Py 2.0 client installed and would like to upgrade to 2.0.1 client, do the following:

1. Uninstall OML4Py 2.0 client by following the instructions in Uninstall the OML4Py Client for On-Premises Databases.
2. Install OML4Py 2.0.1 client by following the instructions in Install OML4Py Client for Linux for On-Premises Databases.

4.2.2.5 Install OML4Py Client for Windows for On-Premises and Autonomous Oracle AI Database

Install OML4Py client 2.1 components required for using OML4Py Client on Windows.

> **Note**
>
> The version of OML4Py must be the same on the server and on each client computer. Also, the version of Python must be the same on the server and on each client computer. See Database and System Requirements for OML4Py for supported configurations.

Feature Limitation

While OML4Py 2.1 Windows Client supports Windows OS, many Python API's and in-database algorithms, there are some limitations to consider:

Type	Description
Supported Windows OS	• Windows Server 2022 x64 - Standard • Windows 11 x64 - Pro
Features not available on Windows platform	ONNX model conversion utility `oml.onnx` and `oml.xgb` in-database algorithms.
Feature limit on Windows platform	`oml.automl` function runs sequentially

Topics

- Install Python for Windows
  Steps to install Python on Windows.
- Install Oracle Instant Client
  Install Oracle Instant Client for Windows client.
- Install the Required Supporting Packages
  Install the required supporting packages for installing OML4Py on Windows.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Install OML4Py Client Package
  Instructions for installing OML4Py Client for Windows.
- Configure Oracle Wallet for Secure Connection to Autonomous AI Database
  Oracle Wallet is a secure software container used to store authentication credentials, such as SSL certificates and private keys, needed to connect securely to Oracle Autonomous AI Database. Proper wallet configuration enables applications, such as those using Oracle Machine Learning for Python (OML4Py), to connect to Autonomous AI Database instances with strong data encryption and authentication.
- Verify the Installation of Windows Client
  Once OML4Py is installed, you can connect your Python session to the OML4Py server. This connection enables your Python environment to access data stored in your Oracle AI Database or Autonomous AI Database schema, while also leveraging the database server's processing power, memory, and storage through the OML4Py interface.

4.2.2.5.1 Install Python for Windows

Steps to install Python on Windows.

> **Note**
>
> Python 3.13.2 or higher is required to use OML4Py Client 2.1 on Windows.

Steps for downloading and installating Python 3.13.2:

1. Go to the Python website Python website and scroll down to the Files section. Download the Windows system 64-bit, Windows installer (64-bit). The downloaded file name should be "python-3.13.2-amd64".
2. Run the downloaded installer by double-clicking the file. The following the Python setup screen will appear:
3. After running the Python installer:
   - If you check "Add python.exe to PATH":

     The installer adds the installation directory and its `Scripts\` location to your system's PATH. This lets you to type `python` to start the interpreter, and `pip` to install packages directly from the command line. You can then click **Install Now** and follow the on-screen instructions. Once installation finishes, Python is ready to use.
   - If you leave "Add python.exe to PATH" unchecked:

     The installer will not update your PATH. This means you'll need to manually add the installation directory (and the `Scripts\` folder) to the PATH environment variable before you can run `python` or `pip` from the command line. To manually modify the PATH using the directions in See: Setting environment variables.

     For example, if you installed Python3.13.2 to the default directory `C:\Users\<YourUsername>\AppData\Local\Programs\Python\Python313\`, you can run the following commands to set up environment variable:

     ```
     set PYTHONHOME =
     C:\Users\<YourUsername>\AppData\Local\Programs\Python\Python313\
     set PATH=%PATH%;%PYTHONHOME%;%PYTHONHOME%\Scripts\
     ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

This PATH setting will only remain for the current session. To make it permanent, you must add the instant client path to the Windows environment variables.

To set the environment variable permanently on Windows 11, follow these steps:

a. To configure the Python environment variables permanently: Follow the navigation path. Choose from :
   - Navigation path:Start → Settings → System → About → (scroll down) → Advanced system settings → Advanced tab → Environment Variables
   - Alternate shortcut:Press Win + r type sysdm.cpl, and press Enter to open System Properties → Advanced → Environment Variables directly

b. Under the **User variables**, click **New**. Add the Variable name: `PYTHONHOME` and Variable value: `C:\Users\<YourUsername>\AppData\Local\Programs\Python\Python313`.

c. If you did not add `PYTHONHOME` to `PATH` during the Python installation, in the same **User variables** section, select **Path**, then click **Edit** → New, and add: `%PYTHONHOME%%PYTHONHOME%\Scripts`. Click OK to close all dialogs.

d. Reopen your terminal for the changes to take effect.

Verify the Python Installation
You can now start Python by running the command `python` in a command shell.

```
python


Python 3.13.2 (tags/v3.13.2:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
```

To verify the directory where Python is installed, use the `sys.executable` command from the `sys` package. For example:

```
import sys
print(sys.executable)


C:\Users\<YourUsername>\AppData\Local\Programs\Python\Python313\python.exe
```

This returns the absolute path of the Python executable binary.

4.2.2.5.2 Install Oracle Instant Client

Install Oracle Instant Client for Windows client.

The OML4Py client requires Oracle Instant Client to connect to an Oracle database.

To install Oracle Instant Client, do the following:

1. Download the Oracle Instant Client for your system. Go to the Oracle Instant Client Downloads page.
2. Download an Oracle 23.8 (Basic Light) or Oracle 21.19 (Basic Light) zip file, matching your application architecture.

Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Unzip the package into a single directory that is accessible to your application, for example `C:\oracle\instantclient_23_8`.
4. Set the environment variable `PATH`.

   You must add the path to `instantclient_23_8` to the beginning `PATH` environment variable.

   ```
   set PATH=C:\oracle\instantclient_23_8;%PATH%
   ```

   This PATH setting will only remain for the current session. To make it permanent, you must add the instant client path to the Windows environment variables.

   To set the environment variable permanently on Windows 11, follow these steps:

   a. To configure the Python environment variables permanently: Follow the navigation path. Choose from :
      - Navigation path:Start → Settings → System → About → (scroll down) → Advanced system settings → Advanced tab → Environment Variables
      - Alternate shortcut:Press Win + r type sysdm.cpl, and press Enter to open System Properties → Advanced → Environment Variables directly

   b. Under the **User variables**, click **New**. Add the Variable name: `PATH` and Variable value: `C:\oracle\instantclient_23_8;%PATH%`.

   c. Click OK to close all dialogs.

   d. Reopen your terminal for the changes to take effect.

4.2.2.5.3 Install the Required Supporting Packages

Install the required supporting packages for installing OML4Py on Windows.

Both the OML4Py server and client installations for Windows client require that you also install a set of supporting Python packages and utilities, as described below.

Get unzip on the Windows

During the installation of OML4Py, unzip utilities are necessary. unzip commands are required to unpack files during setup while running automated scripts.

To verify whether `unzip` is installed, open Command Prompt (cmd.exe) and type `unzip`. If you receive a message like 'unzip' is not recognized as an internal or external command, operable program or batch file", then follow the instructions to download unzip:

1. Click here to download the unzip setup file.

   > **Note**
   >
   > No installation is required.

2. Navigate to the folder where the .exe files is stored. Create the folder named `Unzip` and move the .exe files to the new folder. The path to the .exe files should be similar to:

   ```
   <path_to_dir>\Unzip\unzip.exe
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Restart your cmd and type:

   ```
   echo %PATH%
   ```

   At the end of the PATH, you will see `D:\Unzip`.

4. Type "Unzip" to see a list of available commands.

Install Required Packages on OML4Py Windows Client

These steps outline how to install the required Python packages for installing OML4Py on Windows client:

1. Create a file named requirements.txt containing the following content:

   ```
   numpy>=2.1.0
   pandas>=2.2.3
   scipy>=1.14.1
   matplotlib>=3.10.0
   oracledb>=2.4.1
   pyreadline3>=3.5.4
   scikit-learn>1.6.1
   ```

2. Choose from:

   - Install the packages with requirements.txt.

     ```
     pip3 install -r requirements.txt
     ```

   - Install the packages from the supporting package bundle. Download oml4py-supporting-win-x86_64-2.1.zip and unzip into supporting folder. Install the packages with `--no-index --find-links`.

     ```
     python3 -m pip install --no-index --find-links=supporting/ -r
     requirements.txt
     ```

     > **Note**
     >
     > `--target <install folder>` can be used to specify the package installation folder.

4.2.2.5.4 Install OML4Py Client Package

Instructions for installing OML4Py Client for Windows.

Steps to install the OML4Py client package on Windows

Download and Extract the OML4Py Client Installation File

To download and extract the OML4Py client installation file, do the following:

1. Download the client installation zip file.

   a. Go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network.

Copyright © 2024, 2026, Oracle and/or its affiliates.

b. Accept the license agreement.

c. Select Oracle Machine Learning for Python Client Install on Windows.

d. Save the zip file to an accessible directory.

2. Navigate to the directory to which you downloaded the zip file and unzip the file.

```
unzip oml4py_windows_client.zip
```

The contents are extracted to a subdirectory named `client`, which contains these four files:

- client/client.bat
- client/libmmd.dll
- client/oml-2.1-cp313-cp313-win_amd64.whl
- client/sitecustomize.py

Install OML4Py client

OMl4Py client can be installed either by using the bat script or by without script.

1. Install OML4Py client with bat script

a. Set all files under client folder to READ only.

```
icacls "client\*" /inheritance:r
icacls "client\*" /remove:g Administrators SYSTEM %USERNAME%
icacls "client\*" /grant:r %USERNAME%:RX Administrators:RX SYSTEM:RX
```

b. Install OML4Py:

To install OML4Py run the following command:

```
cd client
client.bat -i
```

c. Uninstall OML4Py:

To uninstall OML4P run the following command:

```
client.bat -u
```

Client installation script guidance

`-h, --help` : `-h` or `--help` will show the usage of install bat script.

```
client -h
```

`-i, --install`: `-i` or `--install` will install the OML4Py client. `--proxy` will specify HTTP proxy for pip install.

```
client.bat -i --proxy=www.example.com:<port>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

`--dep-pkgs`: It specifies the folder containing prebuilt dependent packages. Pip will install the missing dependent package from `--dep-pkgs` folder first.

```
client -i --dep-pkgs
C:\Users\<YourUsername>\Desktop\oml4py\client\supporting
```

`--target`: `--target` will install client into specified target directory.

```
client -i --target
C:\Users\<YourUsername>\Desktop\oml4py\client\install_test
```

`-y`: `-y` will make the script never prompt, will always agree the script to proceed.

```
client -i -y --target
C:\Users\<YourUsername>\Desktop\oml4py\client\install_test
```

`--no-deps`: `--no-deps` will skip the dependent package checking and installation.

```
client -i --no-deps
```

`-u` or `--uninstall`: `-u` or `--uninstall` will uninstall the OML4Py client.

```
client -u
```

For example:

- Install with `--proxy` option
  - client -i --proxy http://www.-proxy.example.com:80
  - client -i -y --proxy http://www.-proxy.example.com:80
  - client -i -–no-deps --proxy http://www.-proxy.example.com:80
- Install with `--dep-pkgs` option
  - client -i --dep-pkgs C:\Users\<YourUsername>\Desktop\oml4py\client\supporting
  - client -i –y --dep-pkgs C:\Users\<YourUsername>\Desktop\oml4py\client\supporting
  - client -i -–no-deps --dep-pkgs C:\Users\<YourUsername>\Desktop\oml4py\client\supporting
- Install with `--target` option
  - client -i --target C:\Users\<YourUsername>\Desktop\oml4py\client\install_test --proxy http://www.-proxy.example.com:80
  - client -i --target C:\Users\<YourUsername>\Desktop\oml4py\client\install_test --dep-pkgs C:\Users\<YourUsername>\Desktop\oml4py\client\supporting
- Uninstall with `-y` option
  - client -u

Copyright © 2024, 2026, Oracle and/or its affiliates.

- client -u -y

2. To Install OML4Py client without script.

   You can directly install packages from a `.whl` file using pip3 if you don't want to verify dependencies, Python requirements, or set access control on the installation folder.

   a. Install oml whl from client folder:

   ```
   pip3 install --no-deps client/oml-2.1-cp313-cp313-win_amd64.whl
   ```

   b. Copy libmmd.dll to `%PYTHONHOME%`:

   ```
   cp client/libmmd.dll %PYTHONHOME%
   ```

   c. Copy sitecustomize.py to `%PYTHONHOME%/Lib/site-packages/`:

   ```
   cp client/sitecustomize.py %PYTHONHOME%/Lib/site-packages/
   ```

4.2.2.5.5 Configure Oracle Wallet for Secure Connection to Autonomous AI Database

Oracle Wallet is a secure software container used to store authentication credentials, such as SSL certificates and private keys, needed to connect securely to Oracle Autonomous AI Database. Proper wallet configuration enables applications, such as those using Oracle Machine Learning for Python (OML4Py), to connect to Autonomous AI Database instances with strong data encryption and authentication.

The following steps outline how to configure Oracle Wallets to securely connect to an Autonomous AI Database:

1. Download the Oracle Wallet
   a. Log in to the Oracle Cloud Console.
   b. Navigate to **Autonomous AI Database** and select your target database instance.
   c. Click **DB Connection**.
   d. Click **Download Wallet**.
   e. Set a wallet password for secure storage and click **Download**.
   f. Save the ZIP file (Wallet_<db_name>.zip) to a secure location on your client machine.
2. Extract and Secure the Wallet Files
   Unzip the downloaded wallet file into a secure directory (e.g., `mywalletdir`)

   ```
   mkdir -p mywalletdir
   unzip Wallet_<db_name>.zip -d mywalletdir
   cd mywalletdir/
   ls
   ```

   ```
   README ewallet.p12 ojdbc.properties tnsnames.ora cwallet.sso
   keystore.jks sqlnet.ora truststore.jks
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Update the `sqlnet.ora` file to specify the wallet location. If your environment uses a proxy firewall, enable the `SQLNET.USE_HTTPS_PROXY` parameter by setting its value to `on`:

```
WALLET_LOCATION = (SOURCE = (METHOD = file) (METHOD_DATA =
(DIRECTORY="mywalletdir")))
SSL_SERVER_DN_MATCH=yes
SQLNET.USE_HTTPS_PROXY=on
```

4. If you are working behind a firewall, add the proxy address and port number to each service level in your `tnsnames.ora` file. Additionally, include connection pool entries for all service levels. You will also need to create three new entries specifically for the AutoML connection pools, as illustrated below.

> **Note**
>
> If the proxy server contains a firewall to terminate connections within a set time period, the database connection will also be terminated.

For example, `myadb_medium_pool` is another alias for the connection string with `SERVER=POOLED` added to the corresponding one for `myadb_medium`.

- **For `myadb_medium`**:

  - ```
    myadb_low = (description= (retry_count=20)(retry_delay=3)
    (address=(https_proxy=your proxy
          address here)(https_proxy_port=80)(protocol=tcps)(port=1522)
    (host=adb.us-sanjose-1.oraclecloud.com))
    (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclec
    loud.com))(security=(ssl_server_cert_dn="CN=adb.us-
    sanjose-1.oraclecloud.com,OU=Oracle
          ADB SANJOSE,O=Oracle Corporation,L=Redwood
    City,ST=California,C=US")))
    ```

  - ```
    myadb_medium = (description= (retry_count=20)(retry_delay=3)
    (address=(https_proxy=your proxy
          address here)(https_proxy_port=80)(protocol=tcps)(port=1522)
    (host=adb.us-sanjose-1.oraclecloud.com))
    (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclec
    loud.com))(security=(ssl_server_cert_dn="CN=adb.us-
    sanjose-1.oraclecloud.com,OU=Oracle
          ADB SANJOSE,O=Oracle Corporation,L=Redwood
    City,ST=California,C=US")))
    ```

  - ```
    myadb_high = (description= (retry_count=20)(retry_delay=3)
    (address=(https_proxy=your proxy
          address here)(https_proxy_port=80)(protocol=tcps)(port=1522)
    (host=adb.us-sanjose-1.oraclecloud.com))
    (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclec
    loud.com))(security=(ssl_server_cert_dn="CN=adb.us-
    sanjose-1.oraclecloud.com,OU=Oracle
          ADB SANJOSE,O=Oracle Corporation,L=Redwood
    City,ST=California,C=US")))
    ```

- **For `myadb_medium_pool`**:

Copyright © 2024, 2026, Oracle and/or its affiliates.
    ```

- ```
  myadb_low_pool = (description= (retry_count=20)(retry_delay=3)
  (address=(https_proxy=your
        proxy address here)(https_proxy_port=80)(protocol=tcps)
  (port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
  (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclec
  loud.com)(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
  sanjose-1.oraclecloud.com,OU=Oracle
        ADB SANJOSE,O=Oracle Corporation,L=Redwood
  City,ST=California,C=US")))
  ```

- ```
  myadb_medium_pool = (description= (retry_count=20)(retry_delay=3)
  (address=(https_proxy=your
        proxy address here)(https_proxy_port=80)(protocol=tcps)
  (port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
  (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclec
  loud.com)(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
  sanjose-1.oraclecloud.com,OU=Oracle
        ADB SANJOSE,O=Oracle Corporation,L=Redwood
  City,ST=California,C=US")))
  ```

- ```
  myadb_high_pool = (description= (retry_count=20)(retry_delay=3)
  (address=(https_proxy=your
        proxy address here)(https_proxy_port=80)(protocol=tcps)
  (port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
  (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclec
  loud.com)(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
  sanjose-1.oraclecloud.com,OU=Oracle
        ADB SANJOSE,O=Oracle Corporation,L=Redwood
  City,ST=California,C=US")))
  ```

5. Set the `TNS_ADMIN` Environment Variable in Windows GUI
   To enable client applications to locate and use the wallet, set the `TNS_ADMIN` environment variable:

   TNS_ADMIN: Path to the directory containing the wallet and network configuration files (`tnsnames.ora`, `sqlnet.ora`, etc.)

   a. Open the Start Menu

      Click the Windows Start button in the lower left corner of your screen.

   b. Access System Properties

      - Type

        ```
        environment variables
        ```

        in the search box.
      - Select **Edit the system environment variables** from the search results.

   c. Open Environment Variables Window

      In the System Properties window, click the **Environment Variables...** button (usually near the bottom).

   d. Add or Edit `TNS_ADMIN` Variable

Copyright © 2024, 2026, Oracle and/or its affiliates.
        ```

- Under the **User variables** section (to set it for your user only) or **System variables** (to set it system-wide), click **New…**.
- For **Variable name**, enter: `TNS_ADMIN`
- For **Variable value**, enter the full path to your Oracle Wallet directory (for example, `C:\Users\yourname\wallet`).

*If `TNS_ADMIN` already exists, select it and click **Edit...** instead, then update the value.*

e. Apply and Close

- Click **OK** in the New/Edit dialog.
- Click **OK** to close the Environment Variables window.
- Click **OK** again to close the System Properties window.

f. Restart Applications

Any open command prompts or applications will need to be restarted to recognize the new or updated environment variable.

Verify Oracle Autonomous AI Database Connection

Verify that you can securely connect to your Autonomous AI Database (ADB):

1. Start Python and load the `oml` library:

```
python3
import oml
```

2. Establish a database connection using the OML client, which utilizes the wallet for secure access. Set both the `dsn` and `automl` parameters to the appropriate `tnsnames.ora` alias from your wallet directory:

```
oml.connect(user="oml_user", password="oml_user_password",
dsn="myadb_medium", automl="myadb_medium_pool")
```

To avoid exposing your Oracle Machine Learning user credentials in plain text, provide empty strings for the user and password parameters when connecting:

```
oml.connect(user="", password="", dsn="myadb_medium",
automl="myadb_medium_pool")
```

4.2.2.5.6 Verify the Installation of Windows Client

Once OML4Py is installed, you can connect your Python session to the OML4Py server. This connection enables your Python environment to access data stored in your Oracle AI Database or Autonomous AI Database schema, while also leveraging the database server's processing power, memory, and storage through the OML4Py interface.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, service_name=myservicename)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

2. Create a user-defined Python function and store it in the OML4Py script repository.

```
oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
```

3. Call the user-defined function, using the `oml.do_eval` function.

```
res = oml.do_eval(func='TEST')
res


2
```

4. When you are finished testing, you can drop the test.

```
oml.script.drop("TEST")
```

4.2.3 Install OML4Py 2.0.1 for On-Premises Databases

The following topics tell how to install and uninstall the server and client components required for using OML4Py 2.0.1 with an on-premises Oracle Database.

Topics:

- Build and Install Python for Linux for On-Premises Databases
  Instructions for installing Python for Linux for an on-premises oracle database.
- Install the Required Supporting Packages for Linux for On-Premises Databases
  Both the OML4Py server and client installations for an on-premises Oracle database require that you also install a set of supporting Python packages, as described below.
- Install OML4Py Server for On-Premises Oracle Database
  These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.
- Install OML4Py Client for On-Premises Oracle Database
  Instructions for installing and uninstalling the on-premises OML4Py client.

4.2.3.1 Build and Install Python for Linux for On-Premises Databases

Instructions for installing Python for Linux for an on-premises oracle database.

For 23ai, building Python is only necessary for the client. The database has Python 3.12.1 in `$ORACLE_HOME/python`.

> **Note**
>
> The feature supporting text transformers from Hugging Face by converting them into ONNX format models will only work on the OML4Py client. It is not supported on the OML4Py server.

Python 3.12.3 is required to install and use OML4Py 2.1 client.

These steps describe building and installing Python 3.12.3 for Linux.

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. Go to the Python website and download the **Gzipped source tarball**. The downloaded file name is `Python-3.12.3.tgz`

   ```
   wget https://www.python.org/ftp/python/3.12.3/Python-3.12.3.tgz
   ```

2. Create a directory, such as `$HOME/python`, and extract the contents to this directory:

   ```
   mkdir -p $HOME/python
   tar -xvzf Python-3.12.3.tgz --strip-components=1 -C $HOME/python
   ```

   The contents of the Gzipped source tarball will be copied directly to `$HOME/python`.

3. OML4Py requires the presence of the `perl-Env`, `libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel`, `libuuid-devel` and `ncurses-devel` libraries.
   Install these packages as sudo or root user:

   ```
   sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-
   devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel
   ```

4. To build Python 3.12.3, navigate to the `$HOME/python` directory where Python-3.12.3 will be installed. Enter the following commands on the Oracle Machine Learning for Python server:

   ```
   cd $HOME/python
   ./configure --enable-shared --prefix=$HOME/python

   make clean; make
   make altinstall
   ```

   > **Note**
   >
   > Be sure to use the `--enable-shared` flag if you are going to use Embedded Python Execution; otherwise, using an Embedded Python Execution function results in an `extproc` error.

   > **Note**
   >
   > Be sure to invoke `make altinstall` instead of `make install` to avoid overwriting the system Python.

5. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

   ```
   export PYTHONHOME=$HOME/python
   export PATH=$PYTHONHOME/bin:$PATH
   export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> In order to use Python for OML4Py, the variables must be set, and these variables must appear before system Python in PATH and LD_LIBRARY_PATH.

6. Create a symbolic link in your `$HOME/python/bin` directory to link to your python3.12 executable, which you can do with the following commands:

```
cd $HOME/python/bin
ln -s python3.12 python3
```

pip will return warnings during package installation if the latest version is not installed. You can upgrade the version of pip to avoid these warnings:

```
python3 -m pip install --upgrade pip
```

You can now start Python by running the command `python3`. To verify the directory where Python is installed, use the `sys.executable` command from the `sys` package. For example:

```
python3


Python 3.12.3 (main, Aug 14 2024, 15:13:51) [GCC 8.5.0 20210514 (Red Hat
8.5.0-18.0.6)] on linux
Type "help", "copyright", "credits" or "license" for more information.


import sys
print(sys.executable)


/home/<user>/python/bin/python3
```

This returns the absolute path of the Python executable binary.

If you run the command `python3` and you get the error `command not found`, then that means the system cannot find an executable named `python3` in `$PYTHONHOME/bin`. A symlink is required for the OML4Py server installation components. So, in that case, you need to create a symbolic link in your `$HOME/python/bin` directory to link to your `python3.12` executable as described in Step 7.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Topics:

4.2.3.2 Install the Required Supporting Packages for Linux for On-Premises Databases

Both the OML4Py server and client installations for an on-premises Oracle database require that you also install a set of supporting Python packages, as described below.

Installing required packages on OML4Py client machine

> **Note**
>
> `onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` are to be installed on the client only. `onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` support the ONNX conversion feature on the OML4Py client and should installed on the client only. All other packages are installed on both the client and server.

> **Note**
>
> `oracledb 2.2.0` or later version is required to invoke vector database SQL APIs in 23ai.

These steps outline how to install the required Python packages for an on-premises OML4Py client:

1. Create a file named `requirements.txt` containing the following content:

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.1.1
setuptools==68.0.0
scipy==1.12.0
matplotlib==3.8.4
oracledb==2.2.0
scikit-learn==1.4.1.post1
numpy==1.26.4
onnxruntime==1.17.0
onnxruntime-extensions==0.10.1
onnx==1.16.0
torch==2.2.0
transformers==4.38.1
sentencepiece==0.2.0
```

2. Install the packages with `requirements.txt`.

```
pip3.12 install -r requirements.txt
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Installing required packages on OML4Py server machine

On the OML4Py server machine, all these packages must be installed into `$ORACLE_HOME/oml4py/modules` so they can be detected by the Embedded Python Execution process.

These steps outline how to install the required Python packages for an on-premises OML4Py server:

1. Create a file named `requirements2.txt` containing the following content.

```
pandas==2.1.1
setuptools==68.0.0
scipy==1.12.0
matplotlib==3.8.4
oracledb==2.2.0
joblib==1.3.2
scikit-learn==1.4.1.post1
numpy==1.26.4
```

2. Install the packages with `requirements2.txt`. Run the following command, specifying the target directory, `$ORACLE_HOME/oml4py/modules`:

```
pip3.12 install -r requirements2.txt --target=$ORACLE_HOME/oml4py/modules
```

Verify the Package Installation

Load the packages below to ensure they have been installed successfully. Start Python and run the following commands:

```
python3
```

```
Python 3.12.6 (main, Aug 14 2024, 15:13:51) [GCC 8.5.0 20210514 (Red
Hat 8.5.0-18.0.6)] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

```
import numpy
import pandas
import scipy
import matplotlib
import oracledb
import sklearn
import onnx
import torch
import onnxruntime_extensions
import transformers
import sentencepiece
```

If all the packages are installed successfully, no errors will be returned.

4.2.3.3 Install OML4Py Server for On-Premises Oracle Database

These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.
- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.
- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.3.3.1 Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai

Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.

You can install OML4Py by using a Python script included in your 26ai database or by using the Database Configuration Assistant (DBCA).

Install OML4Py Using a SQL Script

To install the on-premises OML4Py server, the following are required:

- A connection to the internet.
- OML4Py supporting packages. For instructions on installing the required supporting packages see Install the Required Supporting Packages for Linux for On-Premises Databases.
- Perl 5.8 or higher installed on your system.

> **Note**
>
> Perl requires the `perl-Env` package. You can install the package as root with the command `yum install perl-Env` .

To check for the existence of `perl-Env`, run the following command. The version will vary depending on your Operating System and version:

```
rpm -qa perl-Env
perl-Env-1.04-395.el8.noarch
```

- Write permission on the directories to which you download and install the server components.

> **Note**
>
> The following environment variables must be set up.

- Set environment variables: Set `PYTHONHOME` and add it to your `PATH`
- Set `ORACLE_HOME` and add it to your `PATH`
- Set `LD_LIBRARY_PATH`

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

To install the OML4Py server for Linux for an on-premises Oracle AI Database 26ai, run the server installation SQL script `pyqcfg.sql`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> The OML4Py server needs to be installed on `CDB$ROOT` first, followed by installation on a PDB for Oracle AI Database 26ai.

1. The `pyqcfg.sql` script is under `$ORACLE_HOME/oml4py/server`. Change directory to `$ORACLE_HOME/oml4py/server`.

   ```
   cd $ORACLE_HOME/oml4py/server
   ```

2. At your operating system prompt, start SQL*Plus, connect to the `CDB$ROOT` and run the `pyqcfg.sql` script.

   ```
   sqlplus / as sysdba


   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
   Version 23.4.0.24.05

   Copyright (c) 1982, 2024, Oracle.  All rights reserved.


   Connected to:
   Oracle AI
                                      Database 26ai Enterprise Edition Release
   23.0.0.0.0 - Production
   Version 23.4.0.24.05
   ```

   To capture the log, spool the installation steps to an external file `install_root.txt`.

   ```
   SQL> spool install_root.txt
   ```

   Verify that you are currently connected to the `CDB$ROOT` container.

   ```
   SQL> show con_name



   CON_NAME
   ------------------------------
   CDB$ROOT
   ```

   Run the `pyqcfg.sql` script to install OML4Py server in the `CDB$ROOT` first.

   ```
   SQL> @pyqcfg.sql SYSAUX TEMP

   SQL> spool off;
   ```

   where

   - `SYSAUX` is the permanent tablespace for the PYQSYS schema.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- TEMP is the temporary tablespace for the PYQSYS schema.

Open the install_root.txt file to see if any errors occurred.

3. At your operating system prompt, start SQL*Plus, connect to your PDB and run the pyqcfg.sql script. To capture the log, spool the installation steps to an external file install_pdb.txt. The following example uses the PDB ORCLPDB and gives example values for the script arguments.

```
sqlplus / as sysdba

SQL> spool install_pdb.txt
```

Log into a PDB where you want to install OML4Py server.

```
SQL> alter session set container=ORCLPDB;
```

Run the pyqcfg.sql script to install OML4Py server in the PDB.

```
SQL> @pyqcfg.sql SYSAUX TEMP

SQL> spool off;
```

where

- SYSAUX is the permanent tablespace for the PYQSYS schema.
- TEMP is the temporary tablespace for the PYQSYS schema.

Open the install_pdb.txt file to see if any errors occurred

Verify the Server Installation

You can verify the database configuration of OML4Py as oracle user by doing the following:

1. On the OML4Py server database instance, start SQL*Plus as the OML user logging into the PDB, in this example, PDB1.

```
sqlplus oml_user/oml_user_password$ORCLPDB
```

2. Run the following command:

```
SELECT * FROM sys.pyq_config;
```

The expected output is as follows:

```
$ sqlplus / as sysdba;
SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 16:23:35 2024


Copyright (c) 1982, 2024, Oracle.  All rights reserved.


Connected to:
Oracle AI
                                    Database 26ai Enterprise Edition Release
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
23.0.0.0.0 - Production
Version 23.4.0.24.05

SQL> alter session set container=ORCLPDB;

Session altered.

SQL> select * from sys.pyq_config;

NAME
--------------------------------------------------------------------------------
-----
VALUE
--------------------------------------------------------------------------------
-----
PYTHONHOME
/u01/app/oracle/product/23.4.0.0/dbhome_1/python

PYTHONPATH
/u01/app/oracle/product/23.4.0.0/dbhome_1/oml4y/modules

VERSION
2.0


NAME
--------------------------------------------------------------------------------
-----
VALUE
--------------------------------------------------------------------------------
-----
PLATFORM
ODB

DSWLIST
oml.*;pandas.*;numpy.*;matplotlib.*;sklearn.*
```

3. To verify the installation of the OML4Py server for an on-premises database see Verify OML4Py Installation for On-Premises Database.

- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.3.3.1.1 Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases

The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

Example 4-12 Install OML4Py Server for On-Premises Oracle Databases

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH

cd $ORACLE_HOME/oml4py/server

$ sqlplus / as sysdba
SQL> spool install_root.txt
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;

SQL> alter session set container=ORCLPDB;
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;
```

4.2.3.3.2 Verify OML4Py Installation for On-Premises Database

Verify the installation of the OML4Py server and client components for an on-premises database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

   ```
   import oml
   oml.connect(user='oml_user', password='oml_user_password', host='myhost',
               port=1521, service_name='myservice')
   ```

2. Create a user-defined Python function and store it in the OML4Py script repository.

   ```
   oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
   ```

3. Call the user-defined function, using the `oml.do_eval` function.

   ```
   res = oml.do_eval(func='TEST')
   res


   2
   ```

4. When you are finished testing, you can drop the test.

   ```
   oml.script.drop("TEST")
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.3.3.3 Grant Users the Required Privileges for On-Premises Database

Instructions for granting the privileges required for using OML4Py with an on-premises database.

To use OML4Py (OML4Py), a user must have certain database privileges. To store and manage user-defined Python functions in the OML4Py script repository, a user must also have the PYQADMIN database role.

User Privileges

After installing the OML4Py server on an on-premises Oracle AI Database server, grant the following privileges to any OML4Py user.

- CREATE SESSION
- CREATE TABLE
- CREATE VIEW
- CREATE PROCEDURE
- CREATE MINING MODEL
- EXECUTE ON CTXSYS.CTX_DDL (required for using Oracle Text Processing capability in the algorithm classes in the `oml.algo` package)

To grant all of these privileges, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is the OML4Py user:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE,
CREATE MINING MODEL, EXECUTE ON CTXSYS.CTX_DDL to oml_user;
```

Script Repository and Datastore Management

The OML4Py script repository stores user-defined Python functions that a user can invoke in an Embedded Python Execution function. An OML4Py datastore stores Python objects that can be used in subsequent Python sessions. A user-defined Python function in the script repository or a datastore can be available to any user or can be restricted for use by the owner only or by those granted access to it.

The OML4Py server installation script creates the PYQADMIN role in the database. A user must have that role to do the following:

- Store user-defined Python functions in the script repository.
- Drop user-defined Python function from the repository
- Grant or revoke permission to use a user-defined Python function in the script repository.
- Grant or revoke permission to use the objects in a datastore.

To grant this role to a user, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is your OML4Py user:

```
GRANT PYQADMIN to oml_user;
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.3.3.4 Create New Users for On-Premises Oracle AI Database

The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.

About the pyquser.sql Script

The `pyquser.sql` script is a component of the on-premises OML4Py server installation. The script is in the `server` directory of the installation. The sysdba privilege is required to run the script.

The `pyquser.sql` script grants the new user the required on-premises Oracle AI Database privileges and, optionally, grants the PYQADMIN database role. The PYQADMIN role is required for creating and managing scripts in the OML4Py script repository for use in Embedded Python Execution.

The `pyquser.sql` script takes the following five positional arguments:

- Username
- User's permanent tablespace
- User's temporary tablespace
- Permanent tablespace quota
- PYQADMIN role

When you run the script, it prompts you for a password for the user.

Create a New User

To use the `pyquser.sql` script, go the `server` subdirectory of the directory that contains the extracted OML4Py server installation files. Run the script as a database administrator.

The following examples use SQL*Plus and the sysdba user to run the `pyquser.sql` script.

Example 4-13 Creating New Users

This example creates the user oml_user with the permanent tablespace USERS with an unlimited quota, the temporary tablespace TEMP, and grants the PYQADMIN role to the oml_user.

```
sqlplus / as sysdba
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin



Enter value for password: <type your password>
```

For a pluggable database:

```
sqlplus / as sysdba
alter session set container=<PDBNAME>
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

The output is similar to the following:

```
SQL> @pyquser.sql oml_user USERS TEMP unlimited pyqadmin
Enter value for password: welcome1
old   1: create user &&1 identified by &password
new   1: create user oml_user identified by welcome1
old   2: default tablespace &&2
new   2: default tablespace USERS
old   3: temporary tablespace &&3
new   3: temporary tablespace TEMP
old   4: quota &&4 on &&2
new   4: quota unlimited on USERS

User created.

old   4:     'create procedure, create mining model to &&1';
new   4:     'create procedure, create mining model to pyquser';
old   6:   IF lower('&&5') = 'pyqadmin' THEN
new   6:   IF lower('pyqadmin') = 'pyqadmin' THEN
old   7:     execute immediate 'grant PYQADMIN to &&1';
new   7:     execute immediate 'grant PYQADMIN to pyquser';

PL/SQL procedure successfully completed.
```

This example creates the user oml_user2 with 20 megabyte quota on the USERS tablespace, the temporary tablespace TEMP, and without the PYQADMIN role.

```
sqlplus / as sysdba
@pyquser.sql oml_user2 USERS TEMP 20M FALSE


Enter value for password: <type your password>
```

4.2.3.3.5 Uninstall the OML4Py Server from an On-Premises Database 26ai

Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

Uninstall the On-Premises OML4Py Server for Linux

To uninstall the on-premises OML4Py server for Linux, do the following:

1. Verify that the `PYTHONHOME` environment variable is set to the Python3.12 directory.

   ```
   echo $PYTHONHOME
   ```

2. Verify that `PYTHONPATH` environment variable is set to the directory in which the `oml` modules are installed.

   ```
   echo $PYTHONPATH
   ```

   If it is not set to the proper directory, set it.

   ```
   export PYTHONPATH=$ORACLE_HOME/oml4py/modules
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Change directories to the directory containing the server installation zip file.

```
cd $ORACLE_HOME/oml4py
```

4. Run the server uninstall script `pyquncfg.sql`, the script is under `$ORACLE_HOME/oml4py/server`.

```
cd $ORACLE_HOME/oml4py/server
```

Start SQL*Plus, connect to the PDB.

```
$ sqlplus / as sysdba;


SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
Version 23.4.0.24.05

Copyright (c) 1982, 2024, Oracle. All rights reserved.


Connected to:
Oracle AI
                                    Database 26ai Enterprise Edition Release
23.0.0.0.0 - Production
Version 23.4.0.24.05
```

Run the `pyquncfg.sql` script to record the log, spool, and the installation steps to an external file.

```
SQL> spool install.txt
SQL> alter session set container=ORCLPDB;
SQL> @pyquncfg.sql
```

4.2.3.4 Install OML4Py Client for On-Premises Oracle Database

Instructions for installing and uninstalling the on-premises OML4Py client.

For instructions on installing the OML4Py client on Autonomous Database, see Install OML4Py Client for Linux for Use With Autonomous AI Database Serverless.

To use the OML4Py client, users will need to install the following:

> **Note**
>
> The supporting packgaes need to be installed prior to the OML4Py client installation.

1. Install Python from source. See Build and Install Python for Linux for On-Premises Databases.
2. Install the Oracle Instant Client. See Install Oracle Instant Client for Linux for On-Premises Databases.

Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Install the OML4Py supporting packages. See Install the Required Supporting Packages for Linux for On-Premises Databases.
4. Install the OML4Py client. See Install OML4Py Client for Linux for On-Premises Databases.

Topics:

- Install Oracle Instant Client and the OML4Py Client for Linux
  Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.
- Verify OML4Py Client Installation for On-Premises Databases
  Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.
- Uninstall the OML4Py Client for On-Premises Databases
  Instructions for uninstalling the OML4Py client.
- Upgrade the OML4Py Client for On-Premises Databases
  Instructions for upgrading the OML4Py client.

4.2.3.4.1 Install Oracle Instant Client and the OML4Py Client for Linux

Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.

To connect the OML4Py client for Linux to an on-premises Oracle AI Database, you must have Oracle Instant Client installed on your local system.

- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

4.2.3.4.1.1 Install Oracle Instant Client for Linux for On-Premises Databases

Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.

The OML4Py client requires Oracle Instant Client to connect to an Oracle database.

To install Oracle Instant Client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.

To install Oracle Instant Client, do the following:

1. Download the Oracle Instant Client for your system. Go to the Oracle Instant Client Downloads page and select Instant Client for Linux x86-64.
2. Locate the section for your version of Oracle Database. These instructions use Version 23.4.0.0.0.
3. In the Base section, in the Download column, click the zip file for the Basic Package and save the file in an accessible directory on your system or use `wget` to download the file:

   ```
   wget https://download.oracle.com/otn_software/linux/instantclient/2340000/instantclient-basic-linux.x64-23.4.0.24.05.zip
   ```

4. Unzip the package:

   ```
   unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
   ```

   Extracting the package creates the subdirectory `instantclient_23_4`, which contains the Oracle Instant Client files.
5. To install the libaio package with sudo or as the root user, run the following command:

   ```
   sudo yum install libaio
   ```

   > **Note**
   >
   > In some Linux distributions, this package is called `libaio1`.

6. Add the directory that contains the Oracle Instant Client files to the beginning of your `LD_LIBRARY_PATH` environment variable:

   ```
   export LD_LIBRARY_PATH=/opt/oracle/instantclient_23_04:$LD_LIBRARY_PATH
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.3.4.1.2 Install OML4Py Client for Linux for On-Premises Databases

Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

Prerequisites

To download and install the on-premises OML4Py client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.
- Perl 5.8 or higher installed on your system.
- Python 3.12.3. To know more about downloading and installing Python 3.12.3 see, Build and Install Python for Linux for On-Premises Databases.
- The OML4Py supporting packages. To know more about OML4Py the supporting packages, see Install the Required Supporting Packages for Linux for On-Premises Databases.

To use the OML4Py client to connect to an on-premises Oracle database, the following are required:

- Oracle Instant Client must be installed on the client machine.
- The OML4Py server must be installed on the on-premises database server.

Download and Extract the OML4Py Client Installation File

To download and extract the OML4Py client installation file, do the following:

1. Download the client installation zip file.
   a. Go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network.
   b. Accept the license agreement and select Oracle Machine Learning for Python Downloads (v2.0.1).
   c. Select Oracle Machine Learning for Python Client Install for Oracle Database on Linux 64 bit.
   d. Save the zip file to an accessible directory.
2. Go to the directory to which you downloaded the zip file and unzip the file.

   ```
   unzip oml4py-client-linux-x86_64-2.0.1.zip
   ```

   The contents are extracted to a subdirectory named `client`, which contains these four files:

   - `OML4PInstallShared.pm`
   - `oml-2.0.1-cp312-cp312-linux_x86_64.whl`
   - `client.pl`
   - `oml4py.ver`

Copyright © 2024, 2026, Oracle and/or its affiliates.

View the Optional Arguments to the Client Installation Perl Script

In the directory above the unzipped client folder, run the client installation Perl script with the `--help` option to display the arguments to the client installation Perl script.

The following command displays the available installation options:

```
perl -Iclient client/client.pl --help


Oracle Machine Learning for Python 2.0.1 Client.

Copyright (c) 2018, 2024 Oracle and/or its affiliates. All rights reserved.
Usage: client.pl [OPTION]...
Install, upgrade, or uninstall OML4P Client.

  -i, --install install or upgrade (default)
  -u, --uninstall uninstall
  -y never prompt
  --ask interactive mode (default)
  --no-embed do not install embedded python functionality
  --no-deps turn off dependencies checking
  --target <dir> install client into <dir>
```

By default, the installation script installs the Embedded Python Execution modules. If you don't want to install this module, then you can use the `--no-embed` flag.

Also by default, the installation script checks for the existence and version of each of the supporting packages that the OML4Py client requires. If a required package is missing or does not meet the version requirement, the installation script displays an error message and exits. You can skip the dependency checking in the client installation by using the `--no-deps` flag. However, to use the `oml` module, you need to have installed acceptable versions of all of the supporting packages.

For a list of the required dependencies, see Install the Required Supporting Packages for Linux for On-Premises Databases.

Run the OML4Py Client Installation Script

To install the OML4Py client, do the following:

1. In the directory above the unzipped client folder, run the script. The following command runs the Perl script in the current directory:

   ```
   perl -Iclient client/client.pl
   ```

   Alternatively, the following command runs the Perl script with the target directory specified:

   ```
   perl -Iclient client/client.pl --target path_to_target_dir
   ```

   The `--target` flag is optional, if you don't want to install it to the current directory.

   When the script displays `Proceed?`, enter `y` or `yes`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

If you use the `--target <dir>` argument to install the `oml` module to the specified directory, then add that location to environment variable `PYTHONPATH` so that Python can find the module:

```
export PYTHONPATH=path_to_target_dir
```

The command displays the following:

```
perl -Iclient client/client.pl


Oracle Machine Learning for Python 2.0.1 Client.

Copyright (c) 2018, 2024 Oracle and/or its affiliates. All rights reserved.
Checking platform ................. Pass
Checking Python ................... Pass
Checking dependencies ............. Pass
Checking OML4P version ............ Pass
Current configuration
  Python Version .................. 3.12.3
  PYTHONHOME ...................... /opt/Python-3.12.3
  Existing OML4P module version .... None
  Operation ....................... Install/Upgrade

Proceed? [yes]

Processing ./client/oml-2.0.1-cp312-cp312-linux_x86_64.whl
Installing collected packages: oml
Successfully installed oml-2.0.1
```

2. To verify that `oml` modules are successfully installed and are ready to use, start Python and import `oml`. At the Linux prompt, enter `python3`.

```
python3
```

At the Python prompt, enter `import oml` and check the client version.

```
import oml
oml.__version__
```

The output is similar to the following:

```
$ python3
Python 3.12.3 (main, Aug 14 2024, 15:13:51) [GCC 8.5.0 20210514 (Red Hat
8.5.0-18.0.6)] on linux
Type "help", "copyright", "credits" or "license" for more information.


import oml
oml.__version__


'2.0.1'
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Display the location of the installation directory.
   If you didn't use the `--target <dir>` argument, then the installed `oml` modules are stored under `$PYTHONHOME/lib/python3.12/site-packages/`. Again, you must have write permission for the target directory.

   In Python, after importing the `oml` module, you can display the directory in which the client is installed. At the Python prompt, enter:

   ```
   oml.__path__
   ```

Connect to the OML4Py Server

Start Python, import `oml`, and create a connection to your OML4Py server using an appropriate password, hostname, and system identifier. The following example uses oml_user as the user and has example argument values. Replace the username and other argument values with the values for your user and database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host=myhost,
            port=1521, service_name=myservicename)
```

After connecting, you can run any of the examples in this publication. For example, you could run Example 8-8.

> **Note**
>
> To use the Embedded Python Execution examples, you must have installed the OML4Py client with the Embedded Python Execution option enabled.
> To use the Automatic Machine Learning (AutoML) examples, you must specify a running connection pool on the server in the `automl` argument in an `oml.connect` invocation.

4.2.3.4.2 Verify OML4Py Client Installation for On-Premises Databases

Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

   ```
   import oml
   oml.connect(user='oml_user', password='oml_user_password', host='myhost',
               port=1521, service_name=myservicename)
   ```

2. Create a user-defined Python function and store it in the OML4Py script repository.

   ```
   oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Call the user-defined function, using the `oml.do_eval` function.

```
res = oml.do_eval(func='TEST')
res


2
```

4. When you are finished testing, you can drop the test.

```
oml.script.drop("TEST")
```

4.2.3.4.3 Uninstall the OML4Py Client for On-Premises Databases

Instructions for uninstalling the OML4Py client.

Uninstall the On-Premises OML4Py Client for Linux

To uninstall the on-premises OML4Py client for Linux, from the directory containing the client installation zip file, run the client installation Perl script with the `-u` argument:

```
perl -Iclient client/client.pl -u
```

When the script displays `Proceed?`, enter `y` or `yes`.

If the client is successfully uninstalled, you'll see the following message:

```
Uninstalling oml-2.0.1:
  Successfully uninstalled oml-2.0.1
```

4.2.3.4.4 Upgrade the OML4Py Client for On-Premises Databases

Instructions for upgrading the OML4Py client.

If you already have OML4Py 2.0.1 client installed and would like to upgrade to 2.1 client, do the following:

1. Uninstall OML4Py 2.0 client by following the instructions in Uninstall the OML4Py 2.0.1 Client for On-Premises Databases.
2. Install OML4Py 2.0.1 client by following the instructions in Install OML4Py 2.1 Client for Linux for On-Premises Databases.

4.2.4 Install OML4Py 2.0 for On-Premises Databases

The following topics tell how to install and uninstall the server and client components required for using OML4Py 2.0 with an on-premises Oracle Database.

Topics:

- Build and Install Python for Linux for On-Premises Databases
  Instructions for installing Python for Linux for an on-premises oracle database.
- Install the Required Supporting Packages for Linux for On-Premises Databases
  Both the OML4Py server and client installations for an on-premises Oracle database require that you also install a set of supporting Python packages, as described below.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Install OML4Py Server for On-Premises Oracle Database
  These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.
- Install OML4Py Client for On-Premises Oracle Database
  Instructions for installing and uninstalling the on-premises OML4Py client.

4.2.4.1 Build and Install Python for Linux for On-Premises Databases

Instructions for installing Python for Linux for an on-premises oracle database.

For 23ai, building Python is only necessary for the client. The database has Python 3.12.1 in `$ORACLE_HOME/python`.

> **Note**
>
> The feature supporting text transformers from Hugging Face by converting them into ONNX format models will only work on the OML4Py client. It is not supported on the OML4Py server.

Python 3.12.0 or Python 3.12.1 is required to install and use OML4Py.

These steps describe building and installing Python 3.12.0 for Linux.

1. Go to the Python website and download the **Gzipped source tarball**. The downloaded file name is `Python-3.12.0.tgz`

   ```
   wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tgz
   ```

2. Create a directory, such as `/home/user/python`, and extract the contents to this directory:

   ```
   mkdir -p /home/user/python
   tar -xvzf Python-3.12.0.tgz --strip-components=1 -C /home/user/python
   ```

   The contents of the Gzipped source tarball will be copied directly to `/home/user/python`

3. Go to the new directory:

   ```
   cd /home/user/python
   ```

4. OML4Py requires the presence of the `perl-Env`, `libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel`, `libuuid-devel` and `ncurses-devel` libraries.
   Install these packages as sudo or root user:

   ```
   sudo yum install perl-Env libffi-devel openssl openssl-devel tk-devel xz-
   devel zlib-devel bzip2-devel readline-devel libuuid-devel ncurses-devel
   ```

5. To build Python 3.12.0, enter the following commands, where `/home/user/python` is the directory in which you installed Python-3.12.0. The command on the Oracle Machine Learning for Python server will be:

   ```
   cd /home/user/python
   ./configure --enable-shared --prefix=/home/user/python
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
make clean; make
make altinstall
```

> **Note**
>
> Be sure to use the `--enable-shared` flag if you are going to use Embedded Python Execution; otherwise, using an Embedded Python Execution function results in an `extproc` error.

> **Note**
>
> Be sure to invoke `make altinstall` instead of `make install` to avoid overwriting the system Python.

6. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

```
export PYTHONHOME=/home/user/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

> **Note**
>
> In order to use Python for OML4Py, the variables must be set, and these variables must appear before system Python in PATH and LD_LIBRARY_PATH.

7. Create a symbolic link in your `/home/user/python/bin` directory to link to your python3.12 executable, which you can do with the following commands:

```
cd /home/user/python/bin
ln -s python3.12 python3
```

pip will return warnings during package installation if the latest version is not installed. You can upgrade the version of pip to avoid these warnings:

```
python3 -m pip install --upgrade pip
```

You can now start Python by running the command `python3`. To verify the directory where Python is installed, use the `sys.executable` command from the `sys` package. For example:

```
python3


Python 3.12.0 (default, Feb 22 2022, 15:13:36)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> print(sys.executable)
/home/user/python/bin/python3
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

This returns the absolute path of the Python executable binary.

If you run the command `python3` and you get the error `command not found`, then that means the system cannot find an executable named `python3` in `$PYTHONHOME/bin`. A symlink is required for the OML4Py server installation components. So, in that case, you need to create a symbolic link in your `/home/user/python/bin` directory to link to your `python3.12` executable as described in Step 6.

4.2.4.2 Install the Required Supporting Packages for Linux for On-Premises Databases

Both the OML4Py server and client installations for an on-premises Oracle database require that you also install a set of supporting Python packages, as described below.

Installing required packages on OML4Py client machine

> **Note**
>
> `onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` are to be installed on the client only. `onnxruntime`, `onnxruntime-extensions`, `onnx`, `transformers`, and `sentencepiece` support the ONNX conversion feature on the OML4Py client and should installed on the client only. All other packages are installed on both the client and server.

> **Note**
>
> `oracledb 2.1.0` or later version is required to invoke vector database SQL APIs in 23ai.

Use `pip3.12` to install the supporting packages. For OML4Py client installation of all the packages, run the following command, specifying the package:

```
pip3.12 install packagename
```

These command installs the required Python packages for on-premises OML4Py client:

```
pip3.12 install pandas==2.1.1
pip3.12 install setuptools==68.0.0
pip3.12 install scipy==1.12.0
pip3.12 install matplotlib==3.8.4
pip3.12 install oracledb==2.2.0
pip3.12 install scikit-learn==1.4.1.post1
pip3.12 install numpy==1.26.4
pip3.12 install onnxruntime==1.17.0
pip3.12 install onnxruntime-extensions==0.10.1
pip3.12 install onnx==1.16.0
pip3.12 install --extra-index-url https://download.pytorch.org/whl/cpu
torch==2.2.0+cpu
pip3.12 install transformers==4.38.1
pip3.12 install sentencepiece==0.2.0
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Installing required packages on OML4Py server machine

On the OML4Py server machine, all these packages must be installed into `$ORACLE_HOME/oml4py/modules` so they can be detected by the Embedded Python Execution process. Run the following command, specifying the package and target directory, `$ORACLE_HOME/oml4py/modules`:

```
pip3.12 install packagename --target=$ORACLE_HOME/oml4py/modules
```

These command installs the required packages:

```
pip3.12 install pandas==2.1.1--target=$ORACLE_HOME/oml4py/modules
pip3.12 install setuptools==68.0.0 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install scipy==1.12.0 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install matplotlib==3.8.4 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install oracledb==2.2.0 --target=$ORACLE_HOME/oml4py/modulesdules
pip3.12 install joblib==1.2.0 --target=$ORACLE_HOME/oml4py/modules
pip3.12 install scikit-learn==1.4.1.post1 --no-deps --target=$ORACLE_HOME/
oml4py/modules
pip3.12 install numpy==1.26.4 --target=$ORACLE_HOME/oml4py/modules
```

Verify the Package Installation

Load the packages below to ensure they have been installed successfully. Start Python and run the following commands:

```
python3
```

```
Python 3.12.0 (default, Feb 22 2022, 15:13:36)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
import numpy
import pandas
import scipy
import matplotlib
import oracledb
import sklearn
```

If all the packages are installed successfully, then no errors are returned.

4.2.4.3 Install OML4Py Server for On-Premises Oracle Database

These instructions tell how to install and uninstall the OML4Py server components for an on-premises Oracle AI Database 26ai.

- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.
- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.
- Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai
  Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.
- Verify OML4Py Installation for On-Premises Database
  Verify the installation of the OML4Py server and client components for an on-premises database.
- Grant Users the Required Privileges for On-Premises Database
  Instructions for granting the privileges required for using OML4Py with an on-premises database.
- Create New Users for On-Premises Oracle AI Database
  The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.
- Uninstall the OML4Py Server from an On-Premises Database 26ai
  Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

4.2.4.3.1 Install OML4Py Server for Linux for On-Premises Oracle AI Database 26ai

Instructions for installing the OML4Py server for Linux for an on-premises Oracle Database 26ai.

You can install OML4Py by using a Python script included in your 26ai database or by using the Database Configuration Assistant (DBCA).

Install OML4Py Using a SQL Script

To install the on-premises OML4Py server, the following are required:

Copyright © 2024, 2026, Oracle and/or its affiliates.

- A connection to the internet.
- OML4Py supporting packages. For instructions on installing the required supporting packages see Install the Required Supporting Packages for Linux for On-Premises Databases.
- Perl 5.8 or higher installed on your system.

> **Note**
>
> Perl requires the `perl-Env` package. You can install the package as root with the command `yum install perl-Env` .

To check for the existence of `perl-Env`, run the following command. The version will vary depending on your Operating System and version:

```
rpm -qa perl-Env
perl-Env-1.04-395.el8.noarch
```

- Write permission on the directories to which you download and install the server components.

> **Note**
>
> The following environment variables must be set up.

- Set environment variables: Set `PYTHONHOME` and add it to your `PATH`
- Set `ORACLE_HOME` and add it to your `PATH`
- Set `LD_LIBRARY_PATH`

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

To install the OML4Py server for Linux for an on-premises Oracle AI Database 26ai, run the server installation SQL script `pyqcfg.sql`.

> **Note**
>
> The OML4Py server needs to be installed on `CDB$ROOT` first, followed by installation on a PDB for Oracle AI Database 26ai.

1. The `pyqcfg.sql` script is under `$ORACLE_HOME/oml4py/server`. Change directory to `$ORACLE_HOME/oml4py/server`.

   ```
   cd $ORACLE_HOME/oml4py/server
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

2. At your operating system prompt, start SQL*Plus, connect to the CDB$ROOT and run the pyqcfg.sql script.

```
sqlplus / as sysdba


SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
Version 23.4.0.24.05

Copyright (c) 1982, 2024, Oracle.  All rights reserved.


Connected to:
Oracle AI
                                    Database 26ai Enterprise Edition Release
23.0.0.0.0 - Production
Version 23.4.0.24.05
```

   To capture the log, spool the installation steps to an external file install_root.txt.

```
SQL> spool install_root.txt
```

   Verify that you are currently connected to the CDB$ROOT container.

```
SQL> show con_name



CON_NAME
------------------------------
CDB$ROOT
```

   Run the pyqcfg.sql script to install OML4Py server in the CDB$ROOT first.

```
SQL> @pyqcfg.sql SYSAUX TEMP

SQL> spool off;
```

   where

   - SYSAUX is the permanent tablespace for the PYQSYS schema.
   - TEMP is the temporary tablespace for the PYQSYS schema.

   Open the install_root.txt file to see if any errors occurred.

3. At your operating system prompt, start SQL*Plus, connect to your PDB and run the pyqcfg.sql script. To capture the log, spool the installation steps to an external file install_pdb.txt. The following example uses the PDB ORCLPDB and gives example values for the script arguments.

```
sqlplus / as sysdba

SQL> spool install_pdb.txt
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Log into a PDB where you want to install OML4Py server.

```
SQL> alter session set container=ORCLPDB;
```

Run the `pyqcfg.sql` script to install OML4Py server in the PDB.

```
SQL> @pyqcfg.sql SYSAUX TEMP

SQL> spool off;
```

where

- `SYSAUX` is the permanent tablespace for the PYQSYS schema.
- `TEMP` is the temporary tablespace for the PYQSYS schema.

Open the `install_pdb.txt` file to see if any errors occurred

Verify the Server Installation

You can verify the database configuration of OML4Py as oracle user by doing the following:

1. On the OML4Py server database instance, start SQL*Plus as the OML user logging into the PDB, in this example, PDB1.

   ```
   sqlplus oml_user/oml_user_password$ORCLPDB
   ```

2. Run the following command:

   ```
   SELECT * FROM sys.pyq_config;
   ```

   The expected output is as follows:

   ```
   $ sqlplus / as sysdba;
   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 16:23:35 2024


   Copyright (c) 1982, 2024, Oracle.  All rights reserved.


   Connected to:
   Oracle AI
                                       Database 26ai Enterprise Edition Release
   23.0.0.0.0 - Production
   Version 23.4.0.24.05

   SQL> alter session set container=ORCLPDB;

   Session altered.

   SQL> select * from sys.pyq_config;

   NAME
   --------------------------------------------------------------------------------
   -----
   VALUE
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--------------------------------------------------------------------------------
-----
PYTHONHOME
/u01/app/oracle/product/23.4.0.0/dbhome_1/python

PYTHONPATH
/u01/app/oracle/product/23.4.0.0/dbhome_1/oml4y/modules

VERSION
2.0


NAME
--------------------------------------------------------------------------------
-----
VALUE
--------------------------------------------------------------------------------
-----
PLATFORM
ODB

DSWLIST
oml.*;pandas.*;numpy.*;matplotlib.*;sklearn.*
```

3. To verify the installation of the OML4Py server for an on-premises database see Verify OML4Py Installation for On-Premises Database.

- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.
- Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases
  The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

4.2.4.3.1.1 Commands Summary for Installing OML4Py Server for On-Premises Oracle Databases

The commands for installing OML4Py server for On-Premises oracle databases are listed in the following example.

Example 4-14 Install OML4Py Server for On-Premises Oracle Databases

```
export ORACLE_HOME=<ORACLE_HOME PATH>
export PYTHONHOME=$ORACLE_HOME/python
export PATH=$PYTHONHOME/bin:$ORACLE_HOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$ORACLE_HOME/lib:$LD_LIBRARY_PATH

cd $ORACLE_HOME/oml4py/server
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
$ sqlplus / as sysdba
SQL> spool install_root.txt
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;

SQL> alter session set container=ORCLPDB;
SQL> @pyqcfg.sql SYSAUX TEMP /u01/app/oracle/product/23.4.0.0/dbhome_1/python
SQL> spool off;
```

4.2.4.3.2 Verify OML4Py Installation for On-Premises Database

Verify the installation of the OML4Py server and client components for an on-premises database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

   ```
   import oml
   oml.connect(user='oml_user', password='oml_user_password', host='myhost',
               port=1521, service_name='myservice')
   ```

2. Create a user-defined Python function and store it in the OML4Py script repository.

   ```
   oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
   ```

3. Call the user-defined function, using the `oml.do_eval` function.

   ```
   res = oml.do_eval(func='TEST')
   res


   2
   ```

4. When you are finished testing, you can drop the test.

   ```
   oml.script.drop("TEST")
   ```

4.2.4.3.3 Grant Users the Required Privileges for On-Premises Database

Instructions for granting the privileges required for using OML4Py with an on-premises database.

To use OML4Py (OML4Py), a user must have certain database privileges. To store and manage user-defined Python functions in the OML4Py script repository, a user must also have the PYQADMIN database role.

User Privileges

After installing the OML4Py server on an on-premises Oracle AI Database server, grant the following privileges to any OML4Py user.

- CREATE SESSION
- CREATE TABLE

Copyright © 2024, 2026, Oracle and/or its affiliates.

- CREATE VIEW
- CREATE PROCEDURE
- CREATE MINING MODEL
- EXECUTE ON CTXSYS.CTX_DDL (required for using Oracle Text Processing capability in the algorithm classes in the `oml.algo` package)

To grant all of these privileges, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is the OML4Py user:

```
GRANT CREATE SESSION, CREATE TABLE, CREATE VIEW, CREATE PROCEDURE,
CREATE MINING MODEL, EXECUTE ON CTXSYS.CTX_DDL to oml_user;
```

Script Repository and Datastore Management

The OML4Py script repository stores user-defined Python functions that a user can invoke in an Embedded Python Execution function. An OML4Py datastore stores Python objects that can be used in subsequent Python sessions. A user-defined Python function in the script repository or a datastore can be available to any user or can be restricted for use by the owner only or by those granted access to it.

The OML4Py server installation script creates the PYQADMIN role in the database. A user must have that role to do the following:

- Store user-defined Python functions in the script repository.
- Drop user-defined Python function from the repository
- Grant or revoke permission to use a user-defined Python function in the script repository.
- Grant or revoke permission to use the objects in a datastore.

To grant this role to a user, on the on-premises Oracle database server start SQL as a database administrator and run the following SQL statement, where `oml_user` is your OML4Py user:

```
GRANT PYQADMIN to oml_user;
```

4.2.4.3.4 Create New Users for On-Premises Oracle AI Database

The `pyquser.sql` script is a convenient way to create a new OML4Py user for on on-premises database.

About the pyquser.sql Script

The `pyquser.sql` script is a component of the on-premises OML4Py server installation. The script is in the `server` directory of the installation. The sysdba privilege is required to run the script.

The `pyquser.sql` script grants the new user the required on-premises Oracle AI Database privileges and, optionally, grants the PYQADMIN database role. The PYQADMIN role is required for creating and managing scripts in the OML4Py script repository for use in Embedded Python Execution.

The `pyquser.sql` script takes the following five positional arguments:

- Username

Copyright © 2024, 2026, Oracle and/or its affiliates.

- User's permanent tablespace
- User's temporary tablespace
- Permanent tablespace quota
- PYQADMIN role

When you run the script, it prompts you for a password for the user.

Create a New User

To use the `pyquser.sql` script, go the `server` subdirectory of the directory that contains the extracted OML4Py server installation files. Run the script as a database administrator.

The following examples use SQL*Plus and the sysdba user to run the `pyquser.sql` script.

Example 4-15 Creating New Users

This example creates the user oml_user with the permanent tablespace USERS with an unlimited quota, the temporary tablespace TEMP, and grants the PYQADMIN role to the oml_user.

```
sqlplus / as sysdba
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin



Enter value for password: <type your password>
```

For a pluggable database:

```
sqlplus / as sysdba
alter session set container=<PDBNAME>
@pyquser.sql oml_user USERS TEMP unlimited pyqadmin
```

The output is similar to the following:

```
SQL> @pyquser.sql oml_user USERS TEMP unlimited pyqadmin
Enter value for password: welcome1
old   1: create user &&1 identified by &password
new   1: create user oml_user identified by welcome1
old   2: default tablespace &&2
new   2: default tablespace USERS
old   3: temporary tablespace &&3
new   3: temporary tablespace TEMP
old   4: quota &&4 on &&2
new   4: quota unlimited on USERS

User created.

old   4:     'create procedure, create mining model to &&1';
new   4:     'create procedure, create mining model to pyquser';
old   6:   IF lower('&&5') = 'pyqadmin' THEN
new   6:   IF lower('pyqadmin') = 'pyqadmin' THEN
old   7:     execute immediate 'grant PYQADMIN to &&1';
new   7:     execute immediate 'grant PYQADMIN to pyquser';
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
PL/SQL procedure successfully completed.
```

This example creates the user oml_user2 with 20 megabyte quota on the USERS tablespace, the temporary tablespace TEMP, and without the PYQADMIN role.

```
sqlplus / as sysdba
@pyquser.sql oml_user2 USERS TEMP 20M FALSE


Enter value for password: <type your password>
```

4.2.4.3.5 Uninstall the OML4Py Server from an On-Premises Database 26ai

Instructions for uninstalling the on-premises OML4Py server components from an on-premises Oracle AI Database 26ai.

Uninstall the On-Premises OML4Py Server for Linux

To uninstall the on-premises OML4Py server for Linux, do the following:

1. Verify that the `PYTHONHOME` environment variable is set to the Python3.12 directory.

   ```
   echo $PYTHONHOME
   ```

2. Verify that `PYTHONPATH` environment variable is set to the directory in which the `oml` modules are installed.

   ```
   echo $PYTHONPATH
   ```

   If it is not set to the proper directory, set it.

   ```
   export PYTHONPATH=$ORACLE_HOME/oml4py/modules
   ```

3. Change directories to the directory containing the server installation zip file.

   ```
   cd $ORACLE_HOME/oml4py
   ```

4. Run the server uninstall script `pyquncfg.sql`, the script is under `$ORACLE_HOME/oml4py/server`.

   ```
   cd $ORACLE_HOME/oml4py/server
   ```

   Start SQL*Plus, connect to the PDB.

   ```
   $ sqlplus / as sysdba;


   SQL*Plus: Release 23.0.0.0.0 - Production on Tue Apr 30 12:40:18 2024
   Version 23.4.0.24.05

   Copyright (c) 1982, 2024, Oracle. All rights reserved.
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Connected to:
Oracle AI
                                        Database 26ai Enterprise Edition Release
23.0.0.0.0 - Production
Version 23.4.0.24.05
```

Run the `pyquncfg.sql` script to record the log, spool, and the installation steps to an external file.

```
SQL> spool install.txt
SQL> alter session set container=ORCLPDB;
SQL> @pyquncfg.sql
```

4.2.4.4 Install OML4Py Client for On-Premises Oracle Database

Instructions for installing and uninstalling the on-premises OML4Py client.

For instructions on installing the OML4Py client on Autonomous Database, see Install OML4Py Client for Linux for Use With Autonomous AI Database Serverless.

To use the OML4Py client, users will need to install the following:

> **Note**
>
> The supporting packgaes need to be installed prior to the OML4Py client installation.

1. Install Python from source. See Build and Install Python for Linux for On-Premises Databases.
2. Install the Oracle Instant Client. See Install Oracle Instant Client for Linux for On-Premises Databases.
3. Install the OML4Py supporting packages. See Install the Required Supporting Packages for Linux for On-Premises Databases.
4. Install the OML4Py client. See Install OML4Py Client for Linux for On-Premises Databases.

Topics:

- Install Oracle Instant Client and the OML4Py Client for Linux
  Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.
- Verify OML4Py Client Installation for On-Premises Databases
  Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.
- Uninstall the OML4Py Client for On-Premises Databases
  Instructions for uninstalling the OML4Py client.
- Upgrade the OML4Py Client for On-Premises Databases
  Instructions for upgrading the OML4Py client.

Copyright © 2024, 2026, Oracle and/or its affiliates.

4.2.4.4.1 Install Oracle Instant Client and the OML4Py Client for Linux

Instructions for installing Oracle Instant Client and the OML4Py client for Linux for an on-premises Oracle AI Database.

To connect the OML4Py client for Linux to an on-premises Oracle AI Database, you must have Oracle Instant Client installed on your local system.

- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle AI Database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle AI Database.
- Commands Summary for Installing Oracle Instant Client and OML4Py Client for On-Premises Oracle AI Database
  Instructions for installing Oracle Instant Client and OML4Py client for On-Premises Oracle AI Database are listed in the following example.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.
- Install Oracle Instant Client for Linux for On-Premises Databases
  Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.
- Install OML4Py Client for Linux for On-Premises Databases
  Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

4.2.4.4.1.1 Install Oracle Instant Client for Linux for On-Premises Databases

Instructions for installing Oracle Instant Client for Linux for use with an on-premises Oracle database.

The OML4Py client requires Oracle Instant Client to connect to an Oracle database.

To install Oracle Instant Client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.

Copyright © 2024, 2026, Oracle and/or its affiliates.

To install Oracle Instant Client, do the following:

1. Download the Oracle Instant Client for your system. Go to the Oracle Instant Client Downloads page and select Instant Client for Linux x86-64.
2. Locate the section for your version of Oracle Database. These instructions use Version 23.4.0.0.0.
3. In the Base section, in the Download column, click the zip file for the Basic Package and save the file in an accessible directory on your system or use `wget` to download the file:

   ```
   wget https://download.oracle.com/otn_software/linux/instantclient/2340000/instantclient-basic-linux.x64-23.4.0.24.05.zip
   ```

4. Unzip the package:

   ```
   unzip instantclient-basic-linux.x64-23.4.0.24.05.zip
   ```

   Extracting the package creates the subdirectory `instantclient_23_4`, which contains the Oracle Instant Client files.
5. To install the libaio package with sudo or as the root user, run the following command:

   ```
   sudo yum install libaio
   ```

   > **Note**
   >
   > In some Linux distributions, this package is called `libaio1`.

6. Add the directory that contains the Oracle Instant Client files to the beginning of your `LD_LIBRARY_PATH` environment variable:

   ```
   export LD_LIBRARY_PATH=/opt/oracle/instantclient_23_04:$LD_LIBRARY_PATH
   ```

4.2.4.4.1.2 Install OML4Py Client for Linux for On-Premises Databases

Instructions for installing the OML4Py client for Linux for use with an on-premises Oracle database.

Prerequisites

To download and install the on-premises OML4Py client, the following are required:

- A connection to the internet.
- Write permission on the directory in which you are installing the client.
- Perl 5.8 or higher installed on your system.
- Python 3.12.0. To know more about downloading and installing Python 3.12.0, see Build and Install Python for Linux for On-Premises Databases.
- The OML4Py supporting packages. To know more about OML4Py the supporting packages, see Install the Required Supporting Packages for Linux for On-Premises Databases.

To use the OML4Py client to connect to an on-premises Oracle database, the following are required:

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Oracle Instant Client must be installed on the client machine.
- The OML4Py server must be installed on the on-premises database server.

Download and Extract the OML4Py Client Installation File

To download and extract the OML4Py client installation file, do the following:

1. Download the client installation zip file.
   a. Go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network.
   b. Accept the license agreement and select **Oracle Machine Learning for Python Downloads (v2.0)**.
   c. Select **Oracle Machine Learning for Python Client Install for Oracle Database on Linux 64 bit**.
   d. Save the zip file to an accessible directory.
2. Go to the directory to which you downloaded the zip file and unzip the file.

   ```
   unzip oml4py-client-linux-x86_64-2.0.zip
   ```

   The contents are extracted to a subdirectory named `client`, which contains these four files:

   - `OML4PInstallShared.pm`
   - `oml-2.0-cp312-cp312-linux_x86_64.whl`
   - `client.pl`
   - `oml4py.ver`

View the Optional Arguments to the Client Installation Perl Script

In the directory above the unzipped client folder, run the client installation Perl script with the `--help` option to display the arguments to the client installation Perl script.

The following command displays the available installation options:

```
perl -Iclient client/client.pl --help

Oracle Machine Learning for Python 2.0 Client.

Copyright (c) 2018, 2024 Oracle and/or its affiliates. All rights reserved.
Usage: client.pl [OPTION]...
Install, upgrade, or uninstall OML4P Client.

  -i, --install install or upgrade (default)
  -u, --uninstall uninstall
  -y never prompt
  --ask interactive mode (default)
  --no-embed do not install embedded python functionality
  --no-deps turn off dependencies checking
  --target <dir> install client into <dir>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

By default, the installation script installs the Embedded Python Execution modules. If you don't want to install this module, then you can use the `--no-embed` flag.

Also by default, the installation script checks for the existence and version of each of the supporting packages that the OML4Py client requires. If a required package is missing or does not meet the version requirement, the installation script displays an error message and exits. You can skip the dependency checking in the client installation by using the `--no-deps` flag. However, to use the `oml` module, you need to have installed acceptable versions of all of the supporting packages.

For a list of the required dependencies, see Install the Required Supporting Packages for Linux for On-Premises Databases.

Run the OML4Py Client Installation Script

To install the OML4Py client, do the following:

1. In the directory above the unzipped client folder, run the script. The following command runs the Perl script in the current directory:

   ```
   perl -Iclient client/client.pl
   ```

   Alternatively, the following command runs the Perl script with the target directory specified:

   ```
   perl -Iclient client/client.pl --target path_to_target_dir
   ```

   The `--target` flag is optional, if you don't want to install it to the current directory.

   When the script displays `Proceed?`, enter `y` or `yes`.

   If you use the `--target <dir>` argument to install the `oml` module to the specified directory, then add that location to environment variable `PYTHONPATH` so that Python can find the module:

   ```
   export PYTHONPATH=path_to_target_dir
   ```

   The command displays the following:

   ```
   perl -Iclient client/client.pl


   Oracle Machine Learning for Python 2.0 Client.

   Copyright (c) 2018, 2022 Oracle and/or its affiliates. All rights reserved.
   Checking platform .................. Pass
   Checking Python .................... Pass
   Checking dependencies .............. Pass
   Checking OML4P version ............. Pass
   Current configuration
     Python Version ................... 3.12.0
     PYTHONHOME ....................... /opt/Python-3.12.0
     Existing OML4P module version .... None
     Operation ........................ Install/Upgrade

   Proceed? [yes]
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Processing ./client/oml-2.0-cp311-cp311-linux_x86_64.whl
Installing collected packages: oml
Successfully installed oml-2.0
```

2. To verify that `oml` modules are successfully installed and are ready to use, start Python and import `oml`. At the Linux prompt, enter `python3`.

   ```
   python3
   ```

   At the Python prompt, enter `import oml`

   ```
   import oml
   ```

   The output is similar to the following:

   ```
   $ python3
   Python 3.12.0 (default, Feb 23 2022, 17:12:33)
   [GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
   Type "help", "copyright", "credits" or "license" for more information.
   >>> import oml
   >>>
   ```

3. Display the location of the installation directory.
   If you didn't use the `--target <dir>` argument, then the installed `oml` modules are stored under `$PYTHONHOME/lib/python3.12/site-packages/`. Again, you must have write permission for the target directory.

   In Python, after importing the `oml` module, you can display the directory in which the client is installed. At the Python prompt, enter:

   ```
   oml.__path__
   ```

Connect to the OML4Py Server

Start Python, import `oml`, and create a connection to your OML4Py server using an appropriate password, hostname, and system identifier. The following example uses oml_user as the user and has example argument values. Replace the username and other argument values with the values for your user and database.

```
import oml
oml.connect(user='oml_user', password='oml_user_password', host=myhost,
            port=1521, service_name=myservicename)
```

After connecting, you can run any of the examples in this publication. For example, you could run Example 8-8.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> To use the Embedded Python Execution examples, you must have installed the OML4Py client with the Embedded Python Execution option enabled.
> To use the Automatic Machine Learning (AutoML) examples, you must specify a running connection pool on the server in the `automl` argument in an `oml.connect` invocation.

4.2.4.4.2 Verify OML4Py Client Installation for On-Premises Databases

Verify the installation of the OML4Py client components for an on-premises Oracle AI Database.

1. In your local Python session, connect to the OML4Py server and invoke the same function by name. In the following example, replace the values for the parameters with those for your database.

   ```
   import oml
   oml.connect(user='oml_user', password='oml_user_password', host='myhost',
               port=1521, service_name=myservicename)
   ```

2. Create a user-defined Python function and store it in the OML4Py script repository.

   ```
   oml.script.create("TEST", func='def func():return 1 + 1', overwrite=True)
   ```

3. Call the user-defined function, using the `oml.do_eval` function.

   ```
   res = oml.do_eval(func='TEST')
   res
   ```

   ```
   2
   ```

4. When you are finished testing, you can drop the test.

   ```
   oml.script.drop("TEST")
   ```

4.2.4.4.3 Uninstall the OML4Py Client for On-Premises Databases

Instructions for uninstalling the OML4Py client.

Uninstall the On-Premises OML4Py Client for Linux

To uninstall the on-premises OML4Py client for Linux, from the directory containing the client installation zip file, run the client installation Perl script with the `-u` argument:

```
perl -Iclient client/client.pl -u
```

When the script displays `Proceed?`, enter `y` or `yes`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

If the client is successfully uninstalled, you'll see the following message:

```
Uninstalling oml-2.0:
  Successfully uninstalled oml-2.0
```

4.2.4.4.4 Upgrade the OML4Py Client for On-Premises Databases

Instructions for upgrading the OML4Py client.

If you already have OML4Py 2.0 client installed and would like to upgrade to 2.0.1 client, do the following:

1. Uninstall OML4Py 2.0 client by following the instructions in Uninstall the OML4Py 2.0 Client for On-Premises Databases.
2. Install OML4Py 2.0.1 client by following the instructions in Install OML4Py 2.0.1 Client for Linux for On-Premises Databases..

Copyright © 2024, 2026, Oracle and/or its affiliates.

5

Install OML4Py Client Container

Use the steps provided below to configure and install the container following best practices with Bash scripts.

The Oracle Machine Learning for Python (OML4Py) client container provides a preconfigured runtime environment that includes Python, Oracle client libraries, and the supporting packages required for OML4Py 2.1.1. Using the container simplifies client installation, minimizes dependency conflicts, and helps ensure a consistent runtime environment across systems.

The container includes the OML4Py client, version 2.1.1, and is designed to work with OML4Py on both Oracle AI Database (on premises and in the cloud) and Oracle Autonomous AI Database.

The OML4Py 2.1.1 client container runs on Oracle Linux 8, 9, and 10 (x86_64) and includes Python 3.13.5 and the preconfigured supporting Python packages listed below:

```
pandas==2.2.3
setuptools==80.10.2
scipy==1.14.1
matplotlib==3.10.4
oracledb==3.3.0
scikit-learn==1.6.1
numpy==2.1.0
onnxruntime==1.20.0
onnxruntime-extensions==0.12.0
onnx==1.21.0
torch==2.10.0
transformers==4.56.1
sentencepiece==0.2.1
```

Rootless Podman Prerequisites

Rootless Podman requires the `/etc/subuid` and `/etc/subgid` files to map user IDs inside the container to unprivileged user IDs on the host. These files must define a user and group ID range for each user, typically at least 65,536 IDs.

1. Configure subordinate user and group ID ranges:

   ```
   sudo usermod \
   --add-subuids 100000-165535 \
   --add-subgids 100000-165535 \
   $USER
   ```

2. Apply the updated configuration:

   ```
   podman system migrate
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

3. Verify the ID mappings:

```
grep $USER /etc/subuid
```

Installation Steps

Steps to install and run the OML4Py Client Container (version 2.1.1) using Podman.

1. Verify Podman Prerequisites
   Confirm that Podman is installed and functioning correctly:

```
podman version
```

   If the command fails or returns an error such as `podman: command not found`, install Podman using the appropriate command for your Oracle Linux version.

   - Oracle Linux 8

```
sudo dnf module install container-tools:ol8
```

   - Oracle Linux 9 or Oracle Linux 10

```
 sudo dnf install -y container-tools
```

   After installation, rerun:

```
podman version
```

2. Sign In to Oracle Container Registry (OCR)
   See Getting Started with the OML4Py 2.1.1 Client Container for complete Oracle Container Registry setup instructions.
   a. Navigate to Oracle Container Registry and select **Sign In**.
   b. Sign in using your Oracle SSO credentials.
      If this is your first time using OCR, complete the one-time registration process to enable OCR access for your Oracle SSO account.
   c. Review and accept the license agreement. This step is required once per user/account.
3. Generate an OCR Authentication Token
   To pull the OML4Py container image with Podman, sign in to OCR using `podman login` command with your Oracle SSO username and an OCR authentication token. Generate the token by following the instructions in Generating an Oracle Container Registry Authentication Token in the Podman User's Guide.

> **Note**
>
> Save the authentication token in a secure location immediately after it is generated. The token is displayed only once. If you lose it, you must delete the existing token and generate a new one.

4. Log In to OCR from Podman and Pull the Image

Copyright © 2024, 2026, Oracle and/or its affiliates.

a. Log in to OCR:

```
podman login container-registry.oracle.com
```

When prompted, enter the following credentials:

- **Username:** your Oracle SSO username
- **Password:** the generated OCR authentication token from the previous step

b. Pull the image:

```
podman pull container-registry.oracle.com/database/oml4py-
client:2.1.1.0.0
```

5. Verify the Image Was Pulled Successfully
To confirm the image is available locally, list the Podman images on your system and verify that `container-registry.oracle.com/database/oml4py-client:2.1.1.0.0` appears in the output:

```
podman images
```

6. (Optional) Use the Image in an Air-Gapped Environment
If the target host does not have internet access, save the OML4Py container image to a tar file on a connected host. Transfer the file to the air-gapped host, and then load the image locally.

a. Save the image to a tar file on a connected host:

```
podman save -o oml4py-client_2.1.1.0.0.tar container-
registry.oracle.com/database/oml4py-client:2.1.1.0.0
```

b. After transferring the tar file to the air-gapped host, load the image:

```
podman load -i oml4py-client_2.1.1.0.0.tar
```

c. Verify the image is available on the air-gapped host:

```
podman images
```

Confirm that `container-registry.oracle.com/database/oml4py-client:2.1.1.0.0` appears in the output.

7. Launch the OML4Py Client Container
Use the following instructions to start a container from the image.

> **Note**
>
> If you plan to use AutoML, ensure that pooled connection entries are added to `tnsnames.ora`. See Configure Pooled Connections for AutoML for more information.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> Use `/u01/oml_home` for any files you need to write inside the container, such as exported models or output files. This directory is the designated writable directory within the container environment.

You can start the container either with or without an Oracle wallet, depending on your database connection configuration. Choose from:

- Option A: Run without an Oracle wallet
  If you are not using an Oracle wallet to connect to the database, run:

  ```
  podman run -it --name <container_name> \
  container-registry.oracle.com/database/oml4py-client:2.1.1.0.0
  ```

- Option B: Run With an Oracle Wallet
  If you are using an Oracle wallet, mount the wallet directory into the container and set `TNS_ADMIN` accordingly.

  Prepare the Wallet:

  a. Upload the wallet to the host and extract it to a directory. For example:

     ```
     unzip <wallet_name>.zip -d /home/opc/wallet
     ```

  b. Navigate to the wallet directory. Verify that `sqlnet.ora` contains the wallet directory configuration.

     ```
     cd /home/opc/wallet
     cat sqlnet.ora
     ```

  c. Update `sqlnet.ora` so that the wallet location points to `/u01/oml_home/wallet` inside the container:

     ```
     sed -i.bak 's|?/network/admin|/u01/oml_home/wallet|g'
     <wallet_file_path>/sqlnet.ora
     ```

  d. Verify that `sqlnet.ora` contains a wallet location similar to the following:

     ```
     WALLET_LOCATION =
       (SOURCE =
         (METHOD = file)
         (METHOD_DATA =
           (DIRECTORY="/u01/oml_home/wallet")))
     ```

  Start the Container with the Wallet Mounted

  ```
  podman run -it --name <container_name> \
  -v <wallet_file_path>:/u01/oml_home/wallet:U,Z \
  -e TNS_ADMIN=/u01/oml_home/wallet \
  container-registry.oracle.com/database/oml4py-client:2.1.1.0.0
  ```

- (Optional) Mount a Host Directory for Container Output

Copyright © 2024, 2026, Oracle and/or its affiliates.

If you want to write files from the container to the host, mount a host directory into the container:

```
podman run -it --name <container_name> \
-v <host_output_folder>:/u01/oml_home/:U,Z \
container-registry.oracle.com/database/oml4py-client:2.1.1.0.0
```

8. Convert and Export an ONNX Model from the Container
After starting the OML4Py client container, you can run the ONNX model conversion workflow from a Python session.

   a. Inside the container, start Python:

   ```
   python3
   ```

   b. Then convert your model. In Python, list the available preconfigured models, create a pipeline for a selected model, and export the model to a file or to the database:

   ```
   import oml
   from oml.utils import ONNXPipeline, ONNXPipelineConfig

   # Display available preconfigured models
   ONNXPipelineConfig.show_preconfigured()

   # Create a pipeline for a selected model
   pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
   v2")

   # Connect to the database
   oml.connect(user="<user>", password="<password>",
   dsn="<service_name>")   # username/password
   oml.connect(user="", password="",
   dsn="<tns_alias>")                      # wallet-based

   # Export the model
   pipeline.export2db("YOUR_MODEL_NAME")     # directly to the database
   pipeline.export2file("YOUR_MODEL_NAME")   # to a file in the container
   ```

   Replace `YOUR_MODEL_NAME` with the name to use for the exported ONNX model. Converted ONNX models can be deployed to Oracle AI Database 26ai or to the Oracle Private AI Services Container. Deploying models to Oracle AI Database enables in-database embedding generation and semantic search workflows using Oracle AI Vector Search.

   Deploying models to the Oracle Private AI Services Container enables embedding generation outside the database through an OpenAI-compatible REST API.

9. (Optional) Load an Exported ONNX Model into the Database
If you exported the model to a file and want to load it into the database manually, move the `.onnx` file to a location accessible by the database server.

   Then, connect as a DBA user and run the following statements:

   ```
   CREATE DIRECTORY ONNX_IMPORT AS '/path/to/models';

   GRANT READ ON DIRECTORY ONNX_IMPORT TO YOUR_USER;
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
GRANT CREATE MINING MODEL TO YOUR_USER;

BEGIN
  DBMS_VECTOR.LOAD_ONNX_MODEL(
    directory  => 'ONNX_IMPORT',
    file_name  => 'YOUR_MODEL_NAME.onnx',
    model_name => 'YOUR_MODEL_NAME'
  );
END;
/
```

10. (Optional) Copy the Converted Model File from the Container
    Exit the container first by typing `exit` or pressing `Ctrl+D` to return to the host shell. Then use the `podman cp` command to copy the converted model file from the container to the host:

    ```
    podman cp <container_name>:<path_to_file> <host_output_folder>
    ```

11. Reconnect to an Existing Container
    To restart and reconnect to an existing container, run:

    ```
    podman start <container_name>
    podman exec -it <container_name> /bin/bash
    ```

- Configure Pooled Connections for AutoML
  AutoML operations require a pooled database connection. To enable pooled connections, add pooled service entries to the `tnsnames.ora` file included in the Oracle wallet used by the container.

5.1 Configure Pooled Connections for AutoML

AutoML operations require a pooled database connection. To enable pooled connections, add pooled service entries to the `tnsnames.ora` file included in the Oracle wallet used by the container.

Add a Pooled Connection Entry

Locate the existing service entry in `tnsnames.ora` and add a corresponding pooled entry., such as `myadb_medium_pool`.

The required configuration change is the addition of: `SERVER=POOLED)` inside the *connect_data* section of the service entry.

The following example shows the pooled services entry that can be added to the `tnsnames.ora` file:

- ```
  myadb_low_pool = (description= (retry_count=20)(retry_delay=3)
  (address=(https_proxy=your proxy address here)(https_proxy_port=80)
  (protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
  (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
  (SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
  sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
  City,ST=California,C=US")))
  ```
- ```
  myadb_medium_pool = (description= (retry_count=20)(retry_delay=3)
  (address=(https_proxy=your proxy address here)(https_proxy_port=80)
  ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
(protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
(connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
(SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
City,ST=California,C=US")))
```

- ```
  myadb_high_pool = (description= (retry_count=20)(retry_delay=3)
  (address=(https_proxy=your proxy address here)(https_proxy_port=80)
  (protocol=tcps)(port=1522)(host=adb.us-sanjose-1.oraclecloud.com))
  (connect_data=(service_name=qtraya2braestch_myadb_medium.adb.oraclecloud.com)
  (SERVER=POOLED))(security=(ssl_server_cert_dn="CN=adb.us-
  sanjose-1.oraclecloud.com,OU=Oracle ADB SANJOSE,O=Oracle Corporation,L=Redwood
  City,ST=California,C=US")))
  ```

Replace the service names with the Autonomous AI Database service names used in your environment.

Connect Using AutoML

When connecting through OML4Py, specify:

- the standard database service using the `dsn` parameter, and
- the pooled service using the `automl` parameter.

For example:

```
oml.connect(
    user="<user>",
    password="<password>",
    dsn="myadb_medium",
    automl="myadb_medium_pool"
)
```

In this example:

- `dsn` specifies the standard database service.
- `automl` specifies the pooled service used for AutoML operations.

Copyright © 2024, 2026, Oracle and/or its affiliates.
```

6

Install OML4Py on Exadata

The following topics cover the installation and configuration of OML4Py on Exadata, Exadata Cloud at Customer, and RAC.

Topics:

- About Oracle Machine Learning for Python on Exadata
  Exadata is an ideal platform for OML4Py. The parallel resources of Python computations in OML4Py take advantage of the massively parallel grid infrastructure of Exadata.
- Configure DCLI to install Python across Exadata compute nodes.
  Using Distributed Command Line Interface (DCLI) can simplify the installation of OML4Py on Exadata.

6.1 About Oracle Machine Learning for Python on Exadata

Exadata is an ideal platform for OML4Py. The parallel resources of Python computations in OML4Py take advantage of the massively parallel grid infrastructure of Exadata.

> **Note**
>
> The version of OML4Py must be the same on the server and on each client computer. Also, the version of Python must be the same on the server and on each client computer. See Database and System Requirements for OML4Py for supported configurations.

To install OML4Py on Exadata:

1. On all compute nodes:
   - Install Python
   - Verify and configure the environment
   - Install the OML4Py supporting pacakges
   - Install the OML4Py server components
2. On the first node only:
   - Install the OML4Py Server components including the database configuration.
   - Create an OML4Py user, if desired. Alternatively, configure an existing database user to use OML4Py. See Create New Users for On-Premises Oracle AI Database.

You can simplify the Python installation on Exadata by using the Distributed Command Line Interface (DCLI).

Copyright © 2024, 2026, Oracle and/or its affiliates.

6.2 Configure DCLI to install Python across Exadata compute nodes.

Using Distributed Command Line Interface (DCLI) can simplify the installation of OML4Py on Exadata.

With DCLI, you can use a single command to install Python across multiple Exadata compute nodes. The following example shows the output of the DCLI help option, which explains the basic syntax of the utility.

Example 6-1 DCLI Help Option Output

```
dcli -h


Distributed Shell for Oracle Storage

This script executes commands on multiple cells in parallel threads.
The cells are referenced by their domain name or ip address.
Local files can be copied to cells and executed on cells.
This tool does not support interactive sessions with host applications.
Use of this tool assumes ssh is running on local host and cells.
The -k option should be used initially to perform key exchange with
cells.  User may be prompted to acknowledge cell authenticity, and
may be prompted for the remote user password.  This -k step is serialized
to prevent overlayed prompts.  After -k option is used once, then
subsequent commands to the same cells do not require -k and will not require
passwords for that user from the host.
Command output (stdout and stderr) is collected and displayed after the
copy and command execution has finished on all cells.
Options allow this command output to be abbreviated.

Return values:
 0 -- file or command was copied and executed successfully on all cells
 1 -- one or more cells could not be reached or remote execution
      returned non-zero status.
 2 -- An error prevented any command execution

Examples:
 dcli -g mycells -k
 dcli -c stsd2s2,stsd2s3 vmstat
 dcli -g mycells cellcli -e alter iormplan active
 dcli -g mycells -x reConfig.scl

Usage: dcli [options] [command]

Options:
  --version             show program's version number and exit
  --batchsize=MAXTHDS   limit the number of target cells on which to run the
                        command or file copy in parallel
  -c CELLS              comma-separated list of cells
  --ctimeout=CTIMEOUT   Maximum time in seconds for initial cell connect
  -d DESTFILE           destination directory or file
  -f FILE               files to be copied
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
-g GROUPFILE          file containing list of cells
-h, --help            show help message and exit
--hidestderr          hide stderr for remotely executed commands in ssh
-k                    push ssh key to cell's authorized_keys file
--key-with-one-password
                      apply one credential for pushing ssh key to
                      authorized_keys files
-l USERID             user to login as on remote cells (default: celladmin)
--root-exadatatmp     root user login using directory /var/log/exadatatmp/
--maxlines=MAXLINES   limit output lines from a cell when in parallel
                      execution over multiple cells (default: 100000)
-n                    abbreviate non-error output
-r REGEXP             abbreviate output lines matching a regular expression
-s SSHOPTIONS         string of options passed through to ssh
--scp=SCPOPTIONS      string of options passed through to scp if different
                      from sshoptions
--serial              serialize execution over the cells
--showbanner          show banner of the remote node in ssh
-t                    list target cells
--unkey               drop keys from target cells' authorized_keys file
-v                    print extra messages to stdout
--vmstat=VMSTATOPS    vmstat command options
-x EXECFILE           file to be copied and executed
```

Configure the Exadata environment to enable automatic authentication for DCLI on each compute node.

1. Generate an SSH public-private key for the root user. Execute the following command as root on any node:

   ```
   ssh-keygen -N '' -f /.ssh/id_dsa -t dsa
   ```

   This command generates public and private key files in the `.ssh` subdirectory of the home directory of the root user.

2. In a text editor, create a file that contains the names of all the compute nodes in the rack. Specify each node name on a separate line. For example, the `nodes` file for a 2-node cluster could contain entries like the following:

   ```
   cat nodes


   exadb01
   exadb02
   ```

3. Run the DCLI command with the `-k` option to establish SSH trust across all the nodes. The `-k` option causes DCLI to contact each node sequentially (not in parallel) and prompts you to enter the password for each node.

   ```
   dcli -t -g nodes -l root -k -s "\-o StrictHostkeyChecking=no"
   ```

   DCLI with `-k` establishes SSH Trust and User Equivalence. Subsequent DCLI commands will not prompt for passwords.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Instructions for installing Python and OML4Py across Exadata compute nodes using DCLI are described in the following topics.

Topics:

- Install Python across Exadata compute nodes using DCLI
  Instructions for installing Python across Exadata compute nodes using DCLI.
- Install OML4Py across Exadata compute nodes using DCLI
  Instructions for installing OML4Py across Exadata compute nodes using DCLI.

6.2.1 Install Python across Exadata compute nodes using DCLI

Instructions for installing Python across Exadata compute nodes using DCLI.

These steps describe building and installing Python for Exdata.

1. Go to the Python website and download the Python 3.12.0 **XZ compressed source tarball** and untar it. The downloaded file name is `Python-3.12.0.tgz`

   ```
   wget https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tgz
   tar xvf Python-3.12.0.tgz
   ```

2. OML4Py requires the presence of the `perl-Env libffi-devel`, `openssl`, `openssl-devel`, `tk-devel`, `xz-devel`, `zlib-devel`, `bzip2-devel`, `readline-devel` and `libuuid-devel` libraries. Install these libraries using the command:

   ```
   dcli -t -g nodes -l root "yum -y install perl-Env libffi-devel openssl
   openssl-devel tk-devel xz-devel zlib-devel bzip2-devel readline-devel
   libuuid-devel"
   ```

3. Set the PYTHONHOME environment on each node:

   ```
   dcli -t -g nodes -l oracle "export PYTHONHOME=$ORACLE_HOME/python; export
   PATH=$ORACLE_HOME/python/bin$PATH; export LD_LIBRARY_PATH=$ORACLE_HOME/
   python/lib:$LD_LIBRARY_PATH; export PIP_REQUIRE_VIRTUALENV=false"

   dcli -t -g nodes -l oracle "tar xvfz $ORACLE_HOME/Python-3.12.0.tgz -
   C $ORACLE_HOME/python"

   dcli -t -g nodes -l oracle "cd $ORACLE_HOME/python; ./configure --enable-
   shared --prefix=$ORACLE_HOME/python"

   dcli -t -g nodes -l oracle "cd $ORACLE_HOME/python; make clean; make"

   dcli -t -g nodes -l oracle "cd $ORACLE_HOME/python; make altinstall"
   ```

4. Create a symbolic link in your `$PYTHONHOME/bin` directory. You need to link it to your Python-3.12 executable, which you can do with the following commands:

   ```
   dcli -t -g nodes -l oracle "cd $PYTHONHOME/bin"
   dcli -t -g nodes -l oracle "ln -s python3.12 python3"
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

5. Set environment variable `PYTHONHOME` and add it to your `PATH`, and set environment variable `LD_LIBRARY_PATH`:

```
dcli -t -g nodes -l oracle "export PYTHONHOME=$ORACLE_HOME/python"
dcli -t -g nodes -l oracle "export PATH=$PYTHONHOME/bin:$PATH"
dcli -t -g nodes -l oracle "export LD_LIBRARY_PATH=$PYTHONHOME/
lib:$LD_LIBRARY_PATH"
dcli -t -g nodes -l oracle "export PIP_REQUIRE_VIRTUALENV=false"
```

6. You can now start Python by running the command python3. For example:

```
dcli -t -g nodes -l oracle "python3"


exadb01: Python 3.12.0 (default, Feb 10 2022, 14:38:12)
    [GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
    Type "help", "copyright", "credits" or "license" for more information.

exadb02: Python 3.12.0 (default, Feb 10 2022, 14:38:12)
    [GCC 4.8.5 20150623 (Red Hat 4.8.5-44.0.3)] on linux
    Type "help", "copyright", "credits" or "license" for more information.
```

6.2.2 Install OML4Py across Exadata compute nodes using DCLI

Instructions for installing OML4Py across Exadata compute nodes using DCLI.

To install OML4Py on Exadata using DCLI, follow the steps:

1. First install the OML4Py supporting packages to `$ORACLE_HOME/oml4py/modules` on each node. Create a file named `requirements.txt` containing the following content:

```
pandas==2.1.1
setuptools==68.0.0
scipy==1.12.0
matplotlib==3.8.4
oracledb==2.2.0
scikit-learn==1.4.1.post1 --no-deps
numpy==1.26.4
```

2. Install the packages with `requirements.txt`.

```
pip3.12 install -r requirements.txt --target=$ORACLE_HOME/oml4py/modules
```

3. Set the `PYTHONPATH` environment variable to the location of the OML4Py modules:

```
export PYTHONPATH=$ORACLE_HOME/oml4py/modules
```

4. Download the installation file for your system.
   a. Go to the Oracle Machine Learning for Python Downloads page on the Oracle Technology Network.
   b. Accept the license agreement and **select Oracle Machine Learning for Python Downloads (v2.0)**.

Copyright © 2024, 2026, Oracle and/or its affiliates.

c. Select **Oracle Machine Learning for Python Server Install for Oracle AI Database on Linux 64 bit**.

d. Save the file to the `$ORACLE_HOME/oml4py` directory.

To extract the installation file to `$ORACLE_HOME/oml4py` directory, use the command:

```
unzip oml4py-server-linux-x86_64-2.0.zip -d $ORACLE_HOME/oml4py
```

The files are extracted to the `$ORACLE_HOME/oml4py/server` subdirectory.

5. On the first node, from the `$ORACLE_HOME/oml4py` directory, run the server installation script. The following command runs the script in interactive mode:

```
perl -Iserver server/server.pl
```

To run the server script in non-interactive mode, pass the parameters for the pluggable database, and permanent and temporary tablespaces to the script

```
perl -Iserver server/server.pl -y --pdb PDB11 --perm SYSTEM --temp TEMP
```

Run the server script with the --no-db flag on all remaining compute nodes. This sets up the OML4Py server configuration and skips the database configuration steps already performed on the first node:

```
perl -Iserver server/server.pl --no-db
```

For DCLI, we change it to:

> **Note**
>
> The OML4Py supporting packages must be installed individually on each compute node. DCLI cannot be used because it uses the system default Python and cause conflicts with the Python installed for use with OML4Py.

```
dcli -t -g nodes -l oracle "perl -Iserver server/server.pl --no-db"
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

7

Install Third-Party Packages

Oracle Machine Learning Notebooks in the Autonomous AI Database provides a conda interpreter to install third-party Python libraries in a conda environment for use within OML Notebooks sessions and OML4Py embedded execution invocations. Conda is an open-source package and environment management system that enables the use of environments containing third-party Python libraries.

Administrators create conda environments and install packages that can then be accessed by non-administrator users and loaded into their OML Notebooks session. The conda environments can be used by OML4Py Python, SQL, and REST APIs.

> **Note**
>
> - None of the OML features that come with ADB require the customer to install any additional third-party software via the conda feature.
> - When installing third-party software using the conda feature, vulnerability management and license compliance of that software is the sole responsibility of the customer who installed it, not Oracle.

Topics:

- Conda Commands
  This topic contains common commands used by ADMIN while creating and testing conda environments in Autonomous AI Database. Conda is an open-source package and environment management system that enables the use of environments containing third-party Python libraries.
- Administrative Tasks for Creating and Saving a Conda Environment
  In OML Notebooks, user ADMIN can manage the lifecycle of the OML user's conda environments, including creating and deleting environments and installing and deleting packages.
- OML User Tasks for Downloading an Available Conda Environment
  Once user ADMIN installs the environment in Object Storage in the Autonomous AI Database, as an OML user, you can download, activate, and use it in Python paragraphs in notebooks and with embedded execution.
- Using Conda Environments with Embedded Python Execution
  This topic explains the usage of conda environments by running user-defined functions (UDFs) in SQL and REST APIs for embedded Python execution.
- Conda Environment Best Practices
  This section highlights the best practices for creating conda environments in OML4Py to prevent Python and third/fourth-party dependency conflicts.

7.1 Conda Commands

This topic contains common commands used by ADMIN while creating and testing conda environments in Autonomous AI Database. Conda is an open-source package and

Copyright © 2024, 2026, Oracle and/or its affiliates.

environment management system that enables the use of environments containing third-party Python libraries.

Refer to Conda Interpreter Commands for a table of supported conda commands.

Conda Help

To get help for conda commands, run the command name followed by the `--help` flag.

> **Note**
>
> The conda command is not run explicitly because the `%conda` interpreter provides the conda context.

- Get help for all conda commands

  ```
  %conda

  --help
  ```

- Get help for a specific conda command. Run the following command to get help with the `install` command:

  ```
  %conda

  install --help
  ```

Conda Info

The `info` command displays information about the conda installation, including the conda version and available channels.

```
%conda

info
```

Conda Search

The `search` command allows the user to search for packages and display associated information, including the package version and the channel where it resides.

- Search for a specific package. Run the following command to search for the package `scikit-learn`.

  ```
  %conda

  search scikit-learn
  ```

- Search for packages containing 'scikit' in the package name.

  ```
  %conda

  search '*scikit*'
  ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

- Search for a specific version of a package.

```
%conda

search 'numpy==1.12'
```

```
%conda

search 'numpy>=1.12'
```

- Search for a specific version on a specific channel.

```
%conda

search conda-forge::numpy
```

Enhanced Conda Commands

A set of enhanced conda commands in the conda environment lifecycle management package `env-lcm` supports the management of environments saved to Object Storage, including uploading, downloading, listing, and deleting available environments.

Help for conda lifecycle environment commands.

```
%conda

env-lcm --help
```

```
Usage: conda-env-lcm [OPTIONS] COMMAND [ARGS]...

  ADB-S Command Line Interface (CLI) to manage persistence of conda
  environments

Options:
  -v, --version  Show the version and exit.
  --help         Show this message and exit.

Commands:
  delete           Delete a saved conda environment
  download         Download a saved conda environment
  import           Create or update a conda environment from saved metadata
  list-local-envs  List locally available environments for use
  list-saved-envs  List saved conda environments
  upload           Save conda environment for later use
```

Creating Conda Environments

This section demonstrates creating and installing packages to a conda environment, then removing the environment. Here commonly used options available for environment creation and testing are illustrated. The environment exists for the duration of the notebook session and does not persist between sessions unless it is saved to Object Storage. For instructions that include both creating and persisting an environment for OML users, refer to Administrative task to create and save the conda environments. As an ADMIN user:

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. Use the create command to create an environment `myenv` and install the Python keras package.
2. Verify that the new environment is created, and activate the environment.
3. Install, then uninstall an additional Python package, pytorch, in the environment.
4. Deactivate and remove the environment.

> **Note**
>
> The ADMIN user can access the conda environment from Python and R, but does not have the capability to run embedded Python and R execution commands.

For help with the conda `create` command, enter `create --help` in a `%conda` paragraph.

List Environments

Start by listing the environments available by default. Conda contains default environments with some core system libraries and conda dependencies. The active environment is marked with an asterisk (*).

```
%conda

env list


# conda environments:
#
base                  * /usr
conda-pack-env          /usr/envs/conda-pack-env
```

Create Conda Environment

Create conda environment called `myenv` with Python 3.10.x for OML4Py compatibility and install the keras package.

```
%conda

create -n myenv python=3.10.x keras
```

Verify Environment Creation

Verify the myenv environment is in the list of environments. The asterisk (*) indicates active environments. The new environment is created but not activated.

```
%conda

env list


# conda environments:
#
myenv                   /u01/.conda/envs/myenv
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
base                  * /usr
conda-pack-env          /usr/envs/conda-pack-env
```

Activate the Environment

Activate the myenv environment and list the environments to verify the activation. The asterisk (*) next to the environment name confirms the activation.

```
%conda

activate myenv


Conda environment 'myenv' activated
```

List the environments available by default.

```
%conda

env list


# conda environments:
#
myenv                 * /u01/.conda/envs/myenv
base                    /usr
conda-pack-env          /usr/envs/conda-pack-env
```

Installing and Uninstalling Libraries

The ADMIN user can install and uninstall libraries into an environment using the `install` and `uninstall` commands. For help with the conda `install` and `uninstall` commands, type `install --help` and `uninstall --help` in a `%conda` paragraph.

> **Note**
>
> When conda installs a package into an environment, it also installs any required dependencies. As shown here, it's possible to install packages to an existing environment. As a best practice, to avoid dependency conflicts, simultaneously install all the packages you need in a specific environment.

Install Additional Packages

Install the pytorch package into the activated myenv environment.

```
%conda

install pytorch
```

List Packages in the Current Environment

Copyright © 2024, 2026, Oracle and/or its affiliates.

List the packages installed in the current environment, and confirm that keras and pytorch are installed.

```
%conda

list
```

The output is similar to the following:

```
# packages in environment at /u01/.conda/envs/myenv:
#
# Name                    Version                   Build  Channel
_libgcc_mutex             0.1                        main
_openmp_mutex             5.1                       1_gnu
blas                      1.0                         mkl
.
.
.
fftw                      3.3.9                h27cfd23_1
future                    0.18.2           py310h06a4308_1
intel-openmp              2021.4.0           h06a4308_3561
keras                     2.10.0           py310h06a4308_0
keras-preprocessing       1.1.2               pyhd3eb1b0_0
ld_impl_linux-64          2.38                 h1181459_1
libffi                    3.3                  he6710b0_2
libgcc-ng                 11.2.0               h1234567_1
.
.
.
numpy-base                1.23.3           py310h8e6c178_1
openssl                   1.1.1s               h7f8727e_0
pip                       22.2.2           py310h06a4308_0
pycparser                 2.21                    pypi_0    pypi
python                    3.10.6               haa1d7c7_1
pytorch                   1.10.2          cpu_py310h6894f24_0
readline                  8.2                  h5eee18b_0
.
.
.
xz                        5.2.6                h5eee18b_0
zlib                      1.2.13               h5eee18b_0
```

The output above has been truncated and does not show the complete list of packages.

Uninstall Package

Libraries can be uninstalled from an environment using the uninstall command. Let's uninstall the pytorch package from the current environment.

```
%conda

uninstall pytorch
```

Verify Package was Uninstalled

Copyright © 2024, 2026, Oracle and/or its affiliates.

List packages in current environment and verify that the pytorch package was uninstalled.

```
%conda

list
```

The output shown below does not contain the `pytorch` package.

```
# packages in environment at /u01/.conda/envs/myenv:
#
# Name                    Version                   Build  Channel
_libgcc_mutex             0.1                        main
_openmp_mutex             5.1                       1_gnu
blas                      1.0                         mkl
bzip2                     1.0.8                h7b6447c_0
ca-certificates           2022.10.11           h06a4308_0
certifi                   2022.9.24       py310h06a4308_0
cffi                      1.15.1          py310h74dc2b5_0
fftw                      3.3.9                h27cfd23_1
future                    0.18.2          py310h06a4308_1
intel-openmp              2021.4.0           h06a4308_3561
keras                     2.10.0          py310h06a4308_0
keras-preprocessing       1.1.2              pyhd3eb1b0_0
ld_impl_linux-64          2.38                 h1181459_1
libffi                    3.3                  he6710b0_2
libgcc-ng                 11.2.0               h1234567_1
libgfortran-ng            11.2.0               h00389a5_1
libgfortran5              11.2.0               h1234567_1
libgomp                   11.2.0               h1234567_1
libstdcxx-ng              11.2.0               h1234567_1
libuuid                   1.0.3                h7f8727e_2
mkl                       2021.4.0           h06a4308_640
mkl-service               2.4.0           py310h7f8727e_0
mkl_fft                   1.3.1           py310hd6ae3a3_0
mkl_random                1.2.2           py310h00e6091_0
ncurses                   6.3                  h5eee18b_3
ninja                     1.10.2               h06a4308_5
ninja-base                1.10.2               hd09550d_5
numpy                     1.23.3          py310hd5efca6_1
numpy-base                1.23.3          py310h8e6c178_1
openssl                   1.1.1s               h7f8727e_0
pip                       22.2.2          py310h06a4308_0
pycparser                 2.21                      pypi_0    pypi
python                    3.10.6               haa1d7c7_1
readline                  8.2                  h5eee18b_0
scipy                     1.9.3           py310hd5efca6_0
setuptools                65.5.0          py310h06a4308_0
six                       1.16.0             pyhd3eb1b0_1
sqlite                    3.39.3               h5082296_0
tk                        8.6.12               h1ccaba5_0
typing-extensions         4.3.0           py310h06a4308_0
typing_extensions         4.3.0           py310h06a4308_0
tzdata                    2022f                h04d1e81_0
wheel                     0.37.1             pyhd3eb1b0_0
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
xz                        5.2.6                h5eee18b_0
zlib                      1.2.13               h5eee18b_0
```

Removing Environments

If you don't intend to upload the environment to Object Storage for the OML users in the database, you can simply exit the notebook session and it will go out of scope. Alternatively, it can be explicitly removed using the env remove command. Remove the myenv environment and verify it was removed. A best practice is to deactivate the environment prior to removal. For help on the `env remove` command, type `env remove --help` in the `%conda` interpreter.

- Deactivate the environment.

  ```
  %conda

  deactivate


  Conda environment deactivated
  ```

- Remove the environment.

  ```
  %conda

  env remove -n myenv
  ```

  List the environment to see if the environment is removed or not.

  ```
  env list


  # conda environments:
  #
  myrenv                   /u01/.conda/envs/myrenv
  base                  *  /usr
  conda-pack-env           /usr/envs/conda-pack-env
  ```

  Remove all packages in environment `/u01/.conda/envs/myenv`.

Specify Packages for Installation

Install Packages from the conda-forge Channel

Conda channels are the locations where packages are stored. They serve as the base for hosting and managing packages. Conda packages are downloaded from remote channels, which are URLs to directories containing conda packages. The conda command searches a set of channels. By default, packages are automatically downloaded and updated from the default channel. The conda-forge channel is free for all to use. You can modify what remote channels are automatically searched. You might want to do this to maintain a private or internal channel. We use the conda-forge channel, a community channel made up of thousands of contributors, in the following examples.

- Install a specific version of a Package.

  To install a specific version of a package, use `<package_name>=<version>`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Create an environment using conda-forge.

```
%conda

create -n mychannelenv -c conda-forge python=3.10.x

activate mychannelenv
```

- Install a package from conda-forge by specifying the channel.

```
%conda

install scipy --channel conda-forge
```

- Install a specific version of a package.

```
%conda

install scipy=0.15.0
```

7.2 Administrative Tasks for Creating and Saving a Conda Environment

In OML Notebooks, user ADMIN can manage the lifecycle of the OML user's conda environments, including creating and deleting environments and installing and deleting packages.

The conda environments created by user ADMIN are stored in an Object Storage bucket folder associated with the Autonomous AI Database instance. OML users can download these conda environments using enhanced conda commands. Conda environments are available after they are downloaded and activated using the download and activate functions in a `%conda` paragraph. An activated environment is available until it is deactivated.

Create a Conda enviroment

As an ADMIN user in an OML notebook, specify a conda interpreter in a paragraph using `%conda`, then use the `create` command to create a conda environment named *sbenv* to install the seaborn package. Specify the Python version using the `python` parameter. Here, Python 3.12 is used for compatibility with OML4Py.

> **Note**
>
> When conda installs a package into an environment, it also installs any required dependencies. As a best practice, to avoid dependency conflicts, simultaneously install all the packages you need in a specific environment.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> To avoid version conflicts, do not install packages that are already included in OML4Py. For a complete list of pre-installed packages and their versions, see Python Libraries in OML4Py.

> **Note**
>
> Specify the current python version being used, when creating a conda environment for a 3rd-party package to avoid inconsistencies. To determine the python version being used, run the following command:
>
> ```
> import sys
> sys.version
> ```

Use the latest version of Python available in OML Notebooks; in this example, we are using Python 3.12.x.

```
%conda

create -n sbenv -c conda-forge --strict-channel-priority python=3.12.x
seaborn
```

Upload the environment to Object Storage

Upload the environment to the Object Storage associated with the Autonomous AI Database instance. Here you provide an environment description and a tag corresponding to an application name, OML4Py.

```
%conda

upload sbenv --description 'Conda environment with seaborn' -t application
"OML4PY"


Uploading conda environment sbenv
Upload successful for conda environment sbenv
```

The environment is now available for an OML user to download. The uploaded environment will persist in Object Storage until it is deleted. The application tag is required for use with embedded execution. For example, OML4Py embedded Python execution works with conda environments containing the OML4Py tag, and OML4R embedded R execution works with conda environments containing the OML4R tag.

There is one Object Storage bucket for each data center region. The conda environments are saved to a folder in Object Storage corresponding to the tenancy and database. The folder is managed by Autonomous AI Database and only available to users through OML Notebooks. There is an 8G maximum size for a single conda environment, and no size limit on Object Storage.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Logged in as a non-administrator user, specify the conda interpreter in a notebook paragraph using `%conda`. Get the list of environments saved in Object Storage using the `list-saved-envs` command.

```
%conda

list-saved-envs
```

Provide the environment name as an argument to the `-e` parameter and request a list of packages installed in the environment.

```
%conda

list-saved-envs -e sbenv --installed-packages
```

The output is similar to the following:


```
{
  "name": "sbenv",
  "size": "1.7 GiB",
  "description": "Conda environment with seaborn",
  "tags": {
    "application": "OML4PY"
  },
  "number_of_installed_packages": 78,
  "installed_packages": [
    "blas-1.0-mkl",
    "bottleneck-1.3.5-py39h7deecbd_0",
    "brotli-1.0.9-h5eee18b_7",
    "brotli-bin-1.0.9-h5eee18b_7",
    "ca-certificates-2022.07.19-h06a4308_0",
    "certifi-2022.9.14-py39h06a4308_0",
    "cycler-0.11.0-pyhd3eb1b0_0",
    "dbus-1.13.18-hb2f20db_0",
    "expat-2.4.4-h295c915_0",
    "fftw-3.3.9-h27cfd23_1",
    "fontconfig-2.13.1-h6c09931_0",
    "fonttools-4.25.0-pyhd3eb1b0_0",
    "freetype-2.11.0-h70c0345_0",
    "giflib-5.2.1-h7b6447c_0",
    "glib-2.69.1-h4ff587b_1",
    "gst-plugins-base-1.14.0-h8213a91_2",
    "gstreamer-1.14.0-h28cd5cc_2",
    "icu-58.2-he6710b0_3",
    "intel-openmp-2021.4.0-h06a4308_3561",
    "jpeg-9e-h7f8727e_0",
    "kiwisolver-1.4.2-py39h295c915_0",
    "lcms2-2.12-h3be6417_0",
    "ld_impl_linux-64-2.38-h1181459_1",
    "lerc-3.0-h295c915_0",
    "libbrotlicommon-1.0.9-h5eee18b_7",
    "libbrotlidec-1.0.9-h5eee18b_7",
    "libbrotlienc-1.0.9-h5eee18b_7",
    "libdeflate-1.8-h7f8727e_5",
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
    "libffi-3.3-he6710b0_2",
    "libgcc-ng-11.2.0-h1234567_1",
    "libgfortran-ng-11.2.0-h00389a5_1",
    "libgfortran5-11.2.0-h1234567_1",
    "libpng-1.6.37-hbc83047_0",
    "libstdcxx-ng-11.2.0-h1234567_1",
    "libtiff-4.4.0-hecacb30_0",
    "libuuid-1.0.3-h7f8727e_2",
    "libwebp-1.2.2-h55f646e_0",
    "libwebp-base-1.2.2-h7f8727e_0",
    "libxcb-1.15-h7f8727e_0",
    "libxml2-2.9.14-h74e7548_0",
    "lz4-c-1.9.3-h295c915_1",
    "matplotlib-3.5.2-py39h06a4308_0",
    "matplotlib-base-3.5.2-py39hf590b9c_0",
    "mkl-2021.4.0-h06a4308_640",
    "mkl-service-2.4.0-py39h7f8727e_0",
    "mkl_fft-1.3.1-py39hd3c417c_0",
    "mkl_random-1.2.2-py39h51133e4_0",
    "munkres-1.1.4-py_0",
    "ncurses-6.3-h5eee18b_3",
    "numexpr-2.8.3-py39h807cd23_0",
    "numpy-1.22.3-py39he7a7128_0",
    "numpy-base-1.22.3-py39hf524024_0",
    "openssl-1.1.1q-h7f8727e_0",
    "packaging-21.3-pyhd3eb1b0_0",
    "pandas-1.4.4-py39h6a678d5_0",
    "pcre-8.45-h295c915_0",
    "pillow-9.2.0-py39hace64e9_1",
    "pip-22.1.2-py39h06a4308_0",
    "pyparsing-3.0.9-py39h06a4308_0",
    "pyqt-5.9.2-py39h2531618_6",
    "python-3.9.0-hdb3f193_2",
    "python-dateutil-2.8.2-pyhd3eb1b0_0",
    "pytz-2022.1-py39h06a4308_0",
    "qt-5.9.7-h5867ecd_1",
    "readline-8.1.2-h7f8727e_1",
    "scipy-1.7.3-py39h6c91a56_2",
    "seaborn-0.11.2-pyhd3eb1b0_0",
    "setuptools-63.4.1-py39h06a4308_0",
    "sip-4.19.13-py39h295c915_0",
    "six-1.16.0-pyhd3eb1b0_1",
    "sqlite-3.39.2-h5082296_0",
    "tk-8.6.12-h1ccaba5_0",
    "tornado-6.2-py39h5eee18b_0",
    "tzdata-2022c-h04d1e81_0",
    "wheel-0.37.1-pyhd3eb1b0_0",
    "xz-5.2.5-h7f8727e_1",
    "zlib-1.2.12-h5eee18b_3",
    "zstd-1.5.2-ha4553b6_0"
  ]
}
```

Delete an environment saved in an Object Storage

Use the `delete` command to delete an environment saved in an Object Storage.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> Only user ADMIN can delete an environment saved in an Object Storage.

```
%conda

delete sbenv


Deleting conda environment sbenv
Deletion successful for conda environment sbenv
```

7.3 OML User Tasks for Downloading an Available Conda Environment

Once user ADMIN installs the environment in Object Storage in the Autonomous AI Database, as an OML user, you can download, activate, and use it in Python paragraphs in notebooks and with embedded execution.

List all environments persisted in Object Storage

Get the list of environments saved in Object Storage using the `list-saved-envs` command.

```
%conda

list-saved-envs
```

Get information on a named environment persisted in Object Storage

Provide the environment name as an argument to the `-e` parameter and request information on the environment.

```
%conda

list-saved-envs -e sbenv
```

The output is similar to the following:

```
{
  "name": "sbenv",
  "size": "1.2 GiB",
  "description": "Conda environment with seaborn",
  "tags": {
    "application": "OML4PY"
  },
  "number_of_installed_packages": 60
}
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Download and activate the environment

Use the `download` command to download an environment from Object Storage. To activate the downloaded environment, use the `activate` command.

> **Note**
>
> The paragraph that contains the download command must be the first paragraph in the notebook.

```
%conda

download sbenv
activate sbenv


Downloading conda environment sbenv
Download successful for conda environment sbenv
```

List the packages available in the environment

Get the list of all the packages in an active environment using the `list` command.

```
%conda

list
```

The output is similar to the following:

```
# packages in environment at /u01/.conda/envs/sbenv:
#
# Name                    Version                   Build  Channel
blas                      1.0                         mkl
bottleneck                1.3.5           py39h7deecbd_0
brotli                    1.0.9                h5eee18b_7
brotli-bin                1.0.9                h5eee18b_7
ca-certificates           2022.07.19           h06a4308_0
certifi                   2022.9.14       py39h06a4308_0
cycler                    0.11.0             pyhd3eb1b0_0
dbus                      1.13.18              hb2f20db_0
expat                     2.4.4                h295c915_0
fftw                      3.3.9                h27cfd23_1
fontconfig                2.13.1               h6c09931_0
fonttools                 4.25.0             pyhd3eb1b0_0
freetype                  2.11.0               h70c0345_0
giflib                    5.2.1                h7b6447c_0
glib                      2.69.1               h4ff587b_1
gst-plugins-base          1.14.0               h8213a91_2
gstreamer                 1.14.0               h28cd5cc_2
icu                       58.2                 he6710b0_3
intel-openmp              2021.4.0          h06a4308_3561
jpeg                      9e                   h7f8727e_0
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
kiwisolver                1.4.2           py39h295c915_0
lcms2                     2.12                h3be6417_0
ld_impl_linux-64          2.38                h1181459_1
lerc                      3.0                 h295c915_0
libbrotlicommon           1.0.9               h5eee18b_7
libbrotlidec              1.0.9               h5eee18b_7
libbrotlienc              1.0.9               h5eee18b_7
libdeflate                1.8                 h7f8727e_5
libffi                    3.3                 he6710b0_2
libgcc-ng                 11.2.0              h1234567_1
libgfortran-ng            11.2.0              h00389a5_1
libgfortran5              11.2.0              h1234567_1
libpng                    1.6.37              hbc83047_0
libstdcxx-ng              11.2.0              h1234567_1
libtiff                   4.4.0               hecacb30_0
libuuid                   1.0.3               h7f8727e_2
libwebp                   1.2.2               h55f646e_0
libwebp-base              1.2.2               h7f8727e_0
libxcb                    1.15                h7f8727e_0
libxml2                   2.9.14              h74e7548_0
lz4-c                     1.9.3               h295c915_1
matplotlib                3.5.2           py39h06a4308_0
matplotlib-base           3.5.2           py39hf590b9c_0
mkl                       2021.4.0          h06a4308_640
mkl-service               2.4.0           py39h7f8727e_0
mkl_fft                   1.3.1           py39hd3c417c_0
mkl_random                1.2.2           py39h51133e4_0
munkres                   1.1.4                     py_0
ncurses                   6.3                 h5eee18b_3
numexpr                   2.8.3           py39h807cd23_0
numpy                     1.22.3          py39he7a7128_0
numpy-base                1.22.3          py39hf524024_0
openssl                   1.1.1q              h7f8727e_0
packaging                 21.3              pyhd3eb1b0_0
pandas                    1.4.4           py39h6a678d5_0
pcre                      8.45                h295c915_0
pillow                    9.2.0           py39hace64e9_1
pip                       22.1.2          py39h06a4308_0
pyparsing                 3.0.9           py39h06a4308_0
pyqt                      5.9.2           py39h2531618_6
python                    3.9.0               hdb3f193_2
python-dateutil           2.8.2             pyhd3eb1b0_0
pytz                      2022.1          py39h06a4308_0
qt                        5.9.7               h5867ecd_1
readline                  8.1.2               h7f8727e_1
scipy                     1.7.3           py39h6c91a56_2
seaborn                   0.11.2            pyhd3eb1b0_0
setuptools                63.4.1          py39h06a4308_0
sip                       4.19.13         py39h295c915_0
six                       1.16.0            pyhd3eb1b0_1
sqlite                    3.39.2              h5082296_0
tk                        8.6.12              h1ccaba5_0
tornado                   6.2             py39h5eee18b_0
tzdata                    2022c               h04d1e81_0
wheel                     0.37.1            pyhd3eb1b0_0
xz                        5.2.5               h7f8727e_1
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
zlib                      1.2.12              h5eee18b_3
zstd                      1.5.2               ha4553b6_0
```

Example 7-1 Create a visualization using seaborn

The following example shows the use of the available packages in the installed and activated environment. It imports `pandas, seaborn, and matplotlib` packages and loads the `iris` dataset from the seaborn library as a pandas dataframe. The `pairplot` seaborn function plots the pair-wise relationship between all the variables of the dataset.

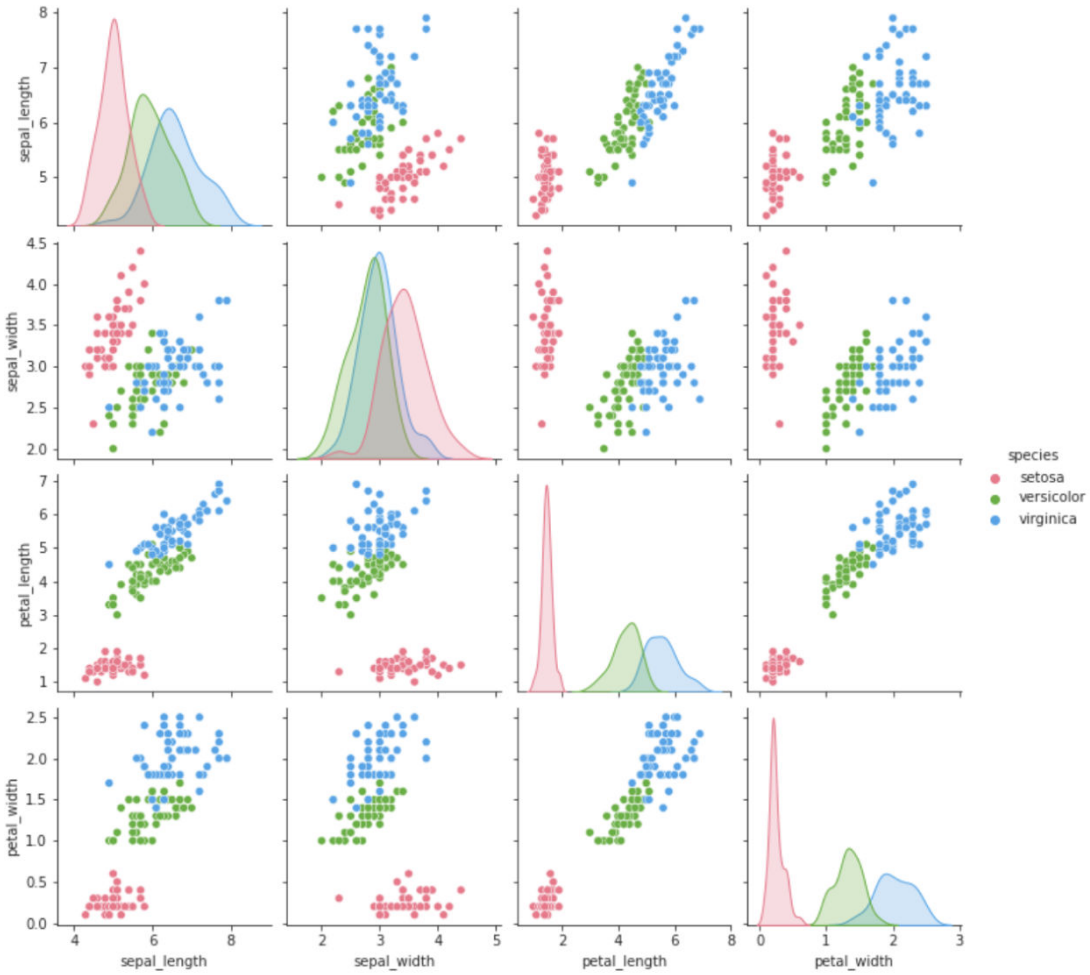
```
%python

import pandas as pd
import seaborn as sb
from matplotlib import pyplot as plt
df = sb.load_dataset('iris')
sb.set_style("ticks")
sb.pairplot(df,hue = 'species',diag_kind = "kde",kind = "scatter",palette =
"husl")
plt.show()
```

The output of the example is the following.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Figure 7-1 Iris pair plot



Example 7-2 Create a string representation of the function and save it to the OML4Py script repository

With OML4Py, functions are saved to the script repository using their string definition representation so they can be run in embedded Python execution. Create a function `sb_plot`, after verifying the function behaves as expected, provide it as a string (within triple quotes for formatting), and save it to the OML4Py script repository. Use the `oml.script.create` function to store a single user-defined Python function in the OML4Py script repository. The parameter `"sb_plot"` is a string that specifies the name of the user-defined function. The parameter `func=sb_plot` is the Python function to run.

```
%python

sb_plot = """def sb_plot():
    import pandas as pd
    import seaborn as sb
    from matplotlib import pyplot as plt
    df = sb.load_dataset('iris')
    sb.set_style("ticks")
    sb.pairplot(df,hue = 'species',diag_kind = "kde",kind = "scatter",palette
= "husl")
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
    plt.show()"""

oml.script.create("sb_plot", func=sb_plot)
```

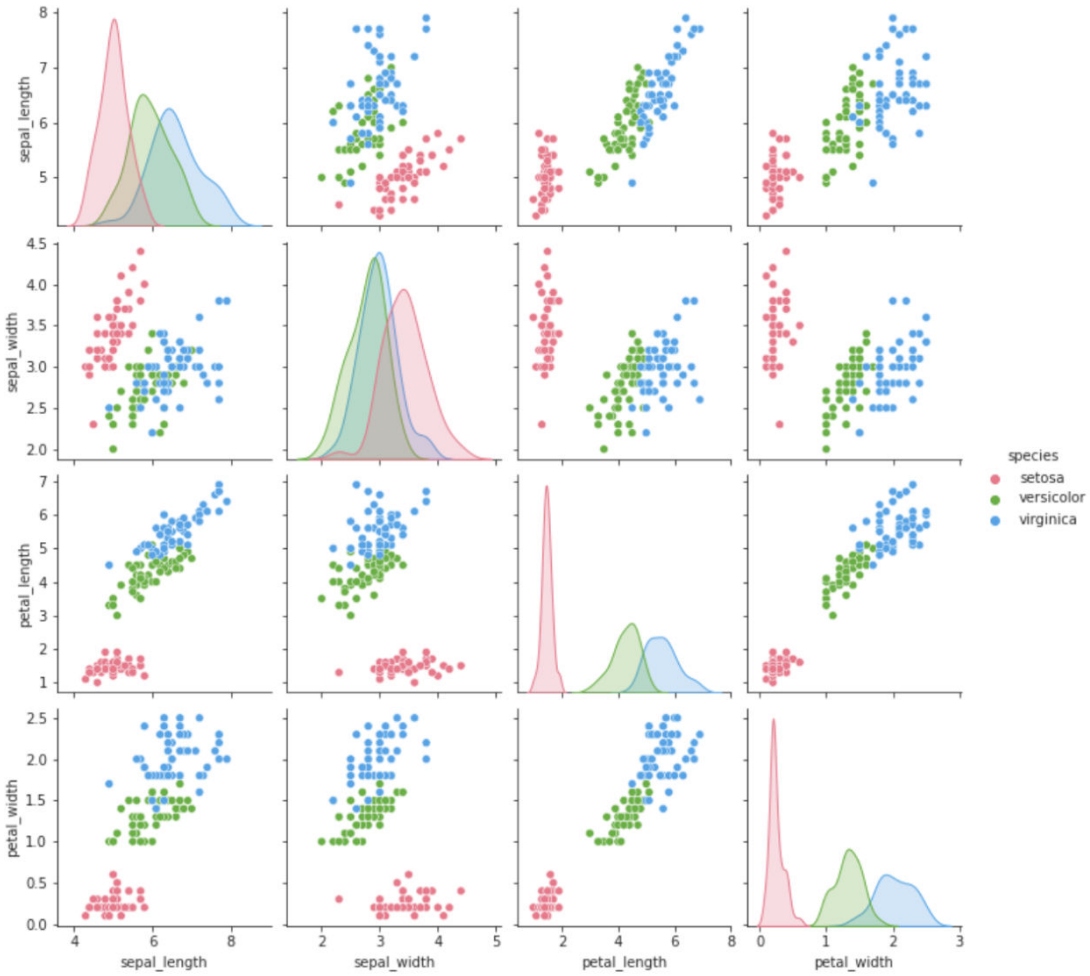
Use the Python API for embedded Python execution to run the user-defined Python function you saved in the script repository.

```
%python

oml.do_eval(func="sb_plot", graphics=True)
```

The output of the example is the following:

Figure 7-2 Iris pair plot



Deactivate the current environment

Use the `deactivate` command to deactivate an enviroment.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> At a given time, only one active environment is supported. So, a newly activated environment would replace an old environment. As a best practice, deactivate an environment before logging off.

```
%conda

deactivate


Conda environment deactivated
```

7.4 Using Conda Environments with Embedded Python Execution

This topic explains the usage of conda environments by running user-defined functions (UDFs) in SQL and REST APIs for embedded Python execution.

Running UDFs in the SQL and REST APIs for embedded Python execution

The conda environments can be used by OML4Py Python, SQL, and REST APIs. To use the SQL and REST API for embedded Python execution, the following information is needed.

1. The token URL from the OML service console in Autonomous AI Database. For more information on how to obtain the token URL and set the access token see Access and Authorization Procedures and Functions (Autonomous AI Database).
2. A script containing a user-defined Python function in the Oracle Machine Learning for Python (OML4Py) script repository. For information on creating a script and saving it to the script repository, see About Embedded Python Execution and the Script Repository.

> **Note**
>
> To use a conda environment when calling OML4Py script execution endpoints, specify the conda environment in the `env_name` field when using SQL, and the `envName` field when using REST.

Run the Python UDF using the SQL API for embedded Python execution - Asynchronous mode

Run a `SELECT` statement that calls the `pyqEval` function. The `PAR_LST` argument specifies the special control argument `oml_graphics_flag` to `true` so that the web server can capture images rendered in the invoked script, the `oml_async_flag` is set to true to submit the job asynchronously. In the `OUT_FMT` argument, the string 'PNG', specifies that the table returns the response in a table with fixed columns (including an image bytes column). The `SCR_NAME`

Copyright © 2024, 2026, Oracle and/or its affiliates.

parameter specifies the function `sb_plot` stored in the script repository. The `ENV_NAME` specifies the environment name `mysbenv` in which the script is called.

```
%script

set long 2000

SELECT * FROM table(pyqEval(
    par_lst => '{"oml_graphics_flag":true, "oml_async_flag":true}',
    out_fmt => 'PNG',
    scr_name => 'sb_plot',
    scr_owner=> NULL,
    env_name => 'mysbenv'));
```

The output is similar to the following:

```
NAME VALUE
--------------------------
             <OML URL>/oml/api/py-scripts/v1/jobs/b82947a7-ec3a-4ca6-
bf86-54b3f2b3a4b0
```

Get the job status

Poll the job status using the pyqJobStatus function. If the job is still running, the return value will note that the job is still running. When the job completes, a job ID and result location are returned.

```
%script

set long 1000
SELECT VALUE from pyqJobStatus(job_id => 'b82947a7-ec3a-4ca6-
bf86-54b3f2b3a4b0');
```

The output returns a job ID:

```
NAME VALUE
--------------------------
             <OML URL>/oml/api/py-scripts/v1/jobs/b82947a7-ec3a-4ca6-
bf86-54b3f2b3a4b0/result
```

Retrieve the result

```
%script

set long 500
SELECT NAME, ID, VALUE, dbms_lob.substr(image,100,1) image FROM
pyqJobResult(job_id => 'b82947a7-ec3a-4ca6-bf86-54b3f2b3a4b0',
out_fmt=>'PNG');
```

The output is similar to the following:

```
NAME ID VALUE IMAGE
--------------------------
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
          1
[{"0":0.0,"1":0.0,"2":0.2333333333,"accuracy":0.2333333333,"macro
avg":0.0777777778,"weighted avg":0.0544444444},
{"0":0.0,"1":0.0,"2":1.0,"accuracy":0.2333333333,"macro
avg":0.3333333333,"weighted avg":0.2333333333},
{"0":0.0,"1":0.0,"2":0.3783783784,"accuracy":0.2333333333,"macro
avg":0.1261261261,"weighted avg":0.0882882883},
{"0":11.0,"1":12.0,"2":7.0,"accuracy":0.2333333333,"macro avg":30.0,"weighted
avg":30.0}]
89504E470D0A1A0A0000000D494844520000046A000003E808060000008668185B000000397445
5874536F667477617265004D6174706C6F746C69622076657273696F6E332E362E322C20687474
70733A2F2F6D6174706C6F746C69622E6F72672F28E8
```

Run the Python UDF using the REST API for embedded Python execution

The following example runs the script named `sb_plot` in the OML4Py REST API for embedded Python execution. The environment name parameter `envName` is set to `mysbenv`. The `graphicsFlag` parameter is set to `true` to return the PNG image and the data from the function in JSON format.

```
$ curl -i -X POST --header "Authorization: Bearer ${token}" \
--header 'Content-Type: application/json' --header 'Accept: application/json'
\
-d '{"envName":"mysbenv", "graphicsFlag":true, "service":"LOW"}' \
"${omlserver}/oml/api/py-scripts/v1/do-eval/sb_plot"
```

The output is similar to the following:

```
NAME ID VALUE IMAGE
---------------------------
1 [{"0":0.0,"1":0.0,"2":0.2333333333,"accuracy":0.2333333333,"macro
avg":0.0777777778,"weighted avg":0.0544444444},
{"0":0.0,"1":0.0,"2":1.0,"accuracy":0.2333333333,"macro
avg":0.3333333333,"weighted avg":0.2333333333},
{"0":0.0,"1":0.0,"2":0.3783783784,"accuracy":0.2333333333,"macro
avg":0.1261261261,"weighted avg":0.0882882883},
{"0":11.0,"1":12.0,"2":7.0,"accuracy":0.2333333333,"macro avg":30.0,"weighted
avg":30.0}]
89504E470D0A1A0A0000000D494844520000046A000003E808060000008668185B000000397445
5874536F667477617265004D6174706C6F746C69622076657273696F6E332E362E322C20687474
70733A2F2F6D6174706C6F746C69622E6F72672F28E8
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

7.5 Conda Environment Best Practices

This section highlights the best practices for creating conda environments in OML4Py to prevent Python and third/fourth-party dependency conflicts.

Python Version

The current Python version being used, `python=3.13.x`, when creating a Conda environment for a 3rd-party package to avoid inconsistencies. To determine the latest Python version being used, run the following command:

```
import sys
sys.version
```

> **Note**
>
> To prevent dependency conflicts, you can create a new environment and install all required packages simultaneously using the `create` command. This allows the solver to resolve all dependencies at once.

Admin Task: Create and Upload a Conda Environment

Use the `create` command to create a Conda environment named `mypyenv` to install the seaborn package. Specify the Python version using the `python` parameter. Use the `upload` command to upload the `mypyenv` environment to the Object Storage associated with the Autonomous AI Database instance for an OML user to download and use it.

Use the latest version of Python available in OML Notebooks; in this example, we are using Python 3.13.5.

```
%conda

create -n mypyenv -c conda-forge --override-channels --strict-channel-
priority python=3.13.5 seaborn
upload mypyenv --overwrite -t application "OML4PY"
```

OML User: Use a Conda Enviroment

An OML user, you can download, activate, and use the environment in Python paragraphs in notebooks:

```
%conda

download mypyenv
activate mypyenv
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Import the `seaborn` library:

```
%python

import seaborn
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

8

Get Started with Oracle Machine Learning for Python

Learn how to use OML4Py to move data between Python sessions and the database, and save Python objects in the database.

These actions are described in the following topics.

Topics:

- Use OML4Py with Oracle Autonomous AI Database
  OML4Py is available through the Python interpreter in Oracle Machine Learning Notebooks in Oracle Autonomous AI Database.
- Use OML4Py with an On-Premises Oracle AI Database
  After the OML4Py server and client components have been installed on your on-premises Oracle AI Database server and you have installed the OML4Py client on your local system, you can connect your client Python session to the OML4Py server.
- Move Data Between the Database and a Python Session
  With OML4Py functions, you can interact with data in a database schema.
- Save Python Objects in the Database
  You can save Python and OML4Py objects in OML4Py datastores, which persist in the database.
- Import and Export Datastore
  Use the Datastore Import and Export feature in OML4Py to migrate Python and OML proxy objects—including datasets, models, and more—between Oracle AI Database instances.

8.1 Use OML4Py with Oracle Autonomous AI Database

OML4Py is available through the Python interpreter in Oracle Machine Learning Notebooks in Oracle Autonomous AI Database.

For more information, see Get Started with Notebooks for Data Analysis and Data Visualization in *Using Oracle Machine Learning Notebooks*.

8.2 Use OML4Py with an On-Premises Oracle AI Database

After the OML4Py server and client components have been installed on your on-premises Oracle AI Database server and you have installed the OML4Py client on your local system, you can connect your client Python session to the OML4Py server.

To connect an OML4Py client to an on-premises Oracle AI Database, you first import the `oml` module and then connect as described in the following topics.

- About Connecting to an On-Premises Oracle AI Database
  OML4Py client components connect a Python session to the OML4Py server components on an on-premises Oracle AI Database server.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- About Oracle Wallets
  An Oracle wallet is a secure software container that stores authentication and signing credentials for an Oracle AI Database.
- Connect to an Oracle AI Database
  Establish an OML4Py connection to an on-premises Oracle AI Database with `oml.connect`.

8.2.1 About Connecting to an On-Premises Oracle AI Database

OML4Py client components connect a Python session to the OML4Py server components on an on-premises Oracle AI Database server.

The connection makes the data in an on-premises Oracle AI Database schema available to the Python user. It also makes the processing power, memory, and storage capacities of the database server available to the Python session through the OML4Py client interface. To use that data and those capabilities, you must create a connection to the Oracle AI Database server.

To use the Automatic Machine Learning (AutoML) capabilities of OML4Py, the following must be true:

- A connection pool must be running on the server.
- You must explicitly use the `automl` argument in an `oml.connect` invocation to specify the running connection pool on the server.

> **Note**
>
> Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle AI Database by issuing the following SQL statement:
> `EXECUTE DBMS_CONNECTION_POOL.START_POOL();`
>
> Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:
>
> `EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();`

> **Note**
>
> Because an AutoML connection requires more database resources than an `oml.connect` connection without AutoML does, you should create an AutoML connection only if you are going to use the AutoML classes.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> - Only one type of connection can be active during a Python session: either a connection with AutoML enabled or one without it enabled. You can, however, terminate one type of connection and initiate the other type during the same Python session. Terminating either type of connection results in the automatic clean up of any temporary objects created in the session during that connection.
>
> If you want to save any objects that you created in one type of connection before changing to the other type, then save the objects in an OML4Py datastore before invoking `oml.connect` again. You can then reload the objects after reconnecting.
> - The `oml.connect` function uses the `cx_Oracle` Python package for database connectivity. In some cases, you might want to use the `cx_Oracle.connect` function of that package to connect to a database. That function has advantages such as the following:
> – Allows multiple connections to a multiple databases, which might be useful in an running Embedded Python Execution functions
> – Permits some SQL data manipulation language (DML) operations that are not available in an `oml.connect` connection
>
> For information on the `cx_Oracle.connect` function, see Connecting to Oracle AI Database in the `cx_Oracle` documentation.

OML4Py Connection Functions

The OML4Py functions related to database connections are the following.

Table 8-1 Connection Functions for OML4Py

Function	Description
`oml.connect`	Establishes an OML4Py connection to an Oracle AI Database.
`oml.disconnect`	Terminates the Oracle AI Database connection.
`oml.isconnected`	Indicates whether an active Oracle AI Database connection exists.
`oml.check_embed`	Indicates whether Embedded Python Execution is enabled in the connected Oracle AI Database.

8.2.2 About Oracle Wallets

An Oracle wallet is a secure software container that stores authentication and signing credentials for an Oracle AI Database.

You can create an OML4Py connection to an Oracle AI Database instance by specifying an Oracle wallet. For instructions on creating an Oracle wallet, see Managing the Secure External Password Store for Password Credentials in *Oracle AI Database Security Guide*.

The Oracle wallet must contain a credential that specifies a `tnsnames.ora` entry such as the following:

```
waltcon = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=myserv.example.com)))
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

To be able to use an Oracle wallet to create an OML4Py connection in which you can use Automatic Machine Learning (AutoML), the wallet must also have a credential that has a `tnsnames.ora` entry for a server connection pool such as the following:

```
waltcon_pool = (DESCRIPTION= (ADDRESS=(PROTOCOL=tcp)(HOST=myhost)
(PORT=1521))(CONNECT_DATA=(SID=mysid)(SERVER=pooled)))
```

> **Note**
>
> Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle AI Database by issuing the following SQL statement:
> `EXECUTE DBMS_CONNECTION_POOL.START_POOL();`
>
> Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:
>
> `EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();`

For examples of creating a connection using an Oracle wallet, see Example 8-6 and Example 8-7.

8.2.3 Connect to an Oracle AI Database

Establish an OML4Py connection to an on-premises Oracle AI Database with `oml.connect`.

The `oml.connect` function establishes a connection to the user's schema in an on-premises Oracle AI Database.

The syntax of the `oml.connect` function is the following.

```
oml.connect(user=None, password=None, host=None, port=None, sid=None,
service_name=None, dsn=None, encoding='UTF-8', nencoding='UTF-8', automl=None)
```

To create a basic connection to the database, you can specify arguments to the `oml.connect` function in the following mutually exclusive combinations:

- `user`, `password`, `dsn`
- `user`, `password`, `host`, `port`, `sid`
- `user`, `password`, `host`, `port`, `service_name`

The arguments specify the following values

Table 8-2 Pararmeters to oml.connect

Parameter	Description
`user`	A string specifying a username.
`password`	A string specifying the password for the user.
`host`	A string specifying the name of the host machine on which the OML4Py server is installed.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 8-2 (Cont.) Parameters to oml.connect

Parameter	Description
`port`	An `int` or a string specifying the Oracle AI Database port number on the host machine.
`sid`	A string specifying the system identifier (SID) of the Oracle AI Database.
`service_name`	A string specifying the service name of the Oracle AI Database.
`dsn`	A string specifying a data source name, which can be a TNS entry for the database or a TNS alias in an Oracle Wallet.
`encoding`	A string specifying the encoding to use for regular database strings.
`nencoding`	A string specifying the encoding to use for national character set database strings.
`automl`	A string or a boolean specifying whether to enable an Automatic Machine Learning (AutoML) connection, which uses the database-resident connection pool. If there is a connection pool running for a host, port, SID (or service name), then you can specify that host, port, SID (or service name) and `automl=True`. If the `dsn` argument is a data source name, then the `automl` argument must be a data source name for a running connection pool. If the `dsn` argument is a TNS alias, then the `automl` argument must be a TNS alias for a connection pool specified in an Oracle Wallet.

To use the AutoML capabilities of OML4Py, the following must be true:

- A connection pool must be running on the server.
- You must explicitly use the `automl` argument in an `oml.connect` invocation to specify the running connection pool on the server.

> **Note**
>
> Before you can create an AutoML connection, a database administrator must first activate the database-resident connection pool in your on-premises Oracle AI Database by issuing the following SQL statement:
> `EXECUTE DBMS_CONNECTION_POOL.START_POOL();`
>
> Once started, the connection pool remains in this state until a database administrator explicitly stops it by issuing the following command:
>
> `EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();`

Only one active OML4Py connection can exist at a time during a Python session. If you call `oml.connect` when an active connection already exists, then the `oml.disconnect` function is implicitly invoked, any temporary objects that you created in the previous session are discarded, and the new connection is established. Before attempting to connect, you can discover whether an active connection exists by using the `oml.isconnected` function.

You explicitly end a connection with the `oml.disconnect` function. If you do not invoke `oml.disconnect`, then the connection is automatically terminated when the Python session ends.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Examples

In the following examples, the values of the some of the arguments to the `oml.connect` function are string variables that are not declared in the example. To use any of the following examples, replace the username, password, port, and variable argument values with the values for your user and database.

Example 8-1 Connecting with a Host, Port, and SID

This example uses the `host`, `port`, and `sid` arguments. It also shows the use of the `oml.isconnected`, `oml.check_embed`, and `oml.disconnect` functions.

```
import oml

oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, sid='mysid')

# Verify that the connection exists.
oml.isconnected()

# Find out whether Embedded Python Execution is enabled in the
# database instance.
oml.check_embed()

# Disconnect from the database.
oml.disconnect()

# Verify that the connection has been terminated.
oml.isconnected()
```

Listing for This Example

```
>>> import oml
>>>
>>> oml.connect(user='oml_user', password='oml_user_password', host='myhost',
...             port=1521, sid='mysid')
>>>
>>> # Verify that the connection exists.
... oml.isconnected()
True
>>>
>>> # Find out whether Embedded Python Execution is enabled in the
... # database instance.
... oml.check_embed()
True
>>>
>>> # Disconnect from the database.
... oml.disconnect()
>>>
>>> # Verify that the connection has been terminated.
... oml.isconnected()
False
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 8-2 Connecting with Host, Port, and Service Name

This example uses the `host`, `port`, and `service_name` arguments.

```
import oml

oml.connect(user='oml_user', password='oml_user_password', host='myhost',
            port=1521, service_name='myservice')
```

Example 8-3 Connecting with a DSN Containing a SID

This example uses the `dsn` argument to specify a SID.

```
import oml

mydsn = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)(PORT=1521))\
          (CONNECT_DATA=(SID=mysid)))"
oml.connect(user='oml_user', password='oml_user_password', dsn=mydsn)
```

Example 8-4 Connecting with a DSN Containing a Service Name

This example uses the `dsn` argument to specify a service name.

```
import oml

myinst = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)\
          (PORT=1521))\
          (CONNECT_DATA=(SERVICE_NAME=myservice.example.com)))"
oml.connect(user='oml_user', password='oml_user_password', dsn=myinst)
```

Example 8-5 Creating a Connection with a DSN and with AutoML Enabled

This example creates an OML4Py connection with AutoML enabled. The example connects to a local database.

```
import oml

mydsn = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=myhost)\
          (PORT=1521))(CONNECT_DATA=(SID=mysid)))"

dsn_pool = "(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhost)\
             (PORT=1521))\
             (CONNECT_DATA=(SERVICE_NAME=myservice.example.com)\
             (SERVER=POOLED)))"

oml.connect(user='oml_user', password='oml_user_password',
            dsn=mydsn, automl=dsn_pool)

# Verify that the connection exists and that AutoML is enabled.
oml.isconnected(check_automl=True)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 8-6 Connecting with an Oracle Wallet

This example creates a connection using the `dsn` argument to specify an Oracle wallet. The `dsn` value, `waltcon` in the example, must refer to the alias in the database `tnsnames.ora` file that was used to create the appropriate credential in the wallet.

```
import oml

oml.connect(user='', password='', dsn='waltcon')
```

> **See Also**
>
> About Oracle Wallets

Example 8-7 Connecting with an Oracle Wallet with AutoML Enabled

This example connects using an Oracle wallet to establish a connection with AutoML enabled by using the `dsn` and `automl` arguments. The example then verifies that the connection has AutoML enabled. The `dsn` and `automl` values, `waltcon` and `waltcon_pool` in the example, must refer to aliases in the database `tnsnames.ora` file that were used to create the appropriate credentials in the wallet.

```
import oml

oml.connect(user='', password='', dsn='waltcon', automl='waltcon_pool')
oml.isconnected(check_automl=True)
```

8.3 Move Data Between the Database and a Python Session

With OML4Py functions, you can interact with data in a database schema.

In your Python session, you can move data to and from the database and create temporary or persistent database tables. The OML4Py functions that perform these actions are described in the following topics.

Topics:

- About Moving Data Between the Database and a Python Session
  Using the functions described in this topic, you can move data between the your local Python session and an Oracle AI Database schema.
- Push Local Python Data to the Database
  Use the `oml.push` function to push data from your local Python session to a temporary table in your Oracle AI Database schema.
- Pull Data from the Database to a Local Python Session
  Use the `pull` method of an `oml` proxy object to create a Python object in your local Python session.
- Create a Python Proxy Object for a Database Object
  Use the `oml.sync` function to create a Python object as a proxy for a database table, view, or SQL statement.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Create a Persistent Database Table from a Python Data Set
  Use the `oml.create` function to create a persistent table in your database schema from data in your Python session.

8.3.1 About Moving Data Between the Database and a Python Session

Using the functions described in this topic, you can move data between the your local Python session and an Oracle AI Database schema.

The following functions create proxy `oml` Python objects from database objects, create database tables from Python objects, list the objects in the workspace, and drop tables and views.

Function	Definition
`oml.create`	Creates a persistent database table from a Python data set.
`oml.cursor`	Returns a `cx_Oracle cursor` object for the current OML4Py database connection.
`oml.dir`	Returns the names of the `oml` objects in the workspace.
`oml.drop`	Drops a database table, view, or model.
`oml_object``.pull`	Creates a local Python object that contains a copy of the database data referenced by the `oml` object.
`oml.push`	Pushes data from the OML Notebooks Python session memory into a temporary table in the database.
`oml.sync`	Creates an `oml.DataFrame` proxy object in Python that represents a database table, view, or query.

With the `oml.push` function, you can create a temporary database table, and its corresponding proxy `oml.DataFrame` object, from a Python object in your local Python session. The temporary table is automatically deleted when the OML Notebook or OML4Py client connection to the database ends unless you have saved its proxy object to a datastore before disconnecting.

With the `pull` method of an `oml` object, you can create a local Python object that contains a copy of the database data represented by an `oml` proxy object.

The `oml.push` function implicitly coerces Python data types to `oml` data types and the `pull` method on `oml` objects coerces `oml` data types to Python data types.

With the `oml.create` function, you can create a persistent database table and a corresponding `oml.DataFrame` proxy object from a Python data set.

With the `oml.sync` function, you can synchronize the metadata of a database table or view with the `oml` object representing the database object.

With the `oml.cursor` function, you can create a `cx_Oracle cursor` object for the current database connection. You can user the `cursor` to run queries against the database, as shown in Example 8-13.

8.3.2 Push Local Python Data to the Database

Use the `oml.push` function to push data from your local Python session to a temporary table in your Oracle AI Database schema.

The `oml.push` function creates a temporary table in the user's database schema and inserts data into the table. It also creates and returns a corresponding proxy `oml.DataFrame` object

Copyright © 2024, 2026, Oracle and/or its affiliates.

that references the table in the Python session. The table exists as long as an `oml` object exists that references it, either in the Python session memory or in an OML4Py datastore.

The syntax of the `oml.push` function is the following:

```
oml.push(x, oranumber=True, dbtypes=None)
```

The `x` argument may be a `pandas.DataFrame` or a list of tuples of equal size that contain the data for the table. For a list of tuples, each tuple represents a row in the table and the column names are set to COL1, COL2, and so on.

The SQL data types of the columns are determined by the following:

- OML4Py determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, it uses all rows in determining the column type.

  If the values in a column are all `None`, or if a column has inconsistent data types that are not `None` in the sampled rows, then a default column type cannot be determined and a `ValueError` is raised unless a SQL type for the column is specified by the `dbtypes` argument.
- For numeric columns, the `oranumber` argument, which is a `bool`, determines the SQL data type. If `True` (the default), then the SQL data type is `NUMBER`. If `False`, then the data type is `BINARY_DOUBLE`.

  If the data in `x` contains `NaN` values, then you should set `oranumber` to `False`.
- For string columns, the default type is `VARCHAR2(4000)`.
- For bytes columns, the default type is `BLOB`.

With the `dbtypes` argument, you can specify the SQL data types for the table columns. The values of `dbtypes` may be either a `dict` that maps `str` to `str` values or a list of `str` values. For a `dict`, the keys are the names of the columns.

Example 8-8 Pushing Data to a Database Table

This example creates `pd_df`, a `pandas.core.frame.DataFrame` object with columns of various data types. It pushes `pd_df` to a temporary database table, which creates the `oml_df` object, which references the table. It then pulls the data from the `oml_df` object to the `df` object in local memory.

```
import oml
import pandas as pd

pd_df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
                      'string' : [None, None, 'a', 'a', 'a', 'b'],
                      'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e']})

# Push the data set to a database table with the specified dbtypes
# for each column.
oml_df = oml.push(pd_df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                                    'string':'CHAR(1)',
                                    'bytes':'RAW(1)'})

# Display the data type of oml_df.
type(oml_df)

# Pull the data from oml_df into local memory.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
df = oml_df.pull()

# Display the data type of df.
type(df)

# Create a list of tuples.
lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
       (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]

# Create an oml.DataFrame using the list.
oml_df2 = oml.push(lst, dbtypes = ['BINARY_DOUBLE','CHAR(1)','RAW(1)'])

type(oml_df2)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> pd_df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
...                       'string' : [None, None, 'a', 'a', 'a', 'b'],
...                       'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e']})
>>>
>>> # Push the data set to a database table with the specified dbtypes
... # for each column.
... oml_df = oml.push(pd_df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                     'string':'CHAR(1)',
...                                     'bytes':'RAW(1)'})
>>>
>>> # Display the data type of oml_df.
... type(oml_df)
<class 'oml.core.frame.DataFrame'>
>>>
>>> # Pull the data from oml_df into local memory.
... df = oml_df.pull()
>>>
>>> # Display the data type of df.
... type(df)
<class 'pandas.core.frame.DataFrame'>
>>>
>>> # Create a list of tuples.
... lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
...        (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]
>>>
>>> # Create an oml.DataFrame using the list.
... oml_df2 = oml.push(lst, dbtypes = ['BINARY_DOUBLE','CHAR(1)','RAW(1)'])
>>>
>>> type(oml_df2)
<class 'oml.core.frame.DataFrame'>
```

8.3.3 Pull Data from the Database to a Local Python Session

Use the `pull` method of an `oml` proxy object to create a Python object in your local Python session.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The `pull` method of an `oml` object returns a Python object of the same type. The object contains a copy of the database data referenced by the `oml` object. The Python object exists in-memory in the Python session in OML Notebooks or in your OML4Py client Python session..

> **Note**
>
> You can pull data to a local `pandas.DataFrame` only if the data can fit into the local Python session memory. Also, even if the data fits in memory but is still very large, you may not be able to perform many, or any, Python functions in the local Python session.

Example 8-9 Pulling Data into Local Memory

This example loads the iris data set and creates the IRIS database table and the `oml_iris` proxy object that references that table. It displays the type of the `oml_iris` object, then pulls the data from it to the `iris` object in local memory and displays its type.

```
import oml
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH','SEPAL_WIDTH',
'PETAL_LENGTH','PETAL_WIDTH'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:
'virginica'}[x], iris.target)), columns = ['SPECIES'])
iris_df = pd.concat([x, y], axis=1)

oml_iris = oml.create(iris_df, table = 'IRIS')

# Display the data type of oml_iris.
type(oml_iris)

# Pull the data from oml_iris into local memory.
iris = oml_iris.pull()

# Display the data type of iris.
type(iris)

# Drop the IRIS database table.
oml.drop('IRIS')
```

Listing for This Example

```
>>> import oml
>>> from sklearn.datasets import load_iris
>>> import pandas as pd
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
>>> iris = datasets.load_iris()

>>> iris = load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['SEPAL_LENGTH','SEPAL_WIDTH',
'PETAL_LENGTH','PETAL_WIDTH'])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor', 2:
'virginica'}[x], iris.target)), columns = ['SPECIES'])
>>> iris_df = pd.concat([x, y], axis=1)

>>> oml_iris = oml.create(iris_df, table = 'IRIS')

>>>
>>> # Display the data type of oml_iris.
... type(oml_iris)
<class 'oml.core.frame.DataFrame'>
>>>
>>> # Pull the data from oml_iris into local memory.
... iris = oml_iris.pull()
>>>
>>> # Display the data type of iris.
... type(iris)
<class 'pandas.core.frame.DataFrame'>
>>>
>>> # Drop the IRIS database table.
... oml.drop('IRIS')
```

8.3.4 Create a Python Proxy Object for a Database Object

Use the `oml.sync` function to create a Python object as a proxy for a database table, view, or SQL statement.

The `oml.sync` function returns an `oml.DataFrame` object or a dictionary of `oml.DataFrame` objects. The `oml.DataFrame` object returned by `oml.sync` is a proxy for the database object.

You can use the proxy `oml.DataFrame` object to select data from the table. When you run a Python function that selects data from the table, the function returns the current data from the database object. However, if some application has added a column to the table, or has otherwise changed the metadata of the database object, the `oml.DataFrame` proxy object does not reflect such a change until you again invoke `oml.sync` for the database object.

> **Tip**
>
> To conserve memory resources and save time, you should only create proxies for the tables that you want to use in your Python session.

You can use the `oml.dir` function to list the `oml.DataFrame` proxy objects in the environment for a schema.

The syntax of the `oml.sync` function is the following:

```
oml.sync(schema=None, regex_match=False, table=None, view=None, query=None)
```

The `schema` argument in `oml.sync` specifies the name of the schema where the database object exists. If `schema=None`, which is the default, then the current schema is used.

To create an `oml.DataFrame` object for a table, use the `table` parameter. To create one for a view, use the `view` parameter. To create one for a SQL `SELECT` statement, use the `query` parameter. You can only specify one of these parameters in an `oml.sync` invocation: the

Copyright © 2024, 2026, Oracle and/or its affiliates.

argument for one of the parameters must be a string and the argument for each of the other two parameters must be `None`.

Creating a proxy object for a query enables you to create an `oml.DataFrame` object without creating a view in the database. This can be useful when you do not have the `CREATE VIEW` system privilege for the current schema. You cannot use the `schema` parameter and the `query` parameter in the same `ore.sync` invocation.

With the `regex_match` argument, you can specify whether the value of the `table` or `view` argument is a regular expression. If `regex_match=True`, then `oml.sync` creates `oml.DataFrame` objects for each database object that matches the pattern. The matched tables or views are returned in a `dict` with the table or view names as keys.

Example 8-10 Creating a Python Object for a Database Table

This example creates an `oml.DataFrame` Python object as a proxy for a database table. For this example, the table COFFEE exists in the user's schema.

```
import oml

# Create the Python object oml_coffee as a proxy for the
# database table COFFEE.
oml_coffee = oml.sync(table = 'COFFEE')
type(oml_coffee)

# List the proxy objects in the schema.
oml.dir()

oml_coffee.head()
```

Listing for This Example

```
>>> import oml
>>>
>>> # Create the Python object oml_coffee as a proxy for the
... # database table COFFEE.
... oml_coffee = oml.sync(table = 'COFFEE')
>>> type(oml_coffee)
<class 'oml.core.frame.DataFrame'>
>>>
>>> # List the proxy objects in the schema.
... oml.dir()
['oml_coffee']
>>>
>>> oml_coffee.head()
   ID COFFEE WINDOW
0   1    esp      w
1   2    cap      d
2   3    cap      w
3   4    kon      w
4   5    ice      w
```

Example 8-11 Using the regex_match Argument

This example uses the `regex_match` argument in creating a `dict` object that contains `oml.DataFrame` proxy objects for tables whose names start with C. For this example, the

Copyright © 2024, 2026, Oracle and/or its affiliates.

COFFEE and COLOR tables exist in the user's schema and are the only tables whose names start with C.

```
# Create a dict of oml.DataFrame proxy objects for tables
# whose names start with 'C'.
oml_cdat = oml.sync(table="^C", regex_match=True)

oml_cdat.keys()
oml_cdat['COFFEE'].columns
oml_cdat['COLOR'].columns
```

Listing for This Example

```
>>> # Create a dict of oml.DataFrame proxy objects for tables
... # whose names start with 'C'.
... oml_cdat = oml.sync(table="^C", regex_match=True)
>>>
>>> oml_cdat.keys()
dict_keys(['COFFEE', 'COLOR']
>>> oml_cdat['COFFEE'].columns
['ID', 'COFFEE', 'WINDOW']
>>> oml_cdat['COLOR'].columns
['REGION', 'EYES', 'HAIR', 'COUNT']
```

Example 8-12 Synchronizing an Updated Table

This example uses `oml.sync` to create an `oml.DataFrame` for the database table COFFEE. For the example, the new column BREW has been added to the database table by some other database process after the first invocation of `oml.sync`. Invoking `oml.sync` again synchronizes the metadata of the `oml.DataFrame` with those of the table.

```
oml_coffee =  oml.sync(table = "COFFEE")
oml_coffee.columns

# After a new column has been inserted into the table.
oml_coffee =  oml.sync(table = "COFFEE")
oml_coffee.columns
```

Listing for This Example

```
>>> oml_coffee =  oml.sync(table = "COFFEE")
>>> oml_coffee.columns
['ID', 'COFFEE', 'WINDOW']
>>>
>>> # After a new column has been inserted into the table.
... oml_coffee =  oml.sync(table = "COFFEE")
>>> oml_coffee.columns
['ID', 'COFFEE', 'WINDOW', 'BREW']
```

8.3.5 Create a Persistent Database Table from a Python Data Set

Use the `oml.create` function to create a persistent table in your database schema from data in your Python session.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The `oml.create` function creates a table in the database schema and returns an `oml.DataFrame` object that is a proxy for the table. The proxy `oml.DataFrame` object has the same name as the table.

> **Note**
>
> When creating a table in Oracle Machine Learning for Python, if you use lowercase or mixed case for the name of the table, then you must use the same lowercase or mixed case name in double quotation marks when using the table in a SQL query or function. If, instead, you use an all uppercase name when creating the table, then the table name is case-insensitive: you can use uppercase, lowercase, or mixed case when using the table without using double quotation marks. The same is true for naming columns in a table.

You can delete the persistent table in a database schema with the `oml.drop` function.

> **Caution**
>
> Use the `oml.drop` function to delete a persistent database table. Use the `del` statement to remove an `oml.DataFrame` proxy object and its associated temporary table; `del` does not delete a persistent table.

The syntax of the `oml.create` function is the following:

```
oml.create(x, table, oranumber=True, dbtypes=None, append=False)
```

The `x` argument is a `pandas.DataFrame` or a list of tuples of equal size that contain the data for the table. For a list of tuples, each tuple represents a row in the table and the column names are set to COL1, COL2, and so on. The `table` argument is a string that specifies a name for the table.

The SQL data types of the columns are determined by the following:

- OML4Py determines default column types by looking at 20 random rows sampled from the table. For tables with less than 20 rows, it uses all rows in determining the column type.

  If the values in a column are all `None`, or if a column has inconsistent data types that are not `None` in the sampled rows, then a default column type cannot be determined and a `ValueError` is raised unless a SQL type for the column is specified by the `dbtypes` argument.
- For numeric columns, the `oranumber` argument, which is a `bool`, determines the SQL data type. If `True` (the default), then the SQL data type is NUMBER. If `False`, then the data type is BINARY DOUBLE.

  If the data in `x` contains NaN values, then you should set `oranumber` to `False`.
- For string columns, the default type is VARCHAR2(4000).
- For bytes columns, the default type is BLOB.

With the `dbtypes` parameter, you can specify the SQL data types for the table columns. The values of `dbtypes` may be either a `dict` that maps `str` to `str` values or a list of `str` values. For a `dict`, the keys are the names of the columns. The `dbtypes` parameter is ignored if the `append` argument is `True`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The `append` argument is a `bool` that specifies whether to append the `x` data to an existing table.

Example 8-13 Creating Database Tables from a Python Data Set

This example creates a `cursor` object for the database connection, creates a `pandas.core.frame.DataFrame` with columns of various data types, then creates a series of tables using different `oml.create` parameters and shows the SQL data types of the table columns.

```
import oml

# Create a cursor object for the current OML4Py database
# connection to run queries and get information from the database.
cr = oml.cursor()

import pandas as pd

df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, 2],
                   'string' : [None, None, 'a', 'a', 'a', 'b'],
                   'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e']})

# Get the order of the columns
df.columns

# Create a table with the default parameters.
oml_df1 = oml.create(df, table = 'tbl1')

# Show the default SQL data types of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl1'")
cr.fetchall()

# Create a table with oranumber set to False.
oml_df2 = oml.create(df, table = 'tbl2', oranumber = False)

# Show the SQL data typea of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl2'")
cr.fetchall()

# Create a table with dbtypes specified as a dict mapping column names
# to SQL data types.
oml_df3 = oml.create(df, table = 'tbl3',
                     dbtypes = {'numeric': 'BINARY_DOUBLE',
                                'bytes':'RAW(1)'})

# Show the SQL data types of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl3'")
cr.fetchall()

# Create a table with dbtypes specified as a list of SQL data types
# matching the order of the columns.
oml_df4 = oml.create(df, table = 'tbl4',
                     dbtypes = ['BINARY_DOUBLE','VARCHAR2(2000)','RAW(1)'])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Show the SQL data type of the columns.
_ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl4'")
cr.fetchall()

# Create a table from a list of tuples.
lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
       (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]
oml_df5 = oml.create(lst, table = 'tbl5',
                     dbtypes = ['BINARY_DOUBLE','CHAR(1)','RAW(1)'])

# Close the cursor
cr.close()

# Drop the tables.
oml.drop('tbl1')
oml.drop('tbl2')
oml.drop('tbl3')
oml.drop('tbl4')
oml.drop('tbl5')
```

Listing for This Example

```
>>> import oml
>>>
>>> # Create a cursor object for the current OML4Py database
... # connection to run queries and get information from the database.
... cr = oml.cursor()
>>>
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, 2],
...                    'string' : [None, None, 'a', 'a', 'a', 'b'],
...                    'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e']})
>>>
>>> # Get the order of the columns.
... df.columns
Index(['numeric', 'string', 'bytes'], dtype='object')
>>>
>>> # Create a table with the default parameters.
... oml_df1 = oml.create(df, table = 'tbl1')
>>>
>>> # Show the default SQL data types of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl1'")
>>> cr.fetchall()
[('NUMBER',), ('VARCHAR2',), ('BLOB',)]
>>>
>>> # Create a table with oranumber set to False.
... oml_df2 = oml.create(df, table = 'tbl2', oranumber = False)
>>>
>>> # Show the SQL data types of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl2'")
>>> cr.fetchall()
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
[('BINARY_DOUBLE',), ('VARCHAR2',), ('BLOB',)]
>>>
>>> # Create a table with dbtypes specified as a dict mapping column names
... # to SQL data types.
... oml_df3 = oml.create(df, table = 'tbl3',
...                      dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                 'bytes':'RAW(1)'})
>>>
>>> # Show the SQL data type of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl3'")
>>> cr.fetchall()
[('BINARY_DOUBLE',), ('VARCHAR2',), ('RAW',)]
>>>
>>> # Create a table with dbtypes specified as a list of SQL data types
... # matching the order of the columns.
... oml_df4 = oml.create(df, table = 'tbl4',
...                      dbtypes = ['BINARY_DOUBLE','VARCHAR2(2000)',
'RAW(1)'])
>>>
>>> # Show the SQL data type of the columns
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl4'")
>>> cr.fetchall()
[('BINARY_DOUBLE',), ('VARCHAR2',), ('RAW',)]
>>>
>>> # Create a table from a list of tuples.
... lst = [(1, None, b'a'), (1.4, None, b'b'), (-4, 'a', b'c'),
...        (3.145, 'a', b'c'), (5, 'a', b'd'), (None, 'b', b'e')]
>>> oml_df5 = oml.create(lst, table ='tbl5',
...                      dbtypes = ['BINARY_DOUBLE','CHAR(1)','RAW(1)'])
>>>
>>> # Show the SQL data type of the columns.
... _ = cr.execute("select data_type from all_tab_columns where table_name =
'tbl5'")
>>> cr.fetchall()
[('BINARY_DOUBLE',), ('CHAR',), ('RAW',)]
>>>
>>> # Close the cursor.
... cr.close()
>>>
>>> # Drop the tables
... oml.drop('tbl1')
>>> oml.drop('tbl2')
>>> oml.drop('tbl3')
>>> oml.drop('tbl4')
>>> oml.drop('tbl5')
```

8.4 Save Python Objects in the Database

You can save Python and OML4Py objects in OML4Py datastores, which persist in the database.

You can grant or revoke read privilege access to a datastore or its objects to one or more users. You can restore the saved objects in another Python session.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The following topics describe the OML4Py functions for creating and managing datastores.

Topics:

- About OML4Py Datastores
  In an OML4Py datastore, you can store Python objects, which you can then use in subsequent Python sessions; you can also make them available to other users or programs.
- Save Objects to a Datastore
  The `oml.ds.save` function saves one or more Python objects to a datastore.
- Load Saved Objects From a Datastore
  The `oml.ds.load` function loads one or more Python objects from a datastore into a Python session.
- Get Information About Datastores
  The `oml.ds.dir` function provides information about datastores.
- Get Information About Datastore Objects
  The `oml.ds.describe` function provides information about the objects in a datastore.
- Delete Datastore Objects
  The `oml.ds.delete` function deletes datastores or objects in a datastore.
- Manage Access to Stored Objects
  The `oml.grant` and `oml.revoke` functions grant or revoke the read privilege to datastores or to user-defined Python functions in the script repository.

8.4.1 About OML4Py Datastores

In an OML4Py datastore, you can store Python objects, which you can then use in subsequent Python sessions; you can also make them available to other users or programs.

Python objects, including OML4Py proxy objects, exist only for the duration of the current Python session unless you explicitly save them. You can save a Python object, including `oml` proxy objects, to a named datastore and then load that object in a later Python session, including an Embedded Python Execution session. OML4Py creates the datastore in the user's database schema. A datastore, and the objects it contains, persist in the database until you delete them.

You can grant or revoke read privilege permission to another user to a datastore that you created or to objects in a datastore.

OML4Py has Python functions for managing objects in a datastore. It also has PL/SQL procedures for granting or revoking the read privilege and database views for listing available datastores and their contents.

Using a datastore, you can do the following:

- Save OML4Py and other Python objects that you create in one Python session and load them in another Python session.
- Pass arguments to Python functions for use in Embedded Python Execution.
- Pass objects for use in Embedded Python Execution. You could, for example, use the `oml.glm` class to build an Oracle Machine Learning model and save it in a datastore. You could then use that model to score data in the database through Embedded Python Execution.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Python Interface for Datastores

The following table lists the Python functions for saving and managing objects in a datastore.

Function	Description
`oml.ds.delete`	Deletes one or more datastores or Python objects from a datastore.
`oml.ds.dir`	Lists the datastores available to the current user.
`oml.ds.load`	Loads Python objects from a datastore into the user's session.
`oml.ds.save`	Saves Python objects to a named datastore in the user's database schema.

The following table lists the Python functions for managing access to datastores and datastore objects.

Function	Description
`oml.grant`	Grants read privilege permission to another user to a datastore or a user-defined Python function in the script repository owned by the current user.
`oml.revoke`	Revokes the read privilege permission that was granted to another user to a datastore or a user-defined Python function in the script repository owned by the current user.

8.4.2 Save Objects to a Datastore

The `oml.ds.save` function saves one or more Python objects to a datastore.

OML4Py creates the datastore in the current user's schema.

The syntax of `oml.ds.save` is the following:

```
oml.ds.save(objs, name, description=' ', grantable=None,
                      overwrite=False, append=False, compression=False)
```

The `objs` argument is a `dict` that contains the name and object pairs to save to the datastore specified by the `name` argument.

With the `description` argument, you can provide some descriptive text that appears when you get information about the datastore. The `description` parameter has no effect when used with the `append` parameter.

With the `grantable` argument, you can specify whether the read privilege to the datastore may be granted to other users.

If you set the `overwrite` argument to `TRUE`, then you can replace an existing datastore with another datastore of the same name.

If you set the `append` argument to `TRUE`, then you can add objects to an existing datastore. The `overwrite` and `append` arguments are mutually exclusive.

If you set compression to `True`, then the serialized Python objects are compressed in the datastore.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 8-14 Saving Python Objects to a Datastore

This example demonstrates creating datastores.

```
import oml
from sklearn import datasets
from sklearn import linear_model
import pandas as pd

# Load three data sets and create oml.DataFrame objects for them.
wine = datasets.load_wine()
x = pd.DataFrame(wine.data, columns = wine.feature_names)
y = pd.DataFrame(wine.target, columns = ['Class'])

# Create the database table WINE.
oml_wine = oml.create(pd.concat([x, y], axis=1), table = 'WINE')
oml_wine.columns

diabetes = datasets.load_diabetes()
x = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
y = pd.DataFrame(diabetes.target, columns=['disease_progression'])
oml_diabetes = oml.create(pd.concat([x, y], axis=1),
                                    table = "DIABETES")
oml_diabetes.columns

boston = datasets.load_boston()
x = pd.DataFrame(boston.data, columns = boston.feature_names.tolist())
y = pd.DataFrame(boston.target, columns = ['Value'])
oml_boston = oml.create(pd.concat([x, y], axis=1), table = "BOSTON")
oml_boston.columns

# Save the wine Bunch object to the datastore directly,
# along with the oml.DataFrame proxy object for the BOSTON table.
oml.ds.save(objs={'wine':wine, 'oml_boston':oml_boston},
            name="ds_pydata", description = "python datasets")

# Save the oml_diabetes proxy object to an existing datastore.
oml.ds.save(objs={'oml_diabetes':oml_diabetes},
                  name="ds_pydata", append=True)

# Save the oml_wine proxy object to another datastore.
oml.ds.save(objs={'oml_wine':oml_wine},
            name="ds_wine_data", description = "wine dataset")

# Create regression models using sklearn and oml.
# The regr1 linear model is a native Python object.
regr1 = linear_model.LinearRegression()
regr1.fit(boston.data, boston.target)
# The regr2 GLM model is an oml object.
regr2 = oml.glm("regression")
X = oml_boston.drop('Value')
y = oml_boston['Value']
regr2 = regr2.fit(X, y)

# Save the native Python object and the oml proxy object to a datastore
# and allow the read privilege to be granted to them.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
oml.ds.save(objs={'regr1':regr1, 'regr2':regr2},
            name="ds_pymodel", grantable=True)

# Grant the read privilege to the datastore to every user.
oml.grant(name="ds_pymodel", typ="datastore", user=None)

# List the datastores to which the read privilege has been granted.
oml.ds.dir(dstype="grant")
```

Listing for This Example

```
>>> import oml
>>> from sklearn import datasets
>>> from sklearn import linear_model
>>> import pandas as pd
>>>
>>> # Load three data sets and create oml.DataFrame objects for them.
>>> wine = datasets.load_wine()
>>> x = pd.DataFrame(wine.data, columns = wine.feature_names)
>>> y = pd.DataFrame(wine.target, columns = ['Class'])
>>>
>>> # Create the database table WINE.
... oml_wine = oml.create(pd.concat([x, y], axis=1), table = 'WINE')
>>> oml_wine.columns
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium',
'total_phenols', 'flavanoids', 'nonflavanoid_phenols', 'proanthocyanins',
'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline', 'Class']
>>>
>>> diabetes = datasets.load_diabetes()
>>> x = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
>>> y = pd.DataFrame(diabetes.target, columns=['disease_progression'])
>>> oml_diabetes = oml.create(pd.concat([x, y], axis=1),
...                           table = "DIABETES")
>>> oml_diabetes.columns
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6',
'disease_progression']
>>>
>>> boston = datasets.load_boston()
>>> x = pd.DataFrame(boston.data, columns = boston.feature_names.tolist())
>>> y = pd.DataFrame(boston.target, columns = ['Value'])
>>> oml_boston = oml.create(pd.concat([x, y], axis=1), table = "BOSTON")
>>> oml_boston.columns
['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
'PTRATIO', 'B', 'LSTAT', 'Value']
>>>
>>> # Save the wine Bunch object to the datastore directly,
... # along with the oml.DataFrame proxy object for the BOSTON table.
... oml.ds.save(objs={'wine':wine, 'oml_boston':oml_boston},
...             name="ds_pydata", description = "python datasets")
>>>
>>> # Save the oml_diabetes proxy object to an existing
datastore.
... oml.ds.save(objs={'oml_diabetes':oml_diabetes},
...                   name="ds_pydata", append=True)
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> # Save the oml_wine proxy object to another datastore.
... oml.ds.save(objs={'oml_wine':oml_wine},
...             name="ds_wine_data", description = "wine dataset")
>>>
>>> # Create regression models using sklearn and oml.
... # The regr1 linear model is a native Python object.
... regr1 = linear_model.LinearRegression()
>>> regr1.fit(boston.data, boston.target)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
>>> # The regr2 GLM model is an oml proxy object.
... regr2 = oml.glm("regression")
>>> X = oml_boston.drop('Value')
>>> y = oml_boston['Value']
>>> regr2 = regr2.fit(X, y)
>>>
>>> # Save the native Python object and the oml proxy object to a datastore
... # and allow the read privilege to be granted to them.
... oml.ds.save(objs={'regr1':regr1, 'regr2':regr2},
...             name="ds_pymodel", grantable=True)
>>>
>>> # Grant the read privilege to the ds_pymodel datastore to every user.
... oml.grant(name="ds_pymodel", typ="datastore", user=None)
>>>
>>> # List the datastores to which the read privilege has been granted.
... oml.ds.dir(dstype="grant")
  datastore_name grantee
0     ds_pymodel  PUBLIC
```

8.4.3 Load Saved Objects From a Datastore

The `oml.ds.load` function loads one or more Python objects from a datastore into a Python session.

The syntax of `oml.ds.load` is the following:

```
oml.ds.load(name, objs=None, owner=None, to_globals=True)
```

The `name` argument specifies the datastore that contains the objects to load.

With the `objs` argument, you identify a specific object or a list of objects to load.

With the boolean `to_globals` parameter, you can specify whether the objects are loaded to a global workspace or to a dictionary object. If the argument to `to_globals` is `True`, then `oml.ds.load` function loads the objects into the global workspace. If the argument is `False`, then the function returns a `dict` object that contains pairs of object names and values.

The `oml.ds.load` function raises a `ValueError` if the `name` argument is an empty string or if the owner of the datastore is not the current user and the read privilege for the datastore has not been granted to the current user.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 8-15 Loading Objects from Datastores

This example loads objects from datastores. For the creation of the datastores used in this example, see Example 8-14.

```
import oml

# Load all Python objects from a datastore to the global workspace.
sorted(oml.ds.load(name="ds_pydata"))

# Load the named Python object from the datastore to the global workspace.
oml.ds.load(name="ds_pymodel", objs=["regr2"])

# Load the named Python object from the datastore to the user's workspace.
oml.ds.load(name="ds_pymodel", objs=["regr1"], to_globals=False)
```

Listing for This Example

```
>>> import oml
>>>
>>> # Load all Python objects from a datastore to the current workspace.
... sorted(oml.ds.load(name="ds_pydata"))
['oml_boston', 'oml_diabetes', 'wine']
>>>
>>> # Load the named Python object from the datastore to the global workspace.
... oml.ds.load(name="ds_pymodel", objs=["regr2"])
['regr2']
>>>
>>> # Load the named Python object from the datastore to the user's workspace.
... oml.ds.load(name="ds_pymodel", objs=["regr1"], to_globals=False)
{'regr1': LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
normalize=False)}
```

8.4.4 Get Information About Datastores

The `oml.ds.dir` function provides information about datastores.

The syntax of `oml.ds.dir` is the following:

```
oml.ds.dir(name=None, regex_match=False, dstype='user')
```

Use the `name` parameter to get information about a specific datastore.

Optionally, you can use the `regex_match` and `dstype` parameters to get information about datastores with certain characteristics. The valid arguments for `dstype` are the following:

Argument	Description
`all`	Lists all of the datastores to which the current user has the read privilege.
`grant`	Lists the datastores for which the current user has granted read privilege to other users.
`granted`	Lists the datastores for which other users have granted read privilege to the current user.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Argument	Description
`grantable`	Lists the datastores that the current user can grant the read privilege to.
`user`	Lists the datastores created by current user.
`private`	Lists the datastores that the current user cannot grant the read privileges to.

The `oml.ds.dir` function returns a `pandas.DataFrame` object that contains different columns depending on which `dstype` argument you use. The following table lists the arguments and the columns returned for the values supplied.

dstype Argument	Columns in the DataFrame Returned
`user`	DSNAME, which contains the datastore name
`private`	NOBJ, which contains the number of objects in the datastore
`grantable`	DSIZE, which contains the size in bytes of each object in the datastore
	CDATE, which contains the creation date of the datastore
	DESCRIPTION, which contains the optional description of the datastore
`all`	All of the columns returned by the `user`, `private`, and `grantable` values, plus this additional column:
`granted`	DSOWNER, which contains the owner of the datastore
`grant`	DSNAME, which contains the datastore name
	GRANTEE, which contains the name of the user to which the read privilege to the datastore has been granted by the current session user

Example 8-16 Getting Information About Datastores

This example demonstrates using different combinations of arguments to the `oml.ds.dir` function. It demonstrates using `oml.dir` to list some or all of the datastores. For the creation of the datastores used in this example, see Example 8-14.

```
import oml

# Show all saved datastores.
oml.ds.dir(dstype="all")[['owner', 'datastore_name', 'object_count']]

# Show datastores to which other users have been granted the read
# privilege.
oml.ds.dir(dstype="grant")

# Show datastores whose names match a pattern.
oml.ds.dir(name='pydata', regex_match=True)\
          [['datastore_name', 'object_count']]
```

Listing for This Example

```
>>> import oml
>>>
>>> # Show all saved datastores.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
... oml.ds.dir(dstype="all")[['owner', 'datastore_name', 'object_count']]
     owner  datastore_name  object_count
0  OML_USER      ds_pydata             3
1  OML_USER     ds_pymodel             2
2  OML_USER   ds_wine_data             1
>>>
>>> # Show datastores to which other users have been granted the read
>>> # privilege.
... oml.ds.dir(dstype="grant")
  datastore_name grantee
0     ds_pymodel  PUBLIC
>>>
>>> oml.ds.dir(name='pydata', regex_match=True)\
...           [['datastore_name', 'object_count']]
  datastore_name  object_count
0      ds_pydata             3
```

8.4.5 Get Information About Datastore Objects

The `oml.ds.describe` function provides information about the objects in a datastore.

The syntax of `oml.ds.describe` is the following:

```
oml.ds.describe(name, owner=None))
```

The `name` argument is a string that specifies the name of a datastore.

The `owner` argument is a string that specifies the owner of the datastore or `None` (the default). If you do not specify the owner, then the function returns information about the datastore if it is owned by the current user.

The `oml.ds.describe` function returns a `pandas.DataFrame` object, each row of which represents an object in the datastore. The columns of the `DataFrame` are the following:

- object_name, which specifies the name of the object
- class, which specifies the class of the object
- size, which specifies the size of the object in bytes
- length, which specifies the length of the object
- row_count, which specifies the rows of the object
- col_count, which specifies the columns of the object

This function raises a `ValueError` if the following occur:

- The current user is not the owner of the datastore and has not been granted read privilege for the datastore.
- The datastore does not exist.

Example 8-17 Getting Information About Datastore Objects

This example demonstrates the using the `oml.ds.describe` function. For the creation of the datastore used in this example, see Example 8-14.

```
import oml
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Describe the contents of the ds_pydata datastore.
oml.ds.describe(name='ds_pydata')
oml.ds.describe(name="ds_pydata")[['object_name', 'class']]
```

Listing for This Example

```
>>> import oml
>>>
>>> # Describe the contents of the ds_pydata datastore.
... oml.ds.describe(name='ds_pydata')
    object_name          class   size  length  row_count  col_count
0    oml_boston  oml.DataFrame   1073     506        506         14
1  oml_diabetes  oml.DataFrame    964     442        442         11
2          wine          Bunch  24177       5          1          5
>>> oml.ds.describe(name="ds_pydata")[['object_name', 'class']]
    object_name          class
0    oml_boston  oml.DataFrame
1  oml_diabetes  oml.DataFrame
2          wine          Bunch
```

8.4.6 Delete Datastore Objects

The `oml.ds.delete` function deletes datastores or objects in a datastore.

Use the `oml.ds.delete` function to delete one or more datastores in your database schema or to delete objects in a datastore.

The syntax of `oml.ds.delete` is the following:

```
oml.ds.delete(name, objs=None, regex_match=False)
```

The argument to the `name` parameter may be one of the following:

- A string that specifies the name of the datastore to modify or delete, or a regular expression that matches the datastores to delete.
- A list of `str` objects that name the datastores from which to delete objects.

The `objs` parameter specifies the objects to delete from a datastore. The argument to the `objs` parameter may be one of the following:

- A string that specifies the object to delete from one or more datastores, or a regular expression that matches the objects to delete.
- `None` (the default), which deletes the entire datastore or datastores.

The `regex_match` parameter is a `bool` that indicates whether the `name` or `objs` arguments are regular expressions. The default value is `False`. The `regex_match` parameter operates as follows:

- If `regex_match=False` and if `name` is not `None`, and:
  - If `objs=None`, then `oml.ds.delete` deletes the datastore or datastores specified in the `name` argument.
  - If you specify one or more datastores with the `name` argument and one or more datastore objects with the `objs` argument, then `oml.ds.delete` deletes the specified Python objects from the datastores.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- If `regex_match=True` and:
  - If `objs=None`, then `oml.ds.delete` deletes the datastores you specified in the `name` argument.
  - If the `name` argument is a string and you specify one or more datastore objects with the `objs` argument, then `oml.ds.delete` deletes from the datastore the objects whose names match the regular expression specified in the `objs` argument.
  - If the `name` argument is a list of `str` objects, then the `objs` argument must be a list of `str` objects of the same length as `name`, and `oml.ds.delete` deletes from the datastores the objects whose names match the regular expressions specified in `objs`.

This function raises an error if the following occur:

- A specified datastore does not exist.
- Argument `regex_match` is `False` and argument `name` is a list of `str` objects larger than 1 and argument `objs` is not `None`.
- Argument `regex_match` is `True` and arguments `name` and `objs` are lists that are not the same length.

Example 8-18 Deleting Datastore Objects

This example demonstrates the using the `oml.ds.delete` function. For the creation of the datastores used in this example, see Example 8-14.

```
import oml

# Show the existing datastores.
oml.ds.dir()

# Show the Python objects in the ds_pydata datastore.
oml.ds.describe(name='ds_pydata')

# Delete some objects from the datastore.
oml.ds.delete(name="ds_pydata", objs=["wine", "oml_boston"])

# Delete a datastore.
oml.ds.delete(name="ds_pydata")

# Delete all datastores whose names match a pattern.
oml.ds.delete(name="_pymodel", regex_match=True)

# Show the existing datastores again.
oml.ds.dir()
```

Listing for This Example

```
>>> import oml
>>>
>>> # Show the existing datastores.
... oml.ds.dir()
  datastore_name  object_count  size                date      description
0      ds_pydata             3  26214 2019-05-18 21:04:06  python datasets
1     ds_pymodel             2   6370 2019-05-18 21:08:18             None
2   ds_wine_data             1   1410 2019-05-18 21:06:53     wine dataset
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> # Show the Python objects in the ds_pydata datastore.
... oml.ds.describe(name='ds_pydata')
    object_name          class  size  length  row_count  col_count
0    oml_boston  oml.DataFrame  1073     506        506         14
1  oml_diabetes  oml.DataFrame   964     442        442         11
2          wine          Bunch  24177       5          1          5
>>>
>>> # Delete some objects from a datastore.
... oml.ds.delete(name="ds_pydata", objs=["wine", "oml_boston"])
{'wine', 'oml_boston'}
>>>
>>> # Delete a datastore.
... oml.ds.delete(name="ds_pydata")
'ds_pydata'
>>>
>>> # Delete all datastores whose names match a pattern.
... oml.ds.delete(name="_pymodel", regex_match=True)
{'ds_pymodel'}
>>>
>>> # Show the existing datastores again.
... oml.ds.dir()
  datastore_name  object_count  size                date   description
0   ds_wine_data             1  1410 2019-05-18 21:06:53  wine dataset
```

8.4.7 Manage Access to Stored Objects

The `oml.grant` and `oml.revoke` functions grant or revoke the read privilege to datastores or to user-defined Python functions in the script repository.

The `oml.grant` function grants the read privilege to another user to a datastore or to a user-defined Python function in the OML4Py script repository. The `oml.revoke` function revokes that privilege.

The syntax of these functions is the following:

```
oml.grant(name, typ='datastore', user=None)
oml.revoke(name, typ='datastore', user=None)
```

The `name` argument is a string that specifies the name of the user-defined Python function in the script repository or the name of a datastore.

The `typ` parameter must be specified. The argument is a string that is either `'datastore'` or `'pyqscript'`.

The `user` argument is a string that specifies the user to whom read privilege to the named datastore or user-defined Python function is granted or from whom it is revoked, or `None` (the default). If you specify `None`, then the read privilege is granted to or revoked from all users.

Example 8-19 Granting and Revoking Access to Datastores

This example displays the datastores to which the read privilege has been granted to all users. It revokes read privilege from the `ds_pymodel` datastore and displays the datastores with public read privilege again. It next grants the read privilege to the user SH and finally displays once

Copyright © 2024, 2026, Oracle and/or its affiliates.

more the datastores to which read privilege has been granted. For the creation of the datastores used in this example, see Example 8-14.

```
import oml

# Show datastores to which other users have been granted read privilege.
oml.ds.dir(dstype="grant")

# Revoke the read privilege from every user.
oml.revoke(name="ds_pymodel", typ="datastore", user=None)

# Again show datastores to which read privilege has been granted.
oml.ds.dir(dstype="grant")

# Grant the read privilege to the user SH.
oml.grant(name="ds_pymodel", typ="datastore", user="SH")

oml.ds.dir(dstype="grant")
```

Listing for This Example

```
>>> import oml
>>>
>>> # Show datastores to which other users have been granted read privilege.
... oml.ds.dir(dstype="grant")
  datastore_name grantee
0     ds_pymodel  PUBLIC
>>>
>>> # Revoke the read privilege from every user.
... oml.revoke(name="ds_pymodel", typ="datastore", user=None)
>>>
>>> # Again show datastores to which read privilege has been granted to other
users.
... oml.ds.dir(dstype="grant")
Empty DataFrame
Columns: [datastore_name, grantee]
Index: []
>>>
>>> # Grant the read privilege to the user SH.
... oml.grant(name="ds_pymodel", typ="datastore", user="SH")
>>>
>>> oml.ds.dir(dstype="grant")
  datastore_name grantee
0     ds_pymodel      SH
```

Example 8-20 Granting and Revoking Access to User-Defined Python Functions

This example grants the read privilege to the MYLM user-defined Python function to the user SH and then revokes that privilege. For the creation of the user-defined Python functions used in this example, see Example 14-11.

```
# List the user-defined Python functions available only to the current user.
oml.script.dir(sctype='user')

# Grant the read privilege to the MYLM user-defined Python function to the
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
user SH.
oml.grant(name="MYLM", typ="pyqscript", user="SH")

# List the user-defined Python functions to which read privilege has been
granted.
oml.script.dir(sctype="grant")

# Revoke the read privilege to the MYLM user-defined Python function from the
user SH.
oml.revoke(name="MYLM", typ="pyqscript", user="SH")

# List the granted user-defined Python functions again to see if the
revocation was successful.
oml.script.dir(sctype="grant")
```

Listing for This Example

```
>>> # List the user-defined Python functions available only to the current
user.
oml.script.dir(sctype='user')
   name                                                script
0  MYLM  def build_lm1(dat):\n  from sklearn import lin...
>>>
>>># Grant the read privilege to the MYLM user-defined Python function to the
user SH.
...oml.grant(name="MYLM", typ="pyqscript", user="SH")
>>>
>>> # List the user-defined Python functions to which read privilege has been
granted.
... oml.script.dir(sctype="grant")
   name grantee
0  MYLM      SH
>>>
>>> # Revoke the read privilege to the MYLM user-defined Python function from
the user SH.
... oml.revoke(name="MYLM", typ="pyqscript", user="SH")
>>>
>>> # List the granted user-defined Python functions again to see if the
revocation was successful.
... oml.script.dir(sctype="grant")
Empty DataFrame
Columns: [name, grantee]
Index: []
```

8.5 Import and Export Datastore

Use the Datastore Import and Export feature in OML4Py to migrate Python and OML proxy objects—including datasets, models, and more—between Oracle AI Database instances.

The OML4Py Datastore stores Python objects and OML4Py proxy objects directly in the database. Proxy objects can reference database entities such as tables or models. Interact with this functionality using dedicated OML4Py APIs, as described in these topics:

Copyright © 2024, 2026, Oracle and/or its affiliates.

Topics:

- Export Datastore
  Use the OML4Py export datastore feature to save Python or OML proxy objects from your current database instance into a zip archive.
- Import Datastore
  Use the OML4Py import datastore feature to restore saved Python or OML proxy objects into your current database instance.

8.5.1 Export Datastore

Use the OML4Py export datastore feature to save Python or OML proxy objects from your current database instance into a zip archive.

Use the `oml.export_ds` API to export a datastore from your current Oracle AI Database instance to a zip archive. This allows you to migrate Python and OML proxy objects—including datasets, models, and other analytical assets—from your instance, so you can later import them into another instance as needed.

Use the `oml.export_ds` function to save these objects to a portable zip archive.

```
oml.export_ds(
    file_name,
    name=None,
    regex_match=False,
    overwrite=False,
    database_directory="DATA_PUMP_DIR"
)
```

Parameters:

- **file_name** (*str*): Specifies the name of the destination zip file for the exported datastores.
- **name** (*str*, *list of str*, or *None*): Specifies the name(s) of the datastores to export, or a regular expression to match datastore names. If `name` is `None` or an empty list, all datastores are exported. Only database objects owned by the current user in the named datastores are included.
- **regex_match** (*bool*, default=False): If set to `True`, treats the `name` argument as a regular expression for matching datastore names.
- **overwrite** (*bool*, default=False): If True, overwrites the zip archive if it already exists.
- **database_directory** (*str*, default=`'DATA_PUMP_DIR'`): Specifies the name of an existing database directory where you have read and write privileges, used to export the datastore contents to a zip archive.

Example 8-21 Export a Datastore

The following example shows how to save Python objects into OML4Py datastores and then export them to a zip archive.

```
import oml
from sklearn import datasets

# Save Python objects into datastores
iris = datasets.load_iris()
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
diabetes = datasets.load_diabetes()
oml.ds.save(objs={'iris': iris}, name="ds_pyobjs1", description="python
datasets")
oml.ds.save(objs={'diabetes': diabetes}, name="ds_pyobjs2",
description="python datasets")

# Export datastores to a zip file
oml.export_ds(file_name="ds1.zip", name='pyobjs', regex_match=True,
overwrite=True)

# Optionally delete original datastores
oml.ds.delete(name=["ds_pyobjs1", "ds_pyobjs2"])
```

8.5.2 Import Datastore

Use the OML4Py import datastore feature to restore saved Python or OML proxy objects into your current database instance.

After exporting a datastore from one Oracle AI Database instance, you can use the `oml.import_ds` API to import it into another instance. This process lets you migrate Python and OML proxy objects—including datasets, models, and other analytical assets—and make them available in your new instance.

Use the `oml.import_ds` function to restore these objects from a zip archive into your current database instance.

```
oml.import_ds(
    file_name,
    grantable=None,
    overwrite=False,
    append=False,
    database_directory="DATA_PUMP_DIR"
)
```

- **file_name** (*str*): Specifies the name and location of the zip file containing the datastores to import.
- **grantable** (*bool*, optional): If set to `True`, allows the imported datastores to have their read privilege granted to other users. This is ignored if `overwrite` or `append` is used.
- **overwrite** (*bool*, optional): If set to `True`, overwrites any existing datastores with the same name.
- **append** (*bool*, optional): If set to `True`, appends the imported Python objects to existing datastores with the same name.
- **database_directory** (*str*, optional, default=`'DATA_PUMP_DIR'`): Specify an existing database directory where you have read and write privileges to transfer the datastore from the zip file into the database.

Example 8-22 Import a Datastore

Import a datastore,ds1.zip, into your current database environment using the OML4Py `oml.import_ds` API. Specify the file name of the datastore archive to restore the Python or

Copyright © 2024, 2026, Oracle and/or its affiliates.

OML proxy objects it contains. After the import, you can view the available datastores and load the objects for immediate use.

```
# Import datastores from a zip file located in the database directory
oml.import_ds(file_name="ds1.zip")

# List available datastores and see object counts
print(oml.ds.dir()[['datastore_name', 'object_count']])

# Load the imported objects from a specific datastore
loaded_objects = oml.ds.load(name="ds_pyobjs1")
print(sorted(loaded_objects))
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

9

Prepare and Explore Data

Use OML4Py methods to prepare data for analysis and to perform exploratory analysis of the data.

Methods of the OML4Py data type classes make it easier for you to prepare very large enterprise database-resident data for modeling. These methods are described in the following topics.

Topics:

- Prepare Data
  Using methods of OML4Py data type classes, you can prepare data for analysis in the database, as described in the following topics.
- Explore Data
  OML4Py provides methods that enable you to perform exploratory data analysis and common statistical operations.
- Render Graphics
  OML4Py provides functions for rendering graphical displays of data.

9.1 Prepare Data

Using methods of OML4Py data type classes, you can prepare data for analysis in the database, as described in the following topics.

- About Preparing Data in the Database
  OML4Py data type classes have methods that enable you to use Python to prepare database data for analysis.
- Select Data
  A typical step in preparing data for analysis is selecting or filtering values of interest from a larger data set.
- Combine Data
  You can join data from `oml.DataFrame` objects that represent database tables by using the `append`, `concat`, and `merge` methods.
- Clean Data
  In preparing data for analysis, a typical step is to transform data by dropping some values.
- Split Data
  Sample and randomly partition data with the `split` and `KFold` methods.

9.1.1 About Preparing Data in the Database

OML4Py data type classes have methods that enable you to use Python to prepare database data for analysis.

You can perform data preparation operations on large quantities of data in the database and then continue operating on that data in-database or pull a subset of the results to your local Python session where, for example, you can use third-party Python packages to perform other operations.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The following table lists methods with which you can perform common data preparation tasks and indicates whether the OML4Py data type class supports the method.

Table 9-1 Methods Supported by Data Types

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Datatime	oml.Timezone	oml.Timedelta	oml.Integer
`append`	Appends another `oml` data object of the same class to an `oml` object.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`ceil`	Computes the ceiling of each element in an `oml.Float` series data object.	✗	✗	✓	✗	✗	✗	✗	✗	✗
`concat`	Combines an `oml` data object column-wise with one or more other data objects.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`count_pattern`	Counts the number of occurrences of a pattern in each string.	✗	✗	✗	✓	✗	✗	✗	✗	✗
`create_view`	Creates an Oracle AI Database view for the data represented by the OML4Py data object.	✗	✗	✗	✗	✓	✓	✓	✓	✓
`dot`	Calculates the inner product of the current `oml.Float` object with another `oml.Float`, or does matrix multiplication with an `oml.DataFrame`.	✗	✗	✓	✗	✗	✗	✗	✗	✓
`drop`	Drops specified columns in an `oml.DataFrame`.	✗	✗	✗	✗	✓	✗	✗	✗	✗
`drop_duplicates`	Removes duplicated elements from an `oml` series data object or duplicated rows from an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`dropna`	Removes missing elements from an `oml` series data object, or rows containing missing values from an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 9-1 (Cont.) Methods Supported by Data Types

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Datatime	oml.Timezone	oml.Timedelta	oml.Integer
`exp`	Computes element-wise e to the power of values in an `oml.Float` series data object.	⊗	⊗	✓	⊗	⊗	⊗	⊗	⊗	✓
`find`	Finds the lowest index in each string in which a substring is found that is greater than or equal to a start index.	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗	⊗
`floor`	Computes the floor of each element in an `oml.Float` series data object.	⊗	⊗	✓	⊗	⊗	⊗	⊗	⊗	⊗
`head`	Returns the first *n* elements of an `oml` series data object or the first *n* rows of an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`KFold`	Splits the `oml` data object randomly into *k* consecutive folds.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`len`	Computes the length of each string in an `oml.Bytes` or `oml.String` series data object.	⊗	✓	⊗	✓	⊗	⊗	⊗	⊗	⊗
`log`	Calculates an element-wise logarithm, to the given base, of values in the `oml.Float` series data object.	⊗	⊗	✓	⊗	⊗	⊗	⊗	⊗	✓
`materialize`	Pushes the contents represented by an OML4Py proxy object (a view, a table, and so on) into a table in Oracle AI Database.	⊗	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗
`merge`	Joins another `oml.DataFrame` to an `oml.DataFrame`.	⊗	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗
`replace`	Replaces an existing value with another value.	⊗	⊗	✓	✓	✓	✓	⊗	⊗	✓

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 9-1 (Cont.) Methods Supported by Data Types

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Datatime	oml.Timezone	oml.Timedelta	oml.Integer
`rename`	Renames columns of an `oml.DataFrame`.	⊗	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗
`round`	Rounds `oml.Float` values to the specified decimal place.	⊗	⊗	✓	⊗	⊗	⊗	⊗	⊗	⊗
`select_types`	Returns the subset of columns that are included or excluded based on their `oml` data type.	⊗	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗
`split`	Splits an `oml` data object randomly into multiple sets.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`sqrt`	Computes the square root of each element in an `oml.Float` series data object.	⊗	⊗	✓	⊗	⊗	⊗	⊗	⊗	✓
`tail`	Returns the last *n* elements of an `oml` series data object or the last *n* rows of an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓

9.1.2 Select Data

A typical step in preparing data for analysis is selecting or filtering values of interest from a larger data set.

The examples in this section demonstrate selecting data from an `oml.DataFrame` object by rows, by columns, and by value.

The examples use the `oml_iris` object created by the following code, which imports the `sklearn.datasets` package and loads the `iris` data set. It creates the `x` and `y` variables, and then creates the persistent database table IRIS and the `oml.DataFrame` object `oml.iris` as a proxy for the table.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns = ['Sepal_Length','Sepal_Width',
                                       'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
                                     2:'virginica'}[x], iris.target)),
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

The examples are in the following topics:

- Select the First or Last Number of Rows
- Select Data by Column
- Select Data by Value

Select the First or Last Number of Rows

The `head` and `tail` methods return the first or last number of elements.

The default number of rows selected is 5.

Example 9-1 Selecting the First and Last Number of Rows

This example selects rows from the `oml.DataFrame` object `oml_iris`. It displays the first five rows and ten rows of `oml_iris` and then the last five and ten rows.

```
# Display the first 5 rows.
oml_iris.head()

# Display the first 10 rows.
oml_iris.head(10)

# Display the last 5 rows.
oml_iris.tail()

# Display the last 10 rows.
oml_iris.tail(10)
```

Listing for This Example

```
>>> # Display the first 5 rows.
... oml_iris.head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
>>>
>>> # Display the first 10 rows.
... oml_iris.head(10)
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
5           5.4          3.9           1.7          0.4  setosa
6           4.6          3.4           1.4          0.3  setosa
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
7           5.0          3.4           1.5          0.2  setosa
8           4.4          2.9           1.4          0.2  setosa
9           4.9          3.1           1.5          0.1  setosa
>>>
>>> # Display the last 5 rows.
... oml_iris.tail()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.7          3.0           5.2          2.3  virginica
1           6.3          2.5           5.0          1.9  virginica
2           6.5          3.0           5.2          2.0  virginica
3           6.2          3.4           5.4          2.3  virginica
4           5.9          3.0           5.1          1.8  virginica

>>>
>>> # Display the last 10 rows.
... oml_iris.tail(10)
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.7          3.1           5.6          2.4  virginica
1           6.9          3.1           5.1          2.3  virginica
2           5.8          2.7           5.1          1.9  virginica
3           6.8          3.2           5.9          2.3  virginica
4           6.7          3.3           5.7          2.5  virginica
5           6.7          3.0           5.2          2.3  virginica
6           6.3          2.5           5.0          1.9  virginica
7           6.5          3.0           5.2          2.0  virginica
8           6.2          3.4           5.4          2.3  virginica
9           5.9          3.0           5.1          1.8  virginica
```

Select Data by Column

Example 9-2 Selecting Data by Columns

The example selects two columns from `oml_iris` and creates the `oml.DataFrame` object `iris_projected1` with them. It then displays the first three rows of `iris_projected1`. The example also selects a range of columns from `oml_iris`, creates `iris_projected2`, and displays its first three rows. Finally, the example selects columns from `oml_iris` by data types, creates `iris_projected3`, and displays its first three rows.

```
# Select all rows with the specified column names.
iris_projected1 = oml_iris[:, ["Sepal_Length", "Petal_Length"]]
iris_projected1.head(3)

# Select all rows with columns whose indices are in the range [1, 4).
iris_projected2 = oml_iris[:, 1:4]
iris_projected2.head(3)

# Select all rows with columns of oml.String data type.
iris_projected3 = oml_iris.select_types(include=[oml.String])
iris_projected3.head(3)
```

Listing for This Example

```
>>> # Select all rows with specified column names.
... iris_projected1 = oml_iris[:, ["Sepal_Length", "Petal_Length"]]
>>> iris_projected1.head(3)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
   Sepal_Length  Petal_Length
0           5.1           1.4
1           4.9           1.4
2           4.7           1.3
>>>
>>> # Select all rows with columns whose indices are in range [1, 4).
... iris_projected2 = oml_iris[:, 1:4]
>>> iris_projected2.head(3)
   Sepal_Width  Petal_Length  Petal_Width
0          3.5           1.4          0.2
1          3.0           1.4          0.2
2          3.2           1.3          0.2
>>>
>>> # Select all rows with columns of oml.String data type.
... iris_projected3 = oml_iris.select_types(include=[oml.String])
>>> iris_projected3.head(3)
  Species
0  setosa
1  setosa
2  setosa
```

Select Data by Value

Example 9-3 Selecting Data by Value

This example filters `oml_iris` to produce `iris_of_filtered1`, which contains the values from the rows of `oml_iris` that have a petal length of less than 1.5 and that are in the Sepal_Length and Petal_Length columns. The example also filters the data using conditions, so that `oml_iris_filtered2` contains the values from `oml_iris` that have a petal length of less than 1.5 or a sepal length equal to 5.0 and `oml_iris_filtered3` contains the values from `oml_iris` that have a petal length of less than 1.5 and a sepal length larger than 5.0.

```
# Select sepal length and petal length where petal length
# is less than 1.5.
oml_iris_filtered1 = oml_iris[oml_iris["Petal_Length"] < 1.5,
                                        ["Sepal_Length", "Petal_Length"]]
len(oml_iris_filtered1)
oml_iris_filtered1.head(3)

### Using the AND and OR conditions in filtering.
# Select all rows in which petal length is less than 1.5 or sepal length
# sepal length is 5.0.
oml_iris_filtered2 = oml_iris[(oml_iris["Petal_Length"] < 1.5) |
                              (oml_iris["Sepal_Length"] == 5.0), :]
len(oml_iris_filtered2)
oml_iris_filtered2.head(3)

# Select all rows in which petal length is less than 1.5 and
# sepal length is larger than 5.0.
oml_iris_filtered3 = oml_iris[(oml_iris["Petal_Length"] < 1.5) &
                              (oml_iris["Sepal_Length"] > 5.0), :]
len(oml_iris_filtered3)
oml_iris_filtered3.head()
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Listing for This Example

```
>>> # Select sepal length and petal length where petal length
... # is less than 1.5.
... oml_iris_filtered1 = oml_iris[oml_iris["Petal_Length"] < 1.5,
...                                        ["Sepal_Length", "Petal_Length"]]
>>> len(oml_iris_filtered1)
24
>>> oml_iris_filtered1.head(3)
   Sepal_Length  Petal_Length
0           5.1           1.4
1           4.9           1.4
2           4.7           1.3
>>>
>>> ### Using the AND and OR conditions in filtering.
... # Select all rows in which petal length is less than 1.5 or
... # sepal length is 5.0.
... oml_iris_filtered2 = oml_iris[(oml_iris["Petal_Length"] < 1.5) |
...                                (oml_iris["Sepal_Length"] == 5.0), :]
>>> len(oml_iris_filtered2)
30
>>> oml_iris_filtered2.head(3)
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
>>>
>>> # Select all rows in which petal length is less than 1.5
... # and sepal length is larger than 5.0.
... oml_iris_filtered3 = oml_iris[(oml_iris["Petal_Length"] < 1.5) &
...                                (oml_iris["Sepal_Length"] > 5.0), :]
>>> len(oml_iris_filtered3)
7
>>> oml_iris_filtered3.head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width Species
0           5.1          3.5           1.4          0.2  setosa
1           5.8          4.0           1.2          0.2  setosa
2           5.4          3.9           1.3          0.4  setosa
3           5.1          3.5           1.4          0.3  setosa
4           5.2          3.4           1.4          0.2  setosa
```

9.1.3 Combine Data

You can join data from `oml.DataFrame` objects that represent database tables by using the `append`, `concat`, and `merge` methods.

Examples of using these methods are in the following topics.

- Append Data from One Object to Another Object
- Combine Two Objects
- Join Data From Two Objects

Append Data from One Object to Another Object

Use the `append` method to join two objects of the same data type.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 9-4 Appending Data from Two Tables

This example first appends the `oml.Float` series object `num1` to another `oml.Float` series object, `num2`. It then appends an `oml.DataFrame` object to another `oml.DataFrame` object, which has the same column types.

```
import oml
import pandas as pd

df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
                   "val" : ["a", "b", "c", "d", "e"],
                   "ch" : ["p", "q", "r", "a", "b"],
                   "num" : [4, 3, 6.7, 7.2, 5]})
oml_df = oml.push(df)

# Append an oml.Float series object to another.
num1 = oml_df['id']
num2 = oml_df['num']
num1.append(num2)

# Append an oml.DataFrame object to another.
x = oml_df[['id', 'val']] # 1st column oml.Float, 2nd column oml.String
y = oml_df[['num', 'ch']] # 1st column oml.Float, 2nd column oml.String
x.append(y)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
...                    "val" : ["a", "b", "c", "d", "e"],
...                    "ch" : ["p", "q", "r", "a", "b"],
...                    "num" : [4, 3, 6.7, 7.2, 5]})
>>> oml_df = oml.push(df)
>>>
>>> # Append an oml.Float series object to another.
... num1 = oml_df['id']
>>> num2 = oml_df['num']
>>> num1.append(num2)
[1, 2, 3, 4, 5, 4, 3, 6.7, 7.2, 5]
>>>
>>> # Explicitly convert oml.Integer to oml.Float
>>> oml.Float(num1).append(num2)
>>> # Append an oml.DataFrame object to another.
... x = oml_df[['id', 'val']] # 1st column oml.Float, 2nd column oml.String
>>> y = oml_df[['num', 'ch']] # 1st column oml.Float, 2nd column oml.String
>>> x.append(y)
    id val
0  1.0   a
1  2.0   b
2  3.0   c
3  4.0   d
4  5.0   e
5  4.0   p
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
6  3.0   q
7  6.7   r
8  7.2   a
9  5.0   b
```

Combine Two Objects

Use the `concat` method to combine columns from one object with those of another object. The `auto_name` argument of the `concat` method controls whether to invoke automatic name conflict resolution. You can also perform customized renaming by passing in a dictionary mapping strings to objects.

To combine two objects with the `concat` method, both objects must represent data from the same underlying database table, view, or query.

Example 9-5 Combining Data Column-Wise

This example first combines the two `oml.DataFrame` objects `x` and `y` column-wise. It then concatenates object `y` with the `oml.Float` series object `w`.

```
import oml
import pandas as pd
from collections import OrderedDict

df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
                   "val" : ["a", "b", "c", "d", "e"],
                   "ch" : ["p", "q", "r", "a", "b"],
                   "num" : [4, 3, 6.7, 7.2, 5]})
oml_df = oml.push(df)

# Create two oml.DataFrame objects and combine the objects column-wise.
x = oml_df[['id', 'val']]
y = oml_df[['num', 'ch']]
x.concat(y)

# Create an oml.Float object with the rounded exponential of two times
# the values in the num column of the oml_df object, then
# concatenate it with the oml.DataFrame object y using a new column name.
w = (oml_df['num']*2).exp().round(decimals=2)
y.concat({'round(exp(2*num))':w})

# Concatenate object x with multiple objects and turn on automatic
# name conflict resolution.
z = oml_df[:,'id']
x.concat([z, w, y], auto_name=True)

# Concatenate multiple oml data objects and perform customized renaming.
x.concat(OrderedDict([('ID',z), ('round(exp(2*num))',w), ('New_',y)]))
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from collections import OrderedDict
>>>
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...                   "val" : ["a", "b", "c", "d", "e"],
...                   "ch" : ["p", "q", "r", "a", "b"],
...                   "num" : [4, 3, 6.7, 7.2, 5]})
>>> oml_df = oml.push(df)

>>> # Create two oml.DataFrame objects and combine the objects column-wise.
... x = oml_df[['id', 'val']]
>>> y = oml_df[['num', 'ch']]
>>> x.concat(y)
   id val  num  ch
0   1   a  4.0   p
1   2   b  3.0   q
2   3   c  6.7   r
3   4   d  7.2   a
4   5   e  5.0   b
>>>
>>> # Create an oml.Float object with the rounded exponential of two times
... # the values in the num column of the oml_df object, then
... # concatenate it with the oml.DataFrame object y using a new column name.
... w = (oml_df['num']*2).exp().round(decimals=2)
>>> y.concat({'round(exp(2*num))':w})
   num ch round(exp(2*num))
0  4.0  p           2980.96
1  3.0  q            403.43
2  6.7  r         660003.22
3  7.2  a        1794074.77
4  5.0  b          22026.47
>>>
>>> # Concatenate object x with multiple objects and turn on automatic
... # name conflict resolution.
... z = oml_df[:,'id']
>>> x.concat([z, w, y], auto_name=True)
  id  val  id3         num  num5  ch
0  1    a    1     2980.96   4.0  p
1  2    b    2      403.43   3.0  q
2  3    c    3   660003.22   6.7  r
3  4    d    4  1794074.77   7.2  a
4  5    e    5    22026.47   5.0  b
>>>
>>> # Concatenate multiple oml data objects and perform customized renaming.
... x.concat(OrderedDict([('ID',z), ('round(exp(2*num))',w), ('New_',y)]))
  id  val  ID  round(exp(2*num))  New_num  New_ch
0  1    a   1           2980.96      4.0       p
1  2    b   2            403.43      3.0       q
2  3    c   3         660003.22      6.7       r
3  4    d   4        1794074.77      7.2       a
4  5    e   5          22026.47      5.0       b
```

Join Data From Two Objects

Use the `merge` method to join data from two objects.

Example 9-6 Joining Data from Two Tables

This example first performs a cross join on the `oml.DataFrame` objects `x` and `y`, which creates the `oml.DataFrame` object `xy`. The example performs a left outer join on the first four rows of `x`

Copyright © 2024, 2026, Oracle and/or its affiliates.

with the `oml.DataFrame` object other on the shared column id and applies the suffixes `.l` and `.r` to column names on the left and right side, respectively. The example then performs a right outer join on the id column on the left side object `x` and the num column on the right side object `y`.

```
import oml
import pandas as pd

df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
                   "val" : ["a", "b", "c", "d", "e"],
                   "ch" : ["p", "q", "r", "a", "b"],
                   "num" : [4, 3, 6.7, 7.2, 5]})
oml_df = oml.push(df)

x = oml_df[['id', 'val']]
y = oml_df[['num', 'ch']]

# Perform a cross join.
xy = x.merge(y)
xy

# Perform a left outer join.
x.head(4).merge(other=oml_df[['id', 'num']], on="id",
                suffixes=['.l','.r'])

# Perform a right outer join.
x.merge(other=y, left_on="id", right_on="num", how="right")
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({"id" : [1, 2, 3, 4, 5],
...                    "val" : ["a", "b", "c", "d", "e"],
...                    "ch" : ["p", "q", "r", "a", "b"],
...                    "num" : [4, 3, 6.7, 7.2, 5]})
>>> oml_df = oml.push(df)
>>>
>>> x = oml_df[['id', 'val']]
>>> y = oml_df[['num', 'ch']]
>>>
>>> # Perform a cross join.
... xy = x.merge(y)
>>> xy
    id_l val_l  num_r ch_r
0      1     a    4.0    p
1      1     a    3.0    q
2      1     a    6.7    r
3      1     a    7.2    a
4      1     a    5.0    b
5      2     b    4.0    p
6      2     b    3.0    q
7      2     b    6.7    r
8      2     b    7.2    a
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
9      2     b    5.0    b
10     3     c    4.0    p
11     3     c    3.0    q
12     3     c    6.7    r
13     3     c    7.2    a
14     3     c    5.0    b
15     4     d    4.0    p
16     4     d    3.0    q
17     4     d    6.7    r
18     4     d    7.2    a
19     4     d    5.0    b
20     5     e    4.0    p
21     5     e    3.0    q
22     5     e    6.7    r
23     5     e    7.2    a
24     5     e    5.0    b
>>>
>>> # Perform a left outer join.
... x.head(4).merge(other=oml_df[['id', 'num']], on="id",
...                 suffixes=['.l','.r'])
   id val.l  num.r
0   1     a    4.0
1   2     b    3.0
2   3     c    6.7
3   4     d    7.2
>>>
>>> # Perform a right outer join.
... x.merge(other=y, left_on="id", right_on="num", how="right")
   id_l val_l  num_r ch_r
0   3.0     c    3.0    q
1   4.0     d    4.0    p
2   5.0     e    5.0    b
3   NaN  None    6.7    r
4   NaN  None    7.2    a
```

9.1.4 Clean Data

In preparing data for analysis, a typical step is to transform data by dropping some values.

You can filter out unneeded data by using the `drop`, `drop_duplicates`, and `dropna` methods.

Example 9-7 Filtering Data

This example demonstrates ways of dropping columns with the `drop` method, dropping missing values with the `dropna` method, and dropping duplicate values with the `drop_duplicates` method.

```
import pandas as pd
import oml

df = pd.DataFrame({'numeric': [1, 1.4, -4, -4, 5.432, None, None],
                   'string1' : [None, None, 'a', 'a', 'a', 'b', None],
                   'string2': ['x', None, 'z', 'z', 'z', 'x', None]})
oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                                 'string1':'CHAR(1)',
                                 'string2':'CHAR(1)'})
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Drop rows with any missing values.
oml_df.dropna(how='any')

# Drop rows in which all column values are missing.
oml_df.dropna(how='all')

# Drop rows in which any numeric column values are missing.
oml_df.dropna(how='any', subset=['numeric'])

# Drop duplicate rows.
oml_df.drop_duplicates()

# Drop rows that have the same value in column 'string1' and 'string2'.
oml_df.drop_duplicates(subset=['string1', 'string2'])

# Drop column 'string2'
oml_df.drop('string2')
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, -4, 5.432, None, None],
...                    'string1' : [None, None, 'a', 'a', 'a', 'b', None],
...                    'string2': ['x', None, 'z', 'z', 'z', 'x', None]})
>>> oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                  'string1':'CHAR(1)',
...                                  'string2':'CHAR(1)'})
>>>
>>> # Drop rows with any missing values.
... oml_df.dropna(how='any')
   numeric string1 string2
0   -4.000       a       z
1   -4.000       a       z
2    5.432       a       z
>>>
>>> # Drop rows in which all column values are missing.
... oml_df.dropna(how='all')
   numeric string1 string2
0    1.000    None       x
1    1.400    None    None
2   -4.000       a       z
3   -4.000       a       z
4    5.432       a       z
5      NaN       b       x
>>>
>>> # Drop rows in which any numeric column values are missing.
... oml_df.dropna(how='any', subset=['numeric'])
   numeric string1 string2
0    1.000    None       x
1    1.400    None    None
2   -4.000       a       z
3   -4.000       a       z
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
4    5.432       a       z
>>>
>>> # Drop duplicate rows.
... oml_df.drop_duplicates()
   numeric string1 string2
0    5.432       a       z
1    1.000    None       x
2   -4.000       a       z
3      NaN       b       x
4    1.400    None    None
5      NaN    None    None
>>>
>>> # Drop rows that have the same value in columns 'string1' and 'string2'.
... oml_df.drop_duplicates(subset=['string1', 'string2'])
   numeric string1 string2
0     -4.0       a       z
1      1.4    None    None
2      1.0    None       x
3      NaN       b       x
>>>
>>> # Drop the column 'string2'.
... oml_df.drop('string2')
   numeric string1
0    1.000    None
1    1.400    None
2   -4.000       a
3   -4.000       a
4    5.432       a
5      NaN       b
6      NaN    None
```

9.1.5 Split Data

Sample and randomly partition data with the `split` and `KFold` methods.

In analyzing large data sets, a typical operation is to randomly partition the data set into subsets for training and testing purposes, which you can do with these methods. You can also sample data with the `split` method.

Example 9-8 Splitting Data into Multiple Sets

This example demonstrates splitting data into multiple sets and into k consecutive folds, which can be used for k-fold cross-validation.

```
import oml
import pandas as pd
from sklearn import datasets

digits = datasets.load_digits()
pd_digits = pd.DataFrame(digits.data,
                         columns=['IMG'+str(i) for i in
                         range(digits['data'].shape[1])])
pd_digits = pd.concat([pd_digits,
                       pd.Series(digits.target,
                                 name = 'target')],
                                 axis = 1)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
oml_digits = oml.push(pd_digits)

# Sample 20% and 80% of the data.
splits = oml_digits.split(ratio=(.2, .8), use_hash = False)
[len(split) for split in splits]

# Split the data into four sets.
splits = oml_digits.split(ratio = (.25, .25, .25, .25),
                          use_hash = False)
[len(split) for split in splits]

# Perform stratification on the target column.
splits = oml_digits.split(strata_cols=['target'])
[split.shape for split in splits]

# Verify that the stratified sampling generates splits in which
# all of the different categories of digits (digits 0~9)
# are present in each split.
[split['target'].drop_duplicates().sort_values().pull()
for split in splits]

# Hash on the target column.
splits = oml_digits.split(hash_cols=['target'])
[split.shape for split in splits]

# Verify that the different categories of digits (digits 0~9) are present
# in only one of the splits generated by hashing on the category column.
[split['target'].drop_duplicates().sort_values().pull()
for split in splits]

# Split the data randomly into 4 consecutive folds.
folds = oml_digits.KFold(n_splits=4)
[(len(fold[0]), len(fold[1])) for fold in folds]
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> digits = datasets.load_digits()
>>> pd_digits = pd.DataFrame(digits.data,
...                          columns=['IMG'+str(i) for i in
...                          range(digits['data'].shape[1])])
>>> pd_digits = pd.concat([pd_digits,
...                        pd.Series(digits.target,
...                                  name = 'target')],
...                                  axis = 1)
>>> oml_digits = oml.push(pd_digits)
>>>
>>> # Sample 20% and 80% of the data.
... splits = oml_digits.split(ratio=(.2, .8), use_hash = False)
>>> [len(split) for split in splits]
[351, 1446]
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> # Split the data into four sets.
... splits = oml_digits.split(ratio = (.25, .25, .25, .25),
...                           use_hash = False)
>>> [len(split) for split in splits]
[432, 460, 451, 454]
>>>
>>> # Perform stratification on the target column.
... splits = oml_digits.split(strata_cols=['target'])
>>> [split.shape for split in splits]
[(1285, 65), (512, 65)]
>>>
>>> # Verify that the stratified sampling generates splits in which
... # all of the different categories of digits (digits 0~9)
... # are present in each split.
... [split['target'].drop_duplicates().sort_values().pull()
... for split in splits]
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]
>>>
>>> # Hash on the target column
... splits = oml_digits.split(hash_cols=['target'])
>>> [split.shape for split in splits]
[(899, 65), (898, 65)]
>>>
>>> # Verify that the different categories of digits (digits 0~9) are present
... # in only one of the splits generated by hashing on the category column.
... [split['target'].drop_duplicates().sort_values().pull()
... for split in splits]
[[0, 1, 3, 5, 8], [2, 4, 6, 7, 9]]
>>>
>>> # Split the data randomly into 4 consecutive folds.
... folds = oml_digits.KFold(n_splits=4)
>>> [(len(fold[0]), len(fold[1])) for fold in folds]
[(1352, 445), (1336, 461), (1379, 418), (1325, 472)]
```

- Define the Target Variable
  The target variable (y), also known as the dependent variable or label, is the outcome that a machine learning model aims to predict or classify.

9.1.5.1 Define the Target Variable

The target variable (y), also known as the dependent variable or label, is the outcome that a machine learning model aims to predict or classify.

The target variable (y) can be an OML object or a string.

- If target variable (y) is a single-column OML object, the target values specified by y must be compatible with the input data (x), meaning they should have the same structure (e.g., the same number of rows).
- If target variable (y) is a string, it represents the name of the column in x that contains the target values (labels) for the model. In this case, x is expected to include a column with the name specified by y, which will be used as the target for training or prediction.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 9-9 Target Variable as an OML Object or a String

This example demonstrates the two different ways to specify the target variable.

```
import oml
import pandas as pd
from sklearn import datasets

# Create training and test datasets with the target variable specified as an
OML object.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create training and test datasets with the target variable specified as a
string.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
train_y = 'Species'
test_dat = dat[1]
```

9.2 Explore Data

OML4Py provides methods that enable you to perform exploratory data analysis and common statistical operations.

These methods are described in the following topics.

- About the Exploratory Data Analysis Methods
  OML4Py provides methods that enable you to perform exploratory data analysis.
- Correlate Data
  Use the `corr` method to perform Pearson, Spearman, or Kendall correlation analysis across columns where possible in an `oml.DataFrame` object.
- Cross-Tabulate Data
  Use the `crosstab` method to perform cross-column analysis of an `oml.DataFrame` object and the `pivot_table` method to convert an `oml.DataFrame` to a spreadsheet-style pivot table.
- Mutate Data
  In preparing data for analysis, a typical operation is to mutate data by reformatting it or deriving new columns and adding them to the data set.
- Sort Data
  The `sort_values` function enables flexible sorting of an `oml.DataFrame` along one or more columns specified by the `by` argument, and returns an `oml.DataFrame`.
- Summarize Data
  The `describe` method calculates descriptive statistics that summarize the central tendency, dispersion, and shape of the data in each column.
- Date, Time, and Integer Data
  OML4Py provides the data types that enable you to manipulate date, time and integer.

Copyright © 2024, 2026, Oracle and/or its affiliates.

9.2.1 About the Exploratory Data Analysis Methods

OML4Py provides methods that enable you to perform exploratory data analysis.

The following table lists methods of OML4Py data type classes with which you can perform common statistical operations and indicates whether the class supports the method.

Table 9-2 Data Exploration Methods Supported by Data Type Classes

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Datetime	oml.Timedelta	oml.Timezone	oml.Integer
`corr`	Computes pairwise correlation between all columns in an `oml.DataFrame` where possible, given the type of coefficient.	⊗	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗
`count`	Computes the number of elements that are not NULL in the series data object or in each column of an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`crosstab`	Computes a cross-tabulation of two or more columns in an `oml.DataFrame`.	⊗	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗
`cumsum`	Computes the cumulative sum after an `oml.Float` series data object is sorted, or of each float or Boolean column after an `oml.DataFrame` object is sorted.	⊗	⊗	✓	⊗	✓	⊗	⊗	⊗	✓
`describe`	Computes descriptive statistics that summarize the central tendency, dispersion, and shape of an `oml` series data distribution, or of each column in an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 9-2 (Cont.) Data Exploration Methods Supported by Data Type Classes

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Datetime	oml.Timedelta	oml.Timezone	oml.Integer
`kurtosis`	Computes the kurtosis of the values in an `oml.Float` series data object, or for each float column in an `oml.DataFrame`.	⊗	⊗	✓	⊗	✓	⊗	⊗	⊗	✓
`max`	Returns the maximum value in a series data object or in each column in an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`mean`	Computes the mean of the values in an `oml.Float` series object, or for each float or Boolean column in an `oml.DataFrame`.	⊗	⊗	✓	⊗	✓	⊗	⊗	⊗	✓
`median`	Computes the median of the values in an `oml.Float` series object, or for each float column in an `oml.DataFrame`.	⊗	⊗	✓	⊗	✓	⊗	⊗	⊗	✓
`min`	Returns the minimum value in a series data object or of each column in an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`nunique`	Computes the number of unique values in a series data object or in each column of an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓
`pivot_table`	Converts an `oml.DataFrame` to a spreadsheet-style pivot table.	⊗	⊗	⊗	⊗	✓	⊗	⊗	⊗	⊗
`sort_values`	Sorts the values in a series data object or sorts the rows in an `oml.DataFrame`.	✓	✓	✓	✓	✓	✓	✓	✓	✓

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 9-2 (Cont.) Data Exploration Methods Supported by Data Type Classes

Method	Description	oml.Boolean	oml.Bytes	oml.Float	oml.String	oml.DataFrame	oml.Datetime	oml.Timedelta	oml.Timezone	oml.Integer
`skew`	Computes the skewness of the values in an `oml.Float` data series object or of each float column in an `oml.DataFrame`.	⊗	⊗	✓	⊗	✓	⊗	⊗	⊗	✓
`std`	Computes the standard deviation of the values in an `oml.Float` data series object or in each float or Boolean column in an `oml.DataFrame`.	⊗	⊗	✓	⊗	✓	⊗	⊗	⊗	✓
`sum`	Computes the sum of the values in an `oml.Float` data series object or of each float or Boolean column in an `oml.DataFrame`.	⊗	⊗	✓	⊗	✓	⊗	⊗	⊗	✓

9.2.2 Correlate Data

Use the `corr` method to perform Pearson, Spearman, or Kendall correlation analysis across columns where possible in an `oml.DataFrame` object.

For details about the function arguments, invoke `help(oml.DataFrame.corr)` or see *Oracle Machine Learning for Python API Reference*.

Example 9-10 Performing Basic Correlation Calculations

This example first creates a temporary database table, with its corresponding proxy `oml.DataFrame` object `oml_df1`, from the `pandas.DataFrame` object `df`. It then verifies the correlation computed between columns A and B, which gives 1, as expected. The values in B are twice the values in A element-wise. The example also changes a value field in `df` and creates a `NaN` entry. It then creates a temporary database table, with the corresponding proxy `oml.DataFrame` object `oml_df2`. Finally, it invokes the `corr` method on `oml_df2` with `skipna` set to `True` (the default) and then `False` to compare the results.

```
import oml
import pandas as pd

df = pd.DataFrame({'A': range(4), 'B': [2*i for i in range(4)]})
oml_df1 = oml.push(df)

# Verify that the correlation between column A and B is 1.
oml_df1.corr()
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Change a value to test the change in the computed correlation result.
df.loc[2, 'A'] = 1.5

# Change an entry to NaN (not a number) to test the 'skipna'
# parameter in the corr method.
df.loc[1, 'B'] = None

# Push df to the database using the floating point column type
# because NaNs cannot be used in Oracle numbers.
oml_df2 = oml.push(df, oranumber=False)

# By default, 'skipna' is True.
oml_df2.corr()
oml_df2.corr(skipna=False)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> df = pd.DataFrame({'A': range(4), 'B': [2*i for i in range(4)]})
>>> oml_df1 = oml.push(df)
>>>
>>> # Verify that the correlation between column A and B is 1.
... oml_df1.corr()
   A  B
A  1  1
B  1  1
>>>
>>> # Change a value to test the change in the computed correlation result.
... df.loc[2, 'A'] = 1.5
>>>
>>> # Change an entry to NaN (not a number) so to test the 'skipna'
... # parameter in the corr method.
... df.loc[1, 'B'] = None
>>>
>>> # Push df to the database using the floating point column type
... # because NaNs cannot be used in Oracle numbers.
... oml_df2 = oml.push(df, oranumber=False)
>>>
>>> # By default, 'skipna' is True.
... oml_df2.corr()
          A         B
A  1.000000  0.981981
B  0.981981  1.000000
>>> oml_df2.corr(skipna=False)
     A    B
A  1.0  NaN
B  NaN  1.0
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

9.2.3 Cross-Tabulate Data

Use the `crosstab` method to perform cross-column analysis of an `oml.DataFrame` object and the `pivot_table` method to convert an `oml.DataFrame` to a spreadsheet-style pivot table.

Cross-tabulation is a statistical technique that finds an interdependent relationship between two columns of values. The `crosstab` method computes a cross-tabulation of two or more columns. By default, it computes a frequency table for the columns unless a column and an aggregation function have been passed to it.

The `pivot_table` method converts a data set into a pivot table. Due to the database 1000 column limit, pivot tables with more than 1000 columns are automatically truncated to display the categories with the most entries for each column value.

For details about the method arguments, invoke `help(oml.DataFrame.crosstab)` or `help(oml.DataFrame.pivot_table)`, or see *Oracle Machine Learning for Python API Reference*.

Example 9-11 Producing Cross-Tabulation and Pivot Tables

This example demonstrates the use of the `crosstab` and `pivot_table` methods.

```
import pandas as pd
import oml

x = pd.DataFrame({
     'GENDER': ['M', 'M', 'F', 'M', 'F', 'M', 'F', 'F',
                None, 'F', 'M', 'F'],
     'HAND': ['L', 'R', 'R', 'L', 'R', None, 'L', 'R',
              'R', 'R', 'R', 'R'],
     'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1,
               39.6, 46.4, 12, 25.3, 37.5],
     'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93,
                  .89, .84, .91, .95]
    })
x = oml.push(x)

# Find the categories that the most entries belonged to.
x.crosstab('GENDER', 'HAND').sort_values('count', ascending=False)

# For each gender value and across all entries, find the ratio of entries
# with different hand values.
x.crosstab('GENDER', 'HAND', pivot = True, margins = True, normalize = 0)

# Find the mean speed across all gender and hand combinations.
x.pivot_table('GENDER', 'HAND', 'SPEED')

# Find the median accuracy and speed for every gender and hand combination.
x.pivot_table('GENDER', 'HAND', aggfunc = oml.DataFrame.median)

# Find the max and min speeds for every gender and hand combination and
# across all combinations.
x.pivot_table('GENDER', 'HAND', 'SPEED',
              aggfunc = [oml.DataFrame.max, oml.DataFrame.min],
              margins = True)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> x = pd.DataFrame({
...       'GENDER': ['M', 'M', 'F', 'M', 'F', 'M', 'F', 'F',
...                  None, 'F', 'M', 'F'],
...       'HAND': ['L', 'R', 'R', 'L', 'R', None, 'L', 'R',
...                'R', 'R', 'R', 'R'],
...       'SPEED': [40.5, 30.4, 60.8, 51.2, 54, 29.3, 34.1,
...                 39.6, 46.4, 12, 25.3, 37.5],
...       'ACCURACY': [.92, .94, .87, .9, .85, .97, .96, .93,
...                    .89, .84, .91, .95]
...     })
>>> x = oml.push(x)
>>>
>>> # Find the categories that the most entries belonged to.
... x.crosstab('GENDER', 'HAND').sort_values('count', ascending=False)
  GENDER  HAND  count
0      F     R      5
1      M     L      2
2      M     R      2
3      M  None      1
4      F     L      1
5   None     R      1
>>>
>>> # For each gender value and across all entries, find the ratio of entries
... # with different hand values.
... x.crosstab('GENDER', 'HAND', pivot = True, margins = True, normalize = 0)
  GENDER  count_(L)  count_(R)  count_(None)
0   None   0.000000   1.000000      0.000000
1      F   0.166667   0.833333      0.000000
2      M   0.400000   0.400000      0.200000
3    All   0.250000   0.666667      0.083333
>>>
>>> # Find the mean speed across all gender and hand combinations.
... x.pivot_table('GENDER', 'HAND', 'SPEED')
  GENDER  mean(SPEED)_(L)  mean(SPEED)_(R)  mean(SPEED)_(None)
0   None              NaN            46.40                 NaN
1      F            34.10            40.78                 NaN
2      M            45.85            27.85                29.3
>>>
>>> # Find the median accuracy and speed for every gender and hand
combination.
... x.pivot_table('GENDER', 'HAND', aggfunc = oml.DataFrame.median)
  GENDER  median(ACCURACY)_(L)  median(ACCURACY)_(R)
median(ACCURACY)_(None)  \
0   None                   NaN                 0.890
NaN
1      F                  0.96                 0.870
NaN
2      M                  0.91                 0.925
0.97

   median(SPEED)_(L)  median(SPEED)_(R)  median(SPEED)_(None)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
0               NaN             46.40                 NaN
1             34.10             39.60                 NaN
2             45.85             27.85                29.3
>>>
>>> # Find the max and min speeds for every gender and hand combination and
... # across all combinations.
... x.pivot_table('GENDER', 'HAND', 'SPEED',
...               aggfunc = [oml.DataFrame.max, oml.DataFrame.min],
...               margins = True)
  GENDER  max(SPEED)_(L)  max(SPEED)_(R)  max(SPEED)_(None)
max(SPEED)_(All)  \
0   None             NaN            46.4                NaN
46.4
1      F            34.1            60.8                NaN
60.8
2      M            51.2            30.4               29.3
51.2
3    All            51.2            60.8               29.3
60.8

   min(SPEED)_(L)  min(SPEED)_(R)  min(SPEED)_(None)  min(SPEED)_(All)
0             NaN            46.4                NaN              46.4
1            34.1            12.0                NaN              12.0
2            40.5            25.3               29.3              25.3
3            34.1            12.0               29.3              12.0
```

9.2.4 Mutate Data

In preparing data for analysis, a typical operation is to mutate data by reformatting it or deriving new columns and adding them to the data set.

These examples demonstrate methods of formatting data and deriving columns.

```
import pandas as pd
import oml

# Create a shopping cart data set.
shopping_cart = pd.DataFrame({
  'Item_name': ['paper_towel', 'ground_pork', 'tofu', 'eggs',
                'pork_loin', 'whole_milk', 'egg_custard'],
  'Item_type': ['grocery', 'meat', 'grocery', 'dairy', 'meat',
                'dairy', 'bakery'],
  'Quantity': [1, 2.6, 4, 1, 1.9, 1, 1],
  'Unit_price': [1.19, 2.79, 0.99, 2.49, 3.19, 2.5, 3.99]
  })
oml_cart = oml.push(shopping_cart)
oml_cart

# Add a column 'Price' multiplying 'Quantity' with 'Unit_price',
# rounded to 2 decimal places.
price = oml_cart['Quantity']*(oml_cart['Unit_price'])
type(price)
price
oml_cart = oml_cart.concat({'Price': price.round(2)})
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Count the pattern 'egg' in the 'Item_name' column.
egg_pattern = oml_cart['Item_name'].count_pattern('egg')
type(egg_pattern)
oml_cart.concat({'Egg_pattern': egg_pattern})

# Find the start index of substring 'pork' in the 'Item_name' column.
pork_startInd = oml_cart['Item_name'].find('pork')
type(pork_startInd)
oml_cart.concat({'Pork_startInd': pork_startInd})

# Check whether items are of grocery category.
is_grocery=oml_cart['Item_type']=='grocery'
type(is_grocery)
oml_cart.concat({'Is_grocery': is_grocery})

# Calculate the length of item names.
name_length=oml_cart['Item_name'].len()
type(name_length)
oml_cart.concat({'Name_length': name_length})

# Get the ceiling, floor, exponential, logarithm and square root
# of the 'Price' column.
oml_cart['Price'].ceil()
oml_cart['Price'].floor()
oml_cart['Price'].exp()
oml_cart['Price'].log()
oml_cart['Price'].sqrt()
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> # Create a shopping cart data set.
... shopping_cart = pd.DataFrame({
...   'Item_name': ['paper_towel', 'ground_pork', 'tofu', 'eggs',
...                 'pork_loin', 'whole_milk', 'egg_custard'],
...   'Item_type': ['grocery', 'meat', 'grocery', 'dairy', 'meat',
...                 'dairy', 'bakery'],
...   'Quantity': [1, 2.6, 4, 1, 1.9, 1, 1],
...   'Unit_price': [1.19, 2.79, 0.99, 2.49, 3.19, 2.5, 3.99]
...   })
>>> oml_cart = oml.push(shopping_cart)
>>> oml_cart
     Item_name Item_type  Quantity  Unit_price
0  paper_towel   grocery       1.0        1.19
1  ground_pork      meat       2.6        2.79
2         tofu   grocery       4.0        0.99
3         eggs     dairy       1.0        2.49
4    pork_loin      meat       1.9        3.19
5   whole_milk     dairy       1.0        2.50
6  egg_custard    bakery       1.0        3.99
>>>
>>> # Add a column 'Price' multiplying 'Quantity' with 'Unit_price',
... # rounded to 2 decimal places.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
... price = oml_cart['Quantity']*(oml_cart['Unit_price'])
>>> type(price)
<class 'oml.core.float.Float'>
>>> price
[1.19, 7.254, 3.96, 2.49, 6.061, 2.5, 3.99]
>>> oml_cart = oml_cart.concat({'Price': price.round(2)})
>>>
>>> # Count the pattern 'egg' in the 'Item_name' column.
... egg_pattern = oml_cart['Item_name'].count_pattern('egg')
>>> type(egg_pattern)
<class 'oml.core.float.Float'>
>>> oml_cart.concat({'Egg_pattern': egg_pattern})
     Item_name Item_type  Quantity  Unit_price  Price  Egg_pattern
0  paper_towel   grocery       1.0        1.19   1.19            0
1  ground_pork      meat       2.6        2.79   7.25            0
2         tofu   grocery       4.0        0.99   3.96            0
3         eggs     dairy       1.0        2.49   2.49            1
4    pork_loin      meat       1.9        3.19   6.06            0
5   whole_milk     dairy       1.0        2.50   2.50            0
6  egg_custard    bakery       1.0        3.99   3.99            1
>>>
>>> # Find the start index of substring 'pork' in the 'Item_name' column.
... pork_startInd = oml_cart['Item_name'].find('pork')
>>> type(pork_startInd)
<class 'oml.core.float.Float'>
>>> oml_cart.concat({'Pork_startInd': pork_startInd})
     Item_name Item_type  Quantity  Unit_price  Price  Pork_startInd
0  paper_towel   grocery       1.0        1.19   1.19             -1
1  ground_pork      meat       2.6        2.79   7.25              7
2         tofu   grocery       4.0        0.99   3.96             -1
3         eggs     dairy       1.0        2.49   2.49             -1
4    pork_loin      meat       1.9        3.19   6.06              0
5   whole_milk     dairy       1.0        2.50   2.50             -1
6  egg_custard    bakery       1.0        3.99   3.99             -1
>>>
>>> # Check whether items are of grocery category.
... is_grocery=oml_cart['Item_type']=='grocery'
>>> type(is_grocery)
<class 'oml.core.boolean.Boolean'>
>>> oml_cart.concat({'Is_grocery': is_grocery})
     Item_name Item_type  Quantity  Unit_price  Price  Is_grocery
0  paper_towel   grocery       1.0        1.19   1.19        True
1  ground_pork      meat       2.6        2.79   7.25       False
2         tofu   grocery       4.0        0.99   3.96        True
3         eggs     dairy       1.0        2.49   2.49       False
4    pork_loin      meat       1.9        3.19   6.06       False
5   whole_milk     dairy       1.0        2.50   2.50       False
6  egg_custard    bakery       1.0        3.99   3.99       False
>>>
>>> # Calculate the length of item names.
... name_length=oml_cart['Item_name'].len()
>>> type(name_length)
<class 'oml.core.float.Float'>
>>> oml_cart.concat({'Name_length': name_length})
     Item_name Item_type  Quantity  Unit_price  Price  Name_length
0  paper_towel   grocery       1.0        1.19   1.19           11
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
1  ground_pork      meat       2.6        2.79   7.25          11
2         tofu   grocery       4.0        0.99   3.96           4
3         eggs     dairy       1.0        2.49   2.49           4
4    pork_loin      meat       1.9        3.19   6.06           9
5   whole_milk     dairy       1.0        2.50   2.50          10
6  egg_custard    bakery       1.0        3.99   3.99          11
>>>
>>> # Get the ceiling, floor, exponential, logarithm and square root
... # of the 'Price' column.
... oml_cart['Price'].ceil()
[2, 8, 4, 3, 7, 3, 4]
>>> oml_cart['Price'].floor()
[1, 7, 3, 2, 6, 2, 3]
>>> oml_cart['Price'].exp()
[3.2870812073831184, 1408.1048482046956, 52.45732594909905,
12.061276120444719, 428.37543685928694, 12.182493960703473, 54.05488936332659]
>>> oml_cart['Price'].log()
[0.173953307123438, 1.9810014688665833, 1.3762440252663892,
0.9122827104766162, 1.801709800081223, 0.9162907318741551, 1.3837912309017721]
>>> oml_cart['Price'].sqrt()
[1.0908712114635715, 2.692582403567252, 1.98997487421324, 1.57797338380595,
2.4617067250182343, 1.5811388300841898, 1.997498435543818]
```

9.2.5 Sort Data

The `sort_values` function enables flexible sorting of an `oml.DataFrame` along one or more columns specified by the `by` argument, and returns an `oml.DataFrame`.

Example 9-12 Sorting Data

The following example demonstrates these operations.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Modify the data set by replacing a few entries with NaNs to test
#  how the na_position parameter works in the sort_values method.
Iris = oml_iris.pull()
Iris['Sepal_Width'].replace({3.5: None}, inplace=True)
Iris['Petal_Length'].replace({1.5: None}, inplace=True)
Iris['Petal_Width'].replace({2.3: None}, inplace=True)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Create another table using the changed data.
oml_iris2 =  oml.create(Iris, table = 'IRIS2')

# Sort the data set first by Sepal_Length then by Sepal_Width
# in descending order and display the first 5 rows of the
# sorted result.
oml_iris2.sort_values(by = ['Sepal_Length', 'Sepal_Width'],
                          ascending=False).head()

# Display the last 5 rows of the data set.
oml_iris2.tail()

# Sort the last 5 rows of the iris data set first by Petal_Length
# then by Petal_Width. By default, rows with NaNs are placed
# after the other rows when the sort keys are the same.
oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'])

# Sort the last 5 rows of the iris data set first by Petal_Length
# and then by Petal_Width. When the values in these two columns
# are the same, place the row with a NaN before the other row.
oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'],
                                na_position = 'first')

oml.drop('IRIS')
oml.drop('IRIS2')
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Modify the data set by replacing a few entries with NaNs to test
... # how the na_position parameter works in the sort_values method.
... Iris = oml_iris.pull()
>>> Iris['Sepal_Width'].replace({3.5: None}, inplace=True)
>>> Iris['Petal_Length'].replace({1.5: None}, inplace=True)
>>> Iris['Petal_Width'].replace({2.3: None}, inplace=True)
>>>
>>> # Create another table using the changed data.
... oml_iris2 =  oml.create(Iris, table = 'IRIS2')
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> # Sort the data set first by 'Sepal_Length' then by 'Sepal_Width'
... # in descending order and displays the first 5 rows of the
... # sorted result.
... oml_iris2.sort_values(by = ['Sepal_Length', 'Sepal_Width'],
...                            ascending=False).head()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           7.9          3.8           6.4          2.0  virginica
1           7.7          3.8           6.7          2.2  virginica
2           7.7          3.0           6.1          NaN  virginica
3           7.7          2.8           6.7          2.0  virginica
4           7.7          2.6           6.9          NaN  virginica
>>>
>>> # Display the last 5 rows of the data set.
... oml_iris2.tail()
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.7          3.0           5.2          NaN  virginica
1           6.3          2.5           5.0          1.9  virginica
2           6.5          3.0           5.2          2.0  virginica
3           6.2          3.4           5.4          NaN  virginica
4           5.9          3.0           5.1          1.8  virginica
>>>
>>> # Sort the last 5 rows of the iris data set first by 'Petal_Length'
... # then by 'Petal_Width'. By default, rows with NaNs are placed
... # after  the other rows when the sort keys are the same.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'])
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.3          2.5           5.0          1.9  virginica
1           5.9          3.0           5.1          1.8  virginica
2           6.5          3.0           5.2          2.0  virginica
3           6.7          3.0           5.2          NaN  virginica
4           6.2          3.4           5.4          NaN  virginica
>>>
>>> # Sort the last 5 rows of the iris data set first by 'Petal_Length'
... # and then by 'Petal_Width'. When the values in these two columns
... # are the same, place the row with a NaN before the other row.
... oml_iris2.tail().sort_values(by = ['Petal_Length', 'Petal_Width'],
...                               na_position = 'first')
   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0           6.3          2.5           5.0          1.9  virginica
1           5.9          3.0           5.1          1.8  virginica
2           6.7          3.0           5.2          NaN  virginica
3           6.5          3.0           5.2          2.0  virginica
4           6.2          3.4           5.4          NaN  virginica
>>>
>>> oml.drop('IRIS')
>>> oml.drop('IRIS2')
```

9.2.6 Summarize Data

The `describe` method calculates descriptive statistics that summarize the central tendency, dispersion, and shape of the data in each column.

You can also specify the types of columns to include or exclude from the results.

With the `sum` and `cumsum` methods, you can compute the sum and cumulative sum of each Float or Boolean column of an `oml.DataFrame`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The `describe` method supports finding the following statistics:

- Mean, minimum, maximum, median, top character, standard deviation
- Number of not-Null values, unique values, top characters
- Percentiles between 0 and 1

Example 9-13 Calculating Descriptive Statistics

The following example demonstrates these operations.

```
import pandas as pd
import oml

df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
                   'string' : [None, None, 'a', 'a', 'a', 'b'],
                   'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e']})

oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
                                 'string':'CHAR(1)',
                                 'bytes':'RAW(1)'})

# Combine a Boolean column with oml_df.
oml_bool = oml_df['numeric'] > 3
oml_df = oml_df.concat(oml_bool)
oml_df.rename({'COL4':'boolean'})

# Describe all of the columns.
oml_df.describe(include='all')

# Exclude Float columns.
oml_df.describe(exclude=[oml.Float])

# Get the sum of values in each Float or Boolean column.
oml_df.sum()

# Find the cumulative sum of values in each Float or Boolean column
# after oml_df is sorted by the bytes column in descending order.
oml_df.cumsum(by = 'bytes', ascending = False)

# Compute the skewness of values in the Float columns.
oml_df.skew()

# Find the median value of Float columns.
oml_df.median()

# Calculate the kurtosis of Float columns.
oml_df.kurtosis()
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> df = pd.DataFrame({'numeric': [1, 1.4, -4, 3.145, 5, None],
...                    'string' : [None, None, 'a', 'a', 'a', 'b'],
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...                    'bytes' : [b'a', b'b', b'c', b'c', b'd', b'e']})
>>>
>>> oml_df = oml.push(df, dbtypes = {'numeric': 'BINARY_DOUBLE',
...                                  'string':'CHAR(1)',
...                                  'bytes':'RAW(1)'})
>>>
>>> # Combine a Boolean column with oml_df.
... oml_bool = oml_df['numeric'] > 3
>>> oml_df = oml_df.concat(oml_bool)
>>> oml_df.rename({'COL4':'boolean'})
  bytes  numeric string  boolean
0  b'a'    1.000   None    False
1  b'b'    1.400   None    False
2  b'c'   -4.000      a    False
3  b'c'    3.145      a     True
4  b'd'    5.000      a     True
5  b'e'      NaN      b     True
>>>
>>> # Describe all of the columns.
... oml_df.describe(include='all')
       bytes   numeric string boolean
count      6  5.000000      4       6
unique     5       NaN      2       2
top     b'c'       NaN      a   False
freq       2       NaN      3       3
mean     NaN  1.309000    NaN     NaN
std      NaN  3.364655    NaN     NaN
min      NaN -4.000000    NaN     NaN
25%      NaN  1.000000    NaN     NaN
50%      NaN  1.400000    NaN     NaN
75%      NaN  3.145000    NaN     NaN
max      NaN  5.000000    NaN     NaN
>>>
>>> # Exclude Float columns.
... oml_df.describe(exclude=[oml.Float])
       bytes string boolean
count      6      4       6
unique     5      2       2
top     b'c'      a   False
freq       2      3       3
>>>
>>> # Get the sum of values in each Float or Boolean column.
... oml_df.sum()
numeric    6.545
boolean    3.000
dtype: float64
>>>
>>> # Find the cumulative sum of values in each Float or Boolean column
... # after oml_df is sorted by the bytes column in descending order.
... oml_df.cumsum(by = 'bytes', ascending = False)
   numeric  boolean
0      NaN        1
1    5.000        2
2    1.000        2
3    4.145        3
4    5.545        3
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
5    6.545        3
>>>
>>> # Compute the skewness of values in the Float columns.
... oml_df.skew()
numeric   -0.683838
dtype: float64
>>>
>>> # Find the median value of Float columns.
... oml_df.median()
numeric    1.4
dtype: float64
>>>
>>> # Calculate the kurtosis of Float columns.
... oml_df.kurtosis()
numeric   -0.582684
dtype: float64
```

9.2.7 Date, Time, and Integer Data

OML4Py provides the data types that enable you to manipulate date, time and integer.

The following newly added data types are now supported in OML4Py:

- `oml.Datetime`
- `oml.Timezone`
- `oml.Timedelta`
- `oml.Integer`

For information on the attributes and methods of `oml.Datetime`, `oml.Timezone`, `oml.Timedelta` and `oml.Integer`, see *Oracle Machine Learning for Python API Reference*.

oml.Datetime

To create a date, you can use the datetime class of the datetime module, which is included in OML4Py.

The datetime class requires three parameters: `year`, `month`, and `day`. It also contains optional parameters for time and timezone that includes `hour`, `minute`, `second`, `microsecond`, and `tzone`.

Example 9-14 Using the oml.Datetime Function

This example creates a proxy object from a table with DATE column.

```
import oml
import pandas as pd
import numpy as np
import datetime
from datetime import datetime, timezone, timedelta
SALES = oml.sync(schema="SH", table="SALES")
z.show(SALES.head())

# Use the following command to compute the statistics on table columns:
pd.set_option('display.max_columns', 50)
pd.set_option('display.width', 1000)
SALES.describe(include='all')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Use the following command to compute statistics on DATE column TIME_ID:
SALES['TIME_ID'].describe()

# Use the following command to extract date-related features:
date = SALES['TIME_ID']
SALES2 = SALES.concat({'YEAR': date.year, 'MONTH': date.month})
SALES2.head()
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np
>>> import datetime
>>> from datetime import datetime, timezone, timedelta
>>> SALES = oml.sync(schema="SH", table="SALES")
>>> z.show(SALES.head())
>>>  PROD_ID  CUST_ID    TIME_ID              CHANNEL_ID  PROMO_ID
QUANTITY_SOLD  AMOUNT_SOLD
     13       524        1998-01-20 00:00:00  2           999
1.0            1205.99
     13       2128       1998-04-05 00:00:00  2           999
1.0            1250.25
     13       3212       1998-04-05 00:00:00  2           999
1.0            1250.25
     13       3375       1998-04-05 00:00:00  2           999
1.0            1250.25
     13       5204       1998-04-05 00:00:00  2           999
1.0            1250.25
>>> pd.set_option('display.max_columns', 50)
>>> pd.set_option('display.width', 1000)
>>> SALES.describe(include='all')
>>>          PROD_ID         CUST_ID             TIME_ID
CHANNEL_ID         PROMO_ID    QUANTITY_SOLD        AMOUNT_SOLD
count  918843.000000   918843.000000              918843
918843.000000    918843.000000         918843.0      918843.000000
unique NaN            NaN                           1460
NaN              NaN                NaN                NaN
top    NaN            NaN            2001-10-18 00:00:00
NaN              NaN                NaN                NaN
freq   NaN            NaN                           2940
NaN              NaN                NaN                NaN
mean   78.183945      7289.807720                    NaN
2.861603       976.396093              1.0         106.879882
std    49.008014      8948.653221                    NaN
0.686874       121.829887              0.0         259.780490
min    13.000000      2.000000                       NaN
2.000000        33.000000              1.0           6.400000
25%    31.000000      2383.000000                    NaN
2.000000       999.000000              1.0          17.380000
50%    48.000000      4927.000000                    NaN
3.000000       999.000000              1.0          34.240000
75%   127.000000      9163.000000                    NaN
3.000000       999.000000              1.0          53.890000
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
max   148.000000    101000.000000                    NaN
9.000000       999.000000             1.0        1782.720000
>>> SALES['TIME_ID'].describe()
>>> count                  918843
    unique                   1460
    top       2001-10-18 00:00:00
    freq                     2940
    Name: TIME_ID, dtype: object
>>> date = SALES['TIME_ID']
>>> SALES2 = SALES.concat({'YEAR': date.year, 'MONTH': date.month})
>>> SALES2.head()
>>> PROD_ID  CUST_ID    TIME_ID  CHANNEL_ID  PROMO_ID  QUANTITY_SOLD
AMOUNT_SOLD  YEAR  MONTH
0        13      524 1998-01-20           2       999            1.0
1205.99  1998      1
1        13     2128 1998-04-05           2       999            1.0
1250.25  1998      4
2        13     3212 1998-04-05           2       999            1.0
1250.25  1998      4
3        13     3375 1998-04-05           2       999            1.0
1250.25  1998      4
4        13     5204 1998-04-05           2       999            1.0
1250.25  1998      4
```

Example 9-15 Using the oml.Datetime Function

This example creates a datetime object with the year, month, day, then creates a temporary proxy object using `oml.push`. The `oml.push` function requires a data frame or list as input, so the date object is converted to a list. The resulting object is an object of class `oml.Datetime`.

```
import oml
import pandas as pd
import numpy as np

import datetime
from datetime import datetime, timezone, timedelta

d1 = datetime(year=2004, month=7, day=24)

print ('d1:', d1)
print('d1 year:', d1.year)
print('d1 month:', d1.month)
d1_lst = [d1]
D1 = oml.push(d1_lst)

print ('type', type(D1))
print ('D1:', D1)
print ('year:', D1.year)
print ('month:', D1.month)
print ('day:', D1.day)
d2 = datetime.fromisoformat('2004-07-24 00:05:23+04:00')
d2_lst=[d2]

D2 = oml.push(d2_lst)

print('D2', D2)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
print('type', type(D2))
D2.strftime()
D2.strftime()
d3 = "14-Jul-05 20:01:01"
d3_lst = [d3]
D3 = oml.push(d3_lst)
oml.Datetime.strptime(D3, "DD-Mon-RR HH24:MI:SS")
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np

>>> import datetime
>>> from datetime import datetime, timezone, timedelta

>>> d1 = datetime(year=2004, month=7, day=24)

>>> print ('d1:', d1)
>>> d1: 2004-07-24 00:00:00
>>> print('d1 year:', d1.year)
>>> d1 year: 2004
>>> print('d1 month:', d1.month)
>>> d1 month: 7

>>> d1_lst = [d1]
>>> D1 = oml.push(d1_lst)

>>> print ('type', type(D1))
>>> type <class 'oml.core.datetime.Datetime'>
>>> print ('D1:', D1)
>>> D1: [datetime.datetime(2004, 7, 24, 0, 0)]
>>> print ('year:', D1.year)
>>> year: [2004]
>>> print ('month:', D1.month)
>>> month: [7]
>>> print ('day:', D1.day)
>>> day: [24]

>>> d2 = datetime.fromisoformat('2004-07-24 00:05:23+04:00')
>>> d2_lst=[d2]
>>> D2 = oml.push(d2_lst)
>>> print('D2', D2)
>>> D2 [datetime.datetime(2004, 7, 24, 0, 5, 23,
tzinfo=datetime.timezone(datetime.timedelta(seconds=14400)))]
>>> print('type', type(D2))
>>> type <class 'oml.core.datetime.Datetime'>

>>> D2.strftime()
>>> ['2004-07-24 00:05:23+04:00']
>>> d3 = "14-Jul-05 20:01:01"
>>> d3_lst = [d3]

>>> D3 = oml.push(d3_lst)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> oml.Datetime.strptime(D3, "DD-Mon-RR HH24:MI:SS")
>>> [datetime.datetime(2005, 7, 14, 20, 1, 1)]
```

oml.Timedelta

`oml.Timedelta` objects represent a span of time, which can be used to perform simple arithmetic operations on `oml.Datetime` objects. The `oml.Timedelta` objects can be multiplied by an integer value or a floating point value. Subtracting dates creates an `oml.Timedelta` object that can be added, subtracted, or multiplied by a `oml.Timedate` object to produce another date.

Example 9-16 Using the oml.Timedelta Function

This example creates a time-based reference using the current date and time, and creates an `oml.Timedelta` object named DELT1 to determine a past and future dates.

```
import oml
import pandas as pd
import numpy as np

import datetime
from datetime import datetime, timezone, timedelta
today = datetime.now()
print('today:', today)

delt1 = timedelta(days=1, hours=2, seconds=5)
print('delt1:', delt1)

dat = pd.DataFrame({'datetime': [today], 'timedelta': [delt1]})
DAT = oml.push(dat, dbtypes = ['TIMESTAMP', 'INTERVAL DAY TO SECOND'])

TODAY = DAT['datetime']
DELT1 = DAT['timedelta']

print('TODAY:', today)
print('DELT1:', DELT1)
past_date1 = TODAY - DELT1
print('past date 1:', past_date1)

past_date2 = TODAY - (DELT1 *3)
print('past date 2:', past_date2)

future_date1 = TODAY + DELT1
print('future date 1:', future_date1)

future_date2 = TODAY + (DELT1 *3)
print('future date 2:', future_date2)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> import numpy as np

>>> import datetime
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> from datetime import datetime, timezone, timedelta
>>> today = datetime.now()
>>> print('today:', today)
>>> today: 2022-12-27 06:00:14.555899

>>> delt1 = timedelta(days=1, hours=2, seconds=5)
>>> print('delt1:', delt1)
>>> delt1: 1 day, 2:00:05

>>> dat = pd.DataFrame({'datetime': [today], 'timedelta': [delt1]})
>>> DAT = oml.push(dat, dbtypes = ['TIMESTAMP', 'INTERVAL DAY TO SECOND'])

>>> TODAY = DAT['datetime']
>>> DELT1 = DAT['timedelta']

>>> print('TODAY:', today)
>>> TODAY: 2022-12-27 06:00:14.555899
>>> print('DELT1:', DELT1)
>>> DELT1: [datetime.timedelta(days=1, seconds=7205)]
>>> past_date1 = TODAY - DELT1
>>> print('past date 1:', past_date1)
>>> past date 1: [datetime.datetime(2022, 12, 26, 4, 0, 9, 555899)]
>>> past_date2 = TODAY - (DELT1 *3)
>>> print('past date 2:', past_date2)
>>> past date 2: [datetime.datetime(2022, 12, 23, 23, 59, 59, 555899)]

>>> future_date1 = TODAY + DELT1
>>> print('future date 1:', future_date1)
>>> future date 1: [datetime.datetime(2022, 12, 28, 8, 0, 19, 555899)]

>>> future_date2 = TODAY + (DELT1 *3)
>>> print('future date 2:', future_date2)
>>> future date 2: [datetime.datetime(2022, 12, 30, 12, 0, 29, 555899)]
```

oml.Integer

The `oml.Integer` class represents the integer data type.

Example 9-17 Using the oml.Integer Function

This example creates an oml.Integer object named INTEGER1 to represent the integer data type.

```
import oml
import pandas as pd
integer1 = oml.push(pd.DataFrame({'INTEGER': [0, -12, 1234, 40, 95]}),
dbtypes = "NUMBER(*, 0)")
integer1
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> integer1 = oml.push(pd.DataFrame({'INTEGER': [0, -12, 1234, 40, 95]}),
dbtypes = "NUMBER(*, 0)")
>>> integer1
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
       INTEGER
0        0
1      -12
2     1234
3       40
4       95
```

Compare two `oml.Datetime` columns in a proxy object using standard arithmetic comparison operators.

Example 9-18 Using value comparison Function

This example compares two `oml.Datetime` columns in a proxy object using standard arithmetic comparison operators.

```
d1 = datetime(2005, 7, 14, 5, 10, 30)
d2 = datetime(2004, 6, 30, 1, 22, 46)
d3 = datetime(2003, 12, 10, 12, 50, 25)
d4 = datetime(2002, 3, 20, 20, 42, 59)

d3 = pd.DataFrame({'X': [d1, d2], 'Y': [d3,
d4]})
D3 = oml.push(d3, dbtypes = ['TIMESTAMP', 'TIMESTAMP'])

print(D3)
D4 = D3['X']
D5 = D3['Y']

print("D4:", D4)
print("D4 type:", type(D4))

print("D5:", D5)
print("D5 type:", type(D5))

print(D4 == D5)

print(D4 != D5)

print(D4 > D5)

print(D4 >= D5)

print(D4 < D5)

print(D4 <= D5)

print("max:", D3['X'].max())
print("min:", D3['Y'].min())
```

Listing for This Example

```
>>> d1 = datetime(2005, 7, 14, 5, 10, 30)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> d2 = datetime(2004, 6, 30, 1, 22, 46)
>>> d3 = datetime(2003, 12, 10, 12, 50, 25)
>>> d4 = datetime(2002, 3, 20, 20, 42, 59)

>>> d3 = pd.DataFrame({'X': [d1, d2], 'Y': [d3,
d4]})
>>> D3 = oml.push(d3, dbtypes = ['TIMESTAMP', 'TIMESTAMP'])

>>> print(D3)
>>>                 X                   Y
>>> 0 2005-07-14 05:10:30 2003-12-10 12:50:25
>>> 1 2004-06-30 01:22:46 2002-03-20 20:42:59
>>> D4 = D3['X']
>>> D5 = D3['Y']

>>> print("D4:", D4)
>>> print("D4 type:", type(D4))
>>> D4: [datetime.datetime(2005, 7, 14, 5, 10, 30), datetime.datetime(2004,
6, 30, 1, 22, 46)]
>>> D4 type: <class 'oml.core.datetime.Datetime'>

>>> print("D5:", D5)
>>> print("D5 type:", type(D5))
>>> D5: [datetime.datetime(2003, 12, 10, 12, 50, 25), datetime.datetime(2002,
3, 20, 20, 42, 59)]
>>> D5 type: <class 'oml.core.datetime.Datetime'>

>>> print(D4 == D5)
>>> [False, False]

>>> print(D4 != D5)
>>> [True, True]

>>> print(D4 > D5)
>>> [True, True]

>>> print(D4 >= D5)
>>> [True, True]

>>> print(D4 < D5)
>>> [False, False]

>>> print(D4 <= D5)
>>> [False,
False]

                      >>> [True, True] [True, True] [True, True] [False,
False] [False,

>>> print("max:", D3['X'].max())
>>> max: 2005-07-14 05:10:30
>>> print("min:", D3['Y'].min())
>>> min: 2002-03-20 20:42:59
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Value Replacement

This function updates the elements of an `oml.Datetime` object, such as year, month, and day.

Example 9-19 Using value replacement function

```
D4 = D3['X'].replace(year=2000)
print("D4:", D4)

D5 = D3['X'].replace(month=11)
print("D5:", D5)

D6 = D3['X'].replace(day=6)
print("D6:", D6)
```

Listing for This Example

```
>>> D4 = D3['X'].replace(year=2000)
>>> print("D4:", D4)
>>> D4: [datetime.datetime(2000, 7, 14, 5, 10, 30), datetime.datetime(2000,
6, 30, 1, 22, 46)]

>>> D5 = D3['X'].replace(month=11)
>>> print("D5:", D5)
>>> D5: [datetime.datetime(2005, 11, 14, 5, 10, 30), datetime.datetime(2004,
11, 30, 1, 22, 46)]

>>> D6 = D3['X'].replace(day=6)
>>> print("D6:", D6)
>>> D6: [datetime.datetime(2005, 7, 6, 5, 10, 30), datetime.datetime(2004, 6,
6, 1, 22, 46)]
```

9.3 Render Graphics

OML4Py provides functions for rendering graphical displays of data.

The `oml.graphics.boxplot` and `oml.graphics.hist` functions compute the statistics necessary to generate box and whisker plots or histograms in-database for scalability and performance.

OML4Py uses the `matplotlib` library to render the output. You can use methods of `matplotlib.pyplot` to customize the created images and `matplotlib.pyplot.show` to show the images. By default, rendered graphics have the same properties as those stored in `matplotlib.rcParams`.

For the parameters of the `oml.graphics.boxplot` and `oml.graphics.hist` functions, invoke `help(oml.graphics.boxplot)` or `help(oml.graphics.hist)`, or see Oracle Machine Learning for Python API Reference.

Generate a Box Plot

Use the `oml.graphics.boxplot` function to generate a box and whisker plot for every column of *x* or for every column object in *x*.

Copyright © 2024, 2026, Oracle and/or its affiliates.

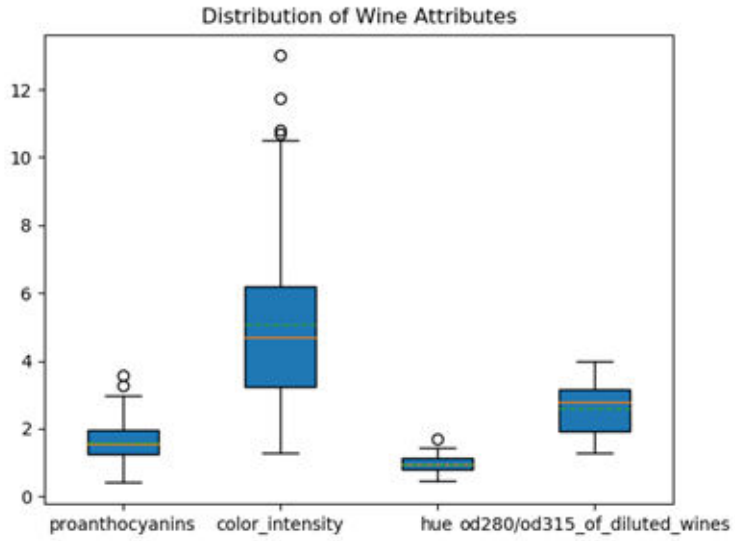
Example 9-20 Using the oml.graphics.boxplot Function

This example first loads the wine data set from `sklearn` and creates the `pandas.DataFrame` object `wine_data`. It then creates a temporary database table, with its corresponding proxy `oml.DataFrame` object `oml_wine`, from `wine_data`. It draws a box and whisker plot on every column with the index ranging from 8 to 12 (not including 12) in `oml_wine`. The arguments `showmeans` and `meanline` are set to `True` to show the arithmetic means and to render the mean as a line spanning the full width of the box. The argument `patch_artist` is set to `True` to have the boxes drawn with Patch artists.

```
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets

wine = datasets.load_wine()
wine_data = pd.DataFrame(wine.data, columns = wine.feature_names)
oml_wine = oml.push(wine_data)
oml.graphics.boxplot(oml_wine[:,8:12], showmeans=True,
                     meanline=True, patch_artist=True,
                     labels=oml_wine.columns[8:12])
plt.title('Distribution of Wine Attributes')
plt.show()
```

The output of the example is the following.



The image shows a box and whisker plot for each of the four columns of the wine data set: Proanthocyanins, Color intensity, Hue, and OD280/OD315 of diluted wines. The boxes extend from the lower to upper quartile values of the data, with a solid orange line at the median. The whiskers that extend from the box show the range of the data. The caps are the horizontal lines at the ends of the whiskers. Flier or outlier points are those past the ends of the whiskers. The mean is shown as a green dotted line spanning the width of the each box.

Generate a Histogram

Use the `oml.graphics.hist` function to compute and draw a histogram for every data set column contained in *x*.

Copyright © 2024, 2026, Oracle and/or its affiliates.

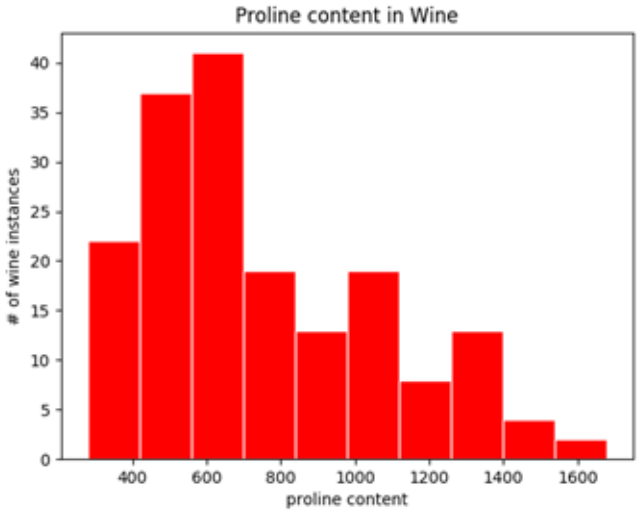
Example 9-21 Using the oml.graphics.hist Function

This example first loads the wine data set from `sklearn` and creates the `pandas.DataFrame` object `wine_data`. It then creates a temporary database table, with its corresponding proxy `oml.DataFrame` object `oml_wine`, from `wine_data`. Next it draws a histogram on the proline column of `oml_wine`. The argument `bins` specifies generating ten equal-width bins. Argument `color` specifies filling the bars with the color purple. Arguments `linestyle` and `edgecolor` are set to draw the bar edges as solid lines in pink.

```
import oml
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine

wine = load_wine()
wine_data = pd.DataFrame(wine.data, columns = wine.feature_names)
oml_wine = oml.push(wine_data)
oml.graphics.hist(oml_wine['proline'], bins=10, color='red',
                  linestyle='solid', edgecolor='white')
plt.title('Proline content in Wine')
plt.xlabel('proline content')
plt.ylabel('# of wine instances')
plt.show()
```

The output of the example is the following.



The image shows a traditional bar-type histogram for the Proline column of the wine data set. The range of proline values is divided into 10 bins of equal size. The height of the rectangular bar for each bin indicates the number of wine instances in each bin. The bars are red with solid white edges.

Copyright © 2024, 2026, Oracle and/or its affiliates.

10

OML4Py Classes That Provide Access to In-Database Machine Learning Algorithms

OML4Py has classes that provide access to in-database Oracle Machine Learning algorithms.

These classes are described in the following topics.

Topics:

- About Machine Learning Classes and Algorithms
  These classes provide access to in-database machine learning algorithms.
- About Model Settings
  You can specify settings that affect the characteristics of a model.
- Shared Settings
  These settings are common to all of the OML4Py machine learning classes.
- Export Oracle Machine Learning for Python Models
  You can export an `oml` model from Python and then score it in SQL.
- Automatic Data Preparation
  Oracle Machine Learning for Python supports Automatic Data Preparation (ADP) and user-directed general data preparation.
- Model Explainability
  Use the OML4Py Explainability module to identify the important features that impact a trained model's predictions.
- Attribute Importance
  The `oml.ai` class computes the relative attribute importance, which ranks attributes according to their significance in predicting a classification or regression target.
- Association Rules
  The `oml.ar` class implements the Apriori algorithm to find frequent itemsets and association rules, all as part of an association model object.
- Decision Tree
  The `oml.dt` class uses the Decision Tree algorithm for classification.
- Expectation Maximization
  The `oml.em` class uses the Expectation Maximization (EM) algorithm to create a clustering model.
- Explicit Semantic Analysis
  The `oml.esa` class extracts text-based features from a corpus of documents and performs document similarity comparisons.
- Generalized Linear Model
  The `oml.glm` class builds a Generalized Linear Model (GLM) model.
- k-Means
  The `oml.km` class uses the *k*-Means (KM) algorithm, which is a hierarchical, distance-based clustering algorithm that partitions data into a specified number of clusters.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Naive Bayes
  The `oml.nb` class creates a Naive Bayes (NB) model for classification.
- Neural Network
  The `oml.nn` class creates a Neural Network (NN) model for classification and regression.
- ONNX
  The `oml.onnx` class allows you to import your own ONNX-format model, load it in the database, and score data using the prediction operators of Oracle Machine Learning.
- Random Forest
  The `oml.rf` class creates a Random Forest (RF) model that provides an ensemble learning technique for classification.
- Singular Value Decomposition
  Use the `oml.svd` class to build a model for feature extraction.
- Support Vector Machine
  The `oml.svm` class creates a Support Vector Machine (SVM) model for classification, regression, or anomaly detection.
- Non-Negative Matrix Factorization
  The `oml.nmf` class creates a Non-Negative Matrix Factorization (NMF) model for feature extraction.
- Exponential Smoothing Method
  The `oml.esm` function uses the Exponential Smoothing Method (ESM) algorithm to create a time series model.
- XGBoost
  The `oml.xgb` class supports the in-database scalable gradient tree boosting algorithm for both classification, regression specifications, ranking models, and survival models. It makes available the open source gradient boosting framework. It prepares the categorical encoding and missing value replacement from the OML infrastructure, calls the in-database XGBoost, builds and persists a model as a first-class database model object, and supports using the model for prediction.

10.1 About Machine Learning Classes and Algorithms

These classes provide access to in-database machine learning algorithms.

Algorithm Classes

Class	Algorithm	Function of Algorithm	Description
`oml.ai`	Minimum Description Length	Attribute importance for classification or regression	Ranks attributes according to their importance in predicting a target.
`oml.ar`	Apriori	Association rules	Performs market basket analysis by identifying co-occurring items (frequent itemsets) within a set.
`oml.dt`	Decision Tree	Classification	Extracts predictive information in the form of human-understandable rules. The rules are if-then-else expressions; they explain the decisions that lead to the prediction.
`oml.em`	Expectation Maximization	Clustering	Performs probabilistic clustering based on a density estimation algorithm.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Class	Algorithm	Function of Algorithm	Description
`oml.esa`	Explicit Semantic Analysis	Feature extraction	Extracts text-based features from a corpus of documents. Performs document similarity comparisons.
`oml.glm`	Generalized Linear Model	Classification Regression	Implements logistic regression for classification of binary targets and linear regression for continuous targets.
`oml.km`	*k*-Means	Clustering	Uses unsupervised learning to group data based on similarity into a predetermined number of clusters.
`oml.nb`	Naive Bayes	Classification	Makes predictions by deriving the probability of a prediction from the underlying evidence, as observed in the data.
`oml.nn`	Neural Network	Classification Regression	Learns from examples and tunes the weights of the connections among the neurons during the learning process.
`oml.oc`	Orthogonal partitioning clustering (O-Cluster)	Clustering	Provides a fast, scalable grid-based clustering algorithm well-suited for analysing large, high-dimensional data sets. It can produce high quality clusters without relying on user-defined parameters.
`oml.rf`	Random Forest	Classification	Provides an ensemble learning technique for classification of data.
`oml.svd`	Singular Value Decomposition	Feature extraction	Performs orthogonal linear transformations that capture the underlying variance of the data by decomposing a rectangular matrix into three matrices.
`oml.svm`	Support Vector Machine	Anomaly detection Classification Regression	Builds a model that is a profile of a class, which, when the model is applied, identifies cases that are somehow different from that profile.
`oml.nmf`	Non-Negative Matrix Factorization	Feature extraction	A state of the art feature extraction algorithm used when there are many attributes and the attributes are ambiguous or have weak predictability.
`oml.esm`	Exponential Smoothing Method (ESM)	Time series model	Enables time series forecasting using Exponential Smoothing Models (ESM). It supports a variety of smoothing techniques for modeling level, trend, and seasonality components in time-dependent data.
`oml.xgb`	XGBoost	Classification Regression	Can be used as a stand-alone predictor or incorporate it into real-world production pipelines for a wide range of problems such as ad click-through rate prediction, hazard risk prediction, web text classification, and so on.
`oml.onnx`	Open Neural Network Exchange	Regression Classification Clustering Embedding	An ONNX-format model can be loaded into the database and used to score data using the prediction operators of Oracle Machine Learning. Although not an "algorithm", this supports models from multiple algorithms and machine learning techniques.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Repeatable Results

You can use the `case_id` parameter in the `fit` method of the OML4Py machine learning algorithm classes to achieve repeatable sampling, data splits (train and held aside), and random data shuffling.

Persisting Models

In-database models created through the OML4Py API exist as temporary objects that are dropped when the database connection ends unless you take one of the following actions:

- Save a default-named model object in a datastore, as in the following example:

```
regr_mod = oml.glm(mining_function = "regression")
oml.ds.save(regr_mod, name = 'regr_ds')
```

- Use the `model_name` parameter in the `fit` function when building the model, as in the following example:

```
regr_mod = regr_mod.fit(X, y, model_name = 'myRegression')
```

- Change the name of an existing model using the `model_name` function of the model, as in the following example:

```
regr_mod.model_name = 'myRegression2'
```

To drop a persistent named model, use the `oml.drop` function.

Creating a Model from an Existing In-Database Model

You can create an OML4Py model as a proxy object for an existing in-database machine learning model. The in-database model could have been created through OML4Py, OML4SQL, or OML4R. To do so, when creating the OML4Py, specify the name of the existing model and, optionally, the name of the owner of the model, as in the following example.

```
ar_mod = oml.ar(model_name = 'existing_ar_model', model_owner = 'SH',
**setting)
```

An OML4Py model created this way persists until you drop it with the `oml.drop` function.

Scoring New Data with a Model

For most of the OML4Py machine learning classes, you can use the `predict` and `predict_proba` methods of the model object to score new data.

For in-database models, you can use the SQL `PREDICTION` function on model proxy objects, which scores directly in the database. You can use in-database models directly from SQL if you prepare the data properly. For open source models, you can use Embedded Python Execution and enable data-parallel execution for performance and scalability.

Deploying Models Through a REST API

The REST API for Oracle Machine Learning Services provides REST endpoints hosted on an Oracle Autonomous AI Database instance. These endpoints allow you to store OML models along with their metadata, and to create scoring endpoints for the models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

10.2 About Model Settings

You can specify settings that affect the characteristics of a model.

Some settings are general, some are specific to an Oracle Machine Learning function, and some are specific to an algorithm.

All settings have default values. If you want to override one or more of the settings for a model, then you must specify the settings with the `**params` parameter when instantiating the model or later by using the `set_params` method of the model.

For the `_init_` method, the argument can be key-value pairs or a `dict`. Each list element's name and value refer to a machine learning algorithm parameter setting name and value, respectively. The setting value must be numeric or a string.

The argument for the `**params` parameter of the `set_params` method is a `dict` object mapping a `str` to a `str`. The key should be the name of the setting, and the value should be the new setting.

Example 10-1 Specifying Model Settings

This example shows the creation of an Expectation Maximization (EM) model and the changing of a setting. For the complete code of the EM model example, see Example 9-10.

```
# Specify settings.
setting = {'emcs_num_iterations': 100}
# Create an EM model object
em_mod = em(n_clusters = 2, **setting)

# Intervening code not shown.

# Change the random seed and refit the model.
em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)
```

10.3 Shared Settings

These settings are common to all of the OML4Py machine learning classes.

The following table lists the settings that are shared by all OML4Py models.

Table 10-1 Shared Model Settings

Setting Name	Setting Value	Description
ODMS_DETAILS	ODMS_ENABLE ODMS_DISABLE	Helps to control model size in the database. Model details can consume significant disk space, especially for partitioned models. The default value is ODMS_ENABLE. If the setting value is ODMS_ENABLE, then model detail tables and views are created along with the model. You can query the model details using SQL. If the value is ODMS_DISABLE, then model detail tables are not created and tables relevant to model details are also not created. The reduction in the space depends on the algorithm. Model size reduction can be on the order of 10x .

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-1 (Cont.) Shared Model Settings

Setting Name	Setting Value	Description
ODMS_MAX_PARTITIONS	1 < *value* <= 1000000	Controls the maximum number of partitions allowed for a partitioned model. The default is 1000.
ODMS_MISSING_VALUE_TREATMENT	ODMS_MISSING_VALUE_AUTO ODMS_MISSING_VALUE_MEAN_MODE ODMS_MISSING_VALUE_DELETE_ROW	Indicates how to treat missing values in the training data. This setting does not affect the scoring data. The default value is ODMS_MISSING_VALUE_AUTO. ODMS_MISSING_VALUE_MEAN_MODE replaces missing values with the mean (numeric attributes) or the mode (categorical attributes) both at build time and apply time where appropriate. ODMS_MISSING_VALUE_AUTO performs different strategies for different algorithms. When ODMS_MISSING_VALUE_TREATMENT is set to ODMS_MISSING_VALUE_DELETE_ROW, the rows in the training data that contain missing values are deleted. However, if you want to replicate this missing value treatment in the scoring data, then you must perform the transformation explicitly. The value ODMS_MISSING_VALUE_DELETE_ROW is applicable to all algorithms.
ODMS_PARTITION_BUILD_TYPE	ODMS_PARTITION_BUILD_INTRA ODMS_PARTITION_BUILD_INTER ODMS_PARTITION_BUILD_HYBRID	Controls the parallel building of partitioned models. ODMS_PARTITION_BUILD_INTRA builds each partition in parallel using all slaves. ODMS_PARTITION_BUILD_INTER builds each partition entirely in a single slave, but multiple partitions may be built at the same time because multiple slaves are active. ODMS_PARTITION_BUILD_HYBRID combines the other two types and is recommended for most situations to adapt to dynamic environments. This is the default value.
ODMS_PARTITION_COLUMNS	Comma separated list of machine learning attributes	Requests the building of a partitioned model. The setting value is a comma-separated list of the machine learning attributes to be used to determine the in-list partition key values. These attributes are taken from the input columns, unless an XFORM_LIST parameter is passed to the model. If XFORM_LIST parameter is passed to the model, then the attributes are taken from the attributes produced by these transformations.
ODMS_TABLESPACE_NAME	*tablespace_name*	Specifies the tablespace in which to store the model. If you explicitly set this to the name of a tablespace (for which you have sufficient quota), then the specified tablespace storage creates the resulting model content. If you do not provide this setting, then the your default tablespace creates the resulting model content.
ODMS_SAMPLE_SIZE	0 < *value*	Determines how many rows to sample (approximately). You can use this setting only if ODMS_SAMPLING is enabled. The default value is system determined.
ODMS_SAMPLING	ODMS_SAMPLING_ENABLE ODMS_SAMPLING_DISABLE	Allows the user to request sampling of the build data. The default is ODMS_SAMPLING_DISABLE.
ODMS_TEXT_MAX_FEATURES	1 <= *value*	The maximum number of distinct features, across all text attributes, to use from a document set passed to the model. The default is 3000. An oml.esa model has the default value of 300000.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-1 (Cont.) Shared Model Settings

Setting Name	Setting Value	Description
ODMS_TEXT_MIN_DOCUMENTS	Non-negative value	This text processing setting controls how many documents a token needs to appear in to be used as a feature. The default is 1. An oml.esa model has the default value of 3.
ODMS_TEXT_POLICY_NAME	The name of an Oracle Text POLICY created using CTX_DDL.CREATE_POLICY.	Affects how individual tokens are extracted from unstructured text. For details about CTX_DDL.CREATE_POLICY, see *Oracle Text Reference*.
PREP_AUTO	PREP_AUTO_ON PREP_AUTO_OFF	This data preparation setting enables fully automated data preparation. The default is PREP_AUTO_ON.
PREP_SCALE_2DNUM	pPREP_SCALE_STDDEV PREP_SCALE_RANGE	This data preparation setting enables scaling data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values: PREP_SCALE_STDDEV: A request to divide the column values by the standard deviation of the column and is often provided together with PREP_SHIFT_MEAN to yield z-score normalization. PREP_SCALE_RANGE: A request to divide the column values by the range of values and is often provided together with PREP_SHIFT_MIN to yield a range of [0,1].
PREP_SCALE_NNUM	PREP_SCALE_MAXABS	This data preparation setting enables scaling data preparation for nested numeric columns. PREP_AUTO must be OFF for this setting to take effect. If specified, then the valid value for this setting is PREP_SCALE_MAXABS, which yields data in the range of [-1,1].
PREP_SHIFT_2DNUM	PREP_SHIFT_MEAN PREP_SHIFT_MIN	This data preparation setting enables centering data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values: PREP_SHIFT_MEAN: Results in subtracting the average of the column from each value. PREP_SHIFT_MIN: Results in subtracting the minimum of the column from each value.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-1 (Cont.) Shared Model Settings

Setting Name	Setting Value	Description
ODMS_BOXCOX **Note** Available only in Oracle AI Database 26ai.	ODMS_BOXCOX_ENABLE ODMS_BOXCOX_DISABLE	This setting enables the Box-Cox variance-stabilization transformation. It is useful when the variance increases as the target value increases. It reduces variance and transforms a multiplicative relationship with the target, with a simpler additive relationship. This setting is applicable only to the Exponential Smoothing algorithm. When a value for EXSM_MODEL setting is not specified, the default value is ODMS_BOXCOX_ENABLE and when a value for the EXSM_MODEL setting is provided, the default value is ODMS_BOXCOX_DISABLE.
ODMS_EXPLOSION_MIN_SUPP **Note** Available only in Oracle AI Database 26ai.	A positive integer	It is the minimum required support for categorical values that must be included in the explosion mapping. It removes categorical values with insufficient row instances to have a statistically significant effect on the model, because, they could potentially degrade performance or exhaust memory. The default is system determined depending on the number of rows in the dataset. A value of 1 results into mapping all categorical values.

10.4 Export Oracle Machine Learning for Python Models

You can export an `oml` model from Python and then score it in SQL.

Export a Model

With the `export_sermodel` function of an OML4Py algorithm model, you can export the model in a serialized format. You can then score that model in SQL. To save a model to a permanent table, you must pass in a name for the new table. If the model is partitioned, then you can optionally select an individual partition to export; otherwise all partitions are exported.

Note

Any data transformations you apply to the data for model building you must also apply to the data for scoring with the imported model.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 10-2 Export a Trained oml.svm Model to a Database Table

This example creates the `x` and `y` variables using the iris data set. It then creates the persistent database table IRIS and the `oml.DataFrame` object `oml_iris` as a proxy for the table.

This example preprocesses the `iris` data set and splits the data set into training data and test data. It then fits an `oml.svm` model according to the training data of the data set, and saves the fitted model in a serialized format to a new table named svm_sermod in the database.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
    oml.drop('IRIS_TEST_DATA')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

df = oml.sync(table = "IRIS").pull()

# Add a case identifier column.
df.insert(0, 'ID', range(0,len(df)))

# Create training data and test data.
IRIS_TMP = oml.push(df).split()
train_x = IRIS_TMP[0].drop('Species')
train_y = IRIS_TMP[0]['Species']
test_dat = IRIS_TMP[1]

# Create the iris_test_data database table.
oml_test_dat = oml.create(test_dat.pull(), table = "IRIS_TEST_DATA")

# Create an oml SVM model object.
svm_mod = oml.svm('classification',
                  svms_kernel_function =
                    'dbms_data_mining.svms_linear')

# Fit the SVM model with the training data.
svm_mod = svm_mod.fit(train_x, train_y, case_id = 'ID')

# Export the oml.svm model to a new table named 'svm_sermod'
# in the database.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
svm_export = svm_mod.export_sermodel(table='svm_sermod')
type(svm_export)

# Show the first 10 characters of the BLOB content from the
# model export.
svm_export.pull()[0][1:10]
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
...    oml.drop('IRIS_TEST_DATA')
...except:
...    pass
>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> df = oml.sync(table = "IRIS").pull()
>>>
>>> # Add a case identifier column.
... df.insert(0, 'ID', range(0,len(df)))
>>>
>>> # Create training data and test data.
... IRIS_TMP = oml.push(df).split()
>>> train_x = IRIS_TMP[0].drop('Species')
>>> train_y = IRIS_TMP[0]['Species']
>>> test_dat = IRIS_TMP[1]
>>>
>>> # Create the iris_test_data database table.
... oml_test_dat = oml.create(test_dat.pull(), table = "IRIS_TEST_DATA")
>>>
>>> # Create an oml SVM model object.
... svm_mod = oml.svm('classification',
...                   svms_kernel_function =
...                     'dbms_data_mining.svms_linear')
>>>
>>> # Fit the SVM model with the training data.
... svm_mod = svm_mod.fit(train_x, train_y, case_id='ID')
>>>
>>> # Export the oml.svm model to a new table named 'svm_sermod'
... # in the database.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
... svm_export = svm_mod.export_sermodel(table='svm_sermod')
>>> type(svm_export)
<class 'oml.core.bytes.Bytes'>
>>>
>>> # Show the first 10 characters of the BLOB content from the
... # model export.
... svm_export.pull()[0][1:10]
b'\xff\xfc|\x00\x00\x02\x9c\x00\x00'
```

Import a Model

In SQL, you can import the serialized format of an OML4Py model into an Oracle Machine Learning for SQL model with the `DBMS_DATA_MINING.IMPORT_SERMODEL` procedure. To that procedure, you pass the BLOB content from the table to which the model was exported and the name of the model to be created. The import procedure provides the ability to score the model. It does not create model views or tables that are needed for querying model details. You can use the SQL function `PREDICTION` to apply the imported model to the test data and get the prediction results.

Example 10-3 Import a Serialized SVM Model as an OML4SQL Model in SQL

This example retrieves the serialized content of the SVM classification model from the svm_sermod table. It uses the `IMPORT_SERMODEL` procedure to create a model named my_iris_svm_classifier with the content from the table. It also predicts test data saved in the iris_test_data table with the newly imported model my_iris_svm_classifier, and compares the prediction results with the target classes.

```
-- After starting SQL*Plus as the OML4Py user.
-- Import the model from the serialized content.

DECLARE
  v_blob blob;

BEGIN
  SELECT SERVAL INTO v_blob FROM "svm_sermod";
  dbms_data_mining.import_sermodel(v_blob, 'my_iris_svm_classifier');
END;
/

-- Set the output column format.
column TARGET_SPECIES format a15
column PREDICT_SPECIES format a15

-- Make predictions and display cases where mod(ID,3) equals 0.
SELECT ID, "Species" AS TARGET_SPECIES,
  PREDICTION(my_iris_svm_classifier USING "Sepal_Length", "Sepal_Width",
             "Petal_Length", "Petal_Width")
             AS PREDICT_SPECIES
             FROM "IRIS_TEST_DATA" WHERE MOD(ID,3) = 0;

-- Drop the imported model
BEGIN
  DBMS_DATA_MINING.DROP_MODEL(model_name => 'my_iris_svm_classifier');
END;
/
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

The prediction produces the following results.

```
ID TARGET_SPECIES PREDICT_SPECIES
-- -------------- --------------
 0 setosa         setosa
24 setosa         setosa
27 setosa         setosa
33 setosa         setosa
36 setosa         setosa
39 setosa         setosa
48 setosa         setosa
54 versicolor     versicolor
57 versicolor     versicolor
93 versicolor     versicolor
114 virginica     virginica
120 virginica     virginica
132 virginica     virginica
13 rows selected.
```

10.5 Automatic Data Preparation

Oracle Machine Learning for Python supports Automatic Data Preparation (ADP) and user-directed general data preparation.

The PREP_* settings enable you to request fully automated (ADP) or manual data preparation. By default, ADP is enabled (PREP_AUTO_ON) . When performed manually, data preparation requirements of each algorithm must be addressed

When you enable ADP, the model uses heuristics to transform the build data according to the requirements of the algorithm. Instead of ADP, you can request that the data be shifted and/or scaled with the PREP_SCALE_* and PREP_SHIFT_* settings. The transformation instructions are stored with the model and reused whenever the model is applied. The model settings can be viewed in USER_MINING_MODEL_SETTINGS.

PREP_* Settings

The values for the PREP_* settings are described in the following table.

Table 10-2 title

Setting Name	Setting Value	Description
PREP_AUTO	PREP_AUTO_ON PREP_AUTO_OFF	This setting enables fully automated data preparation. The default is PREP_AUTO_ON.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-2 (Cont.) title

Setting Name	Setting Value	Description
PREP_SCALE_2DNUM	PREP_SCALE_STDDEV PREP_SCALE_RANGE	This setting enables scaling data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values. PREP_SCALE_STDDEV: A request to divide the column values by the standard deviation of the column and is often provided together with PREP_SHIFT_MEAN to yield z-score normalization. PREP_SCALE_RANGE: A request to divide the column values by the range of values and is often provided together with PREP_SHIFT_MIN to yield a range of [0,1].
PREP_SCALE_NNUM	PREP_SCALE_MAXABS	This setting enables scaling data preparation for nested numeric columns. PREP_AUTO must be OFF for this setting to take effect. If specified, then the valid value for this setting is PREP_SCALE_MAXABS, which yields data in the range of [-1,1].
PREP_SHIFT_2DNUM	PREP_SHIFT_MEAN PREP_SHIFT_MIN	This setting enables centering data preparation for two-dimensional numeric columns. PREP_AUTO must be OFF for this setting to take effect. The following are the possible values: PREP_SHIFT_MEAN: Results in subtracting the average of the column from each value. PREP_SHIFT_MIN: Results in subtracting the minimum of the column from each value.

See Also

- About Model Settings
- Shared Settings

10.6 Model Explainability

Use the OML4Py Explainability module to identify the important features that impact a trained model's predictions.

Machine Learning Explainability (MLX) is the process of explaining and interpreting machine learning models. The OML MLX Python module supports the ability to help better understand a model's behavior and why it makes its predictions. MLX currently provides model-agnostic explanations for classification and regression tasks where explanations treat the ML model as a black-box, instead of using properties from the model to guide the explanation.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The global feature importance explainer object is the interface to the MLX permutation importance explainer. The global feature importance explainer identifies the most important features for a given model and data set. The explainer is model-agnostic and currently supports tabular classification and regression data sets with both numerical and categorical features.

The algorithm estimates feature importance by evaluating the model's sensitivity to changes in a specific feature. Higher sensitivity suggests that the model places higher importance on that feature when making its predictions than on another feature with lower sensitivity.

For information on the `oml.GlobalFeatureImportance` class attributes and methods, call `help(oml.mlx.GlobalFeatureImportance)` or see Oracle Machine Learning for Python API Reference.

Example 10-4 Binary Classification

This example uses the Breast Cancer binary classification data set. Load the data set into the database and a unique case id column.

```
import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

bc_ds = datasets.load_breast_cancer()
bc_data = bc_ds.data.astype(float)
X = pd.DataFrame(bc_data, columns=bc_ds.feature_names)
y = pd.DataFrame(bc_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(bc_data.shape[0]),
                      columns=['CASE_ID'])
df = oml.create(pd.concat([X, y, row_id], axis=1),
                table='BreastCancer')
```

Split the data set into train and test variables.

```
train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
                       seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train a Random Forest model.

```
model = oml.algo.rf(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
        "RF accuracy score = {:.2f}".format(model.score(X_test, y_test))
```

Create the MLX Global Feature Importance explainer, using the binary `f1` metric.

```
gfi = GlobalFeatureImportance(mining_function='classification',
                              score_metric='f1', random_state=32,
                              parallel=4)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Run the explainer to generate the global feature importance. Here we construct an explanation using the train data set and then display the explanation.

```
explanation = gfi.explain(model, X, y, case_id='CASE_ID', n_iter=10)
explanation
```

Drop the BreastCancer table.

```
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> bc_ds = datasets.load_breast_cancer()
>>> bc_data = bc_ds.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns=bc_ds.feature_names)
>>> y = pd.DataFrame(bc_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc_data.shape[0]),
...                       columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1),
...                 table='BreastCancer')
>>>
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                        seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> model = oml.algo.rf(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
...         "RF accuracy score = {:.2f}".format(model.score(X_test, y_test))
'RF accuracy score = 0.95'
>>>
>>> gfi = GlobalFeatureImportance(mining_function='classification',
...                               score_metric='f1', random_state=32,
...                               parallel=4)
>>>
>>> explanation = gfi.explain(model, X, y, case_id='CASE_ID', n_iter=10)
>>> explanation
Global Feature Importance:
[0] worst concave points: Value: 0.0263, Error: 0.0069
[1] worst perimeter: Value: 0.0077, Error: 0.0027
[2] worst radius: Value: 0.0076, Error: 0.0031
[3] worst area: Value: 0.0045, Error: 0.0037
[4] mean concave points: Value: 0.0034, Error: 0.0033
[5] worst texture: Value: 0.0017, Error: 0.0015
[6] area error: Value: 0.0012, Error: 0.0014
[7] worst concavity: Value: 0.0008, Error: 0.0008
[8] worst symmetry: Value: 0.0004, Error: 0.0007
[9] mean texture: Value: 0.0003, Error: 0.0007
[10] mean perimeter: Value: 0.0003, Error: 0.0015
[11] mean radius: Value: 0.0000, Error: 0.0000
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
[12] mean smoothness: Value: 0.0000, Error: 0.0000
[13] mean compactness: Value: 0.0000, Error: 0.0000
[14] mean concavity: Value: 0.0000, Error: 0.0000
[15] mean symmetry: Value: 0.0000, Error: 0.0000
[16] mean fractal dimension: Value: 0.0000, Error: 0.0000
[17] radius error: Value: 0.0000, Error: 0.0000
[18] texture error: Value: 0.0000, Error: 0.0000
[19] smoothness error: Value: 0.0000, Error: 0.0000
[20] compactness error: Value: 0.0000, Error: 0.0000
[21] concavity error: Value: 0.0000, Error: 0.0000
[22] concave points error: Value: 0.0000, Error: 0.0000
[23] symmetry error: Value: 0.0000, Error: 0.0000
[24] fractal dimension error: Value: 0.0000, Error: 0.0000
[25] worst compactness: Value: 0.0000, Error: 0.0000
[26] worst fractal dimension: Value: 0.0000, Error: 0.0000
[27] mean area: Value: -0.0001, Error: 0.0011
[28] worst smoothness: Value: -0.0003, Error: 0.0013

oml.drop('BreastCancer')
```

Example 10-5 Multi-Class Classification

This example uses the Iris multi-class classification data set. Load the data set into the database, adding a unique case id column.

```
import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

iris_ds = datasets.load_iris()
iris_data = iris_ds.data.astype(float)
X = pd.DataFrame(iris_data, columns=iris_ds.feature_names)
y = pd.DataFrame(iris_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(iris_data.shape[0]),
                      columns=['CASE_ID'])
df = oml.create(pd.concat([X, y, row_id], axis=1), table='Iris')
```

Split the data set into train and test variables.

```
train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
                       seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train an SVM model.

```
model = oml.algo.svm(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
"SVM accuracy score = {:.2f}".format(model.score(X_test, y_test))
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Create the MLX Global Feature Importance explainer, using the `f1_weighted` metric.

```
gfi = GlobalFeatureImportance(mining_function='classification',
                              score_metric='f1_weighted',
                              random_state=32, parallel=4)
```

Run the explainer to generate the global feature importance. Here, we use the test data set. Display the explanation.

```
explanation = gfi.explain(model, X_test, y_test,
                          case_id='CASE_ID', n_iter=10)
explanation
```

Drop the Iris table.

```
oml.drop('Iris')
```

Listing for This Example

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> iris_ds = datasets.load_iris()
>>> iris_data = iris_ds.data.astype(float)
>>> X = pd.DataFrame(iris_data, columns=iris_ds.feature_names)
>>> y = pd.DataFrame(iris_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(iris_data.shape[0]),
...                       columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Iris')
>>>
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                        seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> model = oml.algo.svm(ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
>>> "SVM accuracy score = {:.2f}".format(model.score(X_test, y_test))
'SVM accuracy score = 0.94'
>>>
>>> gfi = GlobalFeatureImportance(mining_function='classification',
...                               score_metric='f1_weighted',
...                               random_state=32, parallel=4)
>>>
>>> explanation = gfi.explain(model, X_test, y_test,
...                           case_id='CASE_ID', n_iter=10)
>>> explanation
Global Feature Importance:
[0] petal length (cm): Value: 0.3462, Error: 0.0824
[1] petal width (cm): Value: 0.2417, Error: 0.0687
[2] sepal width (cm): Value: 0.0926, Error: 0.0452
[3] sepal length (cm): Value: 0.0253, Error: 0.0152
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> oml.drop('Iris')
```

Example 10-6 Regression

This example uses the Boston regression data set. Load the data set into the database, adding a unique case id column.

```
import oml
from oml.mlx import GlobalFeatureImportance
import pandas as pd
import numpy as np
from sklearn import datasets

boston_ds = datasets.load_boston()
boston_data = boston_ds.data
X = pd.DataFrame(boston_data, columns=boston_ds.feature_names)
y = pd.DataFrame(boston_ds.target, columns=['TARGET'])
row_id = pd.DataFrame(np.arange(boston_data.shape[0]),
                      columns=['CASE_ID'])
df = oml.create(pd.concat([X, y, row_id], axis=1), table='Boston')
```

Split the data set into train and test variables.

```
train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID', seed=32)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']
```

Train a Neural Network regression model.

```
model = oml.algo.nn(mining_function='regression',
                    ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
"NN R^2 score = {:.2f}".format(model.score(X_test, y_test))
```

Create the MLX Global Feature Importance explainer, using the `r2` metric.

```
gfi = GlobalFeatureImportance(mining_function='regression',
                              score_metric='r2', random_state=32,
                              parallel=4)
```

Run the explainer to generate the global feature importance. Here, we use the test data set. Display the explanation.

```
explanation = gfi.explain(model, df, 'TARGET',
                          case_id='CASE_ID', n_iter=10)
explanation
```

Drop the Boston table.

```
oml.drop('Boston')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Listing for This Example

```
>>> import oml
>>> from oml.mlx import GlobalFeatureImportance
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> boston_ds = datasets.load_boston()
>>> boston_data = boston_ds.data
>>> X = pd.DataFrame(boston_data, columns=boston_ds.feature_names)
>>> y = pd.DataFrame(boston_ds.target, columns=['TARGET'])
>>> row_id = pd.DataFrame(np.arange(boston_data.shape[0]),
...                       columns=['CASE_ID'])
>>> df = oml.create(pd.concat([X, y, row_id], axis=1), table='Boston')
>>>
>>> train, test = df.split(ratio=(0.8, 0.2), hash_cols='CASE_ID',
...                        seed=32)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> model = oml.algo.nn(mining_function='regression',
..                     ODMS_RANDOM_SEED=32).fit(X, y, case_id='CASE_ID')
>>> "NN R^2 score = {:.2f}".format(model.score(X_test, y_test))
'NN R^2 score = 0.85'
>>>
>>> gfi = GlobalFeatureImportance(mining_function='regression',
...                               score_metric='r2', random_state=32,
...                               parallel=4)
>>>
>>> explanation = gfi.explain(model, df, 'TARGET',
...                           case_id='CASE_ID', n_iter=10)
>>> explanation
Global Feature Importance:
[0] LSTAT: Value: 0.7686, Error: 0.0513
[1] RM: Value: 0.5734, Error: 0.0475
[2] CRIM: Value: 0.5131, Error: 0.0345
[3] DIS: Value: 0.4170, Error: 0.0632
[4] NOX: Value: 0.2592, Error: 0.0206
[5] AGE: Value: 0.2083, Error: 0.0212
[6] RAD: Value: 0.1956, Error: 0.0188
[7] INDUS: Value: 0.1792, Error: 0.0199
[8] B: Value: 0.0982, Error: 0.0146
[9] PTRATIO: Value: 0.0822, Error: 0.0069
[10] TAX: Value: 0.0566, Error: 0.0139
[11] ZN: Value: 0.0397, Error: 0.0081
[12] CHAS: Value: 0.0125, Error: 0.0045

>>> oml.drop('Boston')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

10.7 Attribute Importance

The `oml.ai` class computes the relative attribute importance, which ranks attributes according to their significance in predicting a classification or regression target.

The `oml.ai` class uses the Minimum Description Length (MDL) algorithm to calculate attribute importance. MDL assumes that the simplest, most compact representation of the data is the best and most probable explanation of the data.

You can use methods of the `oml.ai` class to compute the relative importance of predictor variables when predicting a response variable.

> **Note**
>
> Oracle Machine Learning does not support the scoring operation for `oml.ai`.

The results of `oml.ai` are the attributes of the build data ranked according to their predictive influence on a specified target attribute. You can use the ranking and the measure of importance for selecting attributes.

For information on the `oml.ai` class attributes and methods, invoke `help(oml.ai)` or see Oracle Machine Learning for Python API Reference.

> **See Also**
>
> - About Model Settings
> - Shared Settings

Example 10-7 Ranking Attribute Significance with oml.ai

This example creates the `x` and `y` variables using the iris data set. It then creates the persistent database table IRIS and the `oml.DataFrame` object `oml_iris` as a proxy for the table.

This example demonstrates the use of various methods of the `oml.ai` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train _y = dat[0]['Species']
test_dat = dat[1]

# Specify settings.
setting = {'ODMS_SAMPLING':'ODMS_SAMPLING_DISABLE'}

# Create an AI model object.
ai_mod = oml.ai(**setting)

# Fit the AI model according to the training data and parameter
# settings.
ai_mod = ai_mod.fit(train_x, train_y)

# Show the model details.
ai_mod
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> # Specify settings.
... setting = {'ODMS_SAMPLING':'ODMS_SAMPLING_DISABLE'}
>>>
>>> # Create an AI model object.
... ai_mod = oml.ai(**setting)
>>>
>>> # Fit the AI model according to the training data and parameter
... # settings.
>>> ai_mod = ai_mod.fit(train_x, train_y)
>>>
>>> # Show the model details.
... ai_mod

Algorithm Name: Attribute Importance

Mining Function: ATTRIBUTE_IMPORTANCE

Settings:
                   setting name            setting value
0                     ALGO_NAME              ALGO_AI_MDL
1                   ODMS_DETAILS              ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                       ON

Global Statistics:
   attribute name     attribute value
0        NUM_ROWS                 104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Importance:

       variable  importance  rank
0   Petal_Width    0.615851     1
1  Petal_Length    0.362519     2
2  Sepal_Length    0.042751     3
3   Sepal_Width   -0.155867     4
```

10.8 Association Rules

The `oml.ar` class implements the Apriori algorithm to find frequent itemsets and association rules, all as part of an association model object.

The Apriori algorithm is efficient and scales well with respect to the number of transactions, number of items, and number of itemsets and rules produced.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Use the `oml.ar` class to identify frequent itemsets within large volumes of transactional data, such as in market basket analysis. The results of an association model are the rules that identify patterns of association within the data.

An association rule identifies a pattern in the data in which the appearance of a set of items in a transactional record implies another set of items. The groups of items used to form rules must pass a minimum threshold according to how often they occur (the *support* of the rule) and how often the consequent follows the antecedent (the *confidence* of the rule). Association models generate all rules that have support and confidence greater than user-specified thresholds.

Oracle Machine Learning does not support the scoring operation for association modeling.

For information on the `oml.ar` class attributes and methods, invoke `help(oml.ar)` or see Oracle Machine Learning for Python API Reference.

Settings for an Association Rules Model

The following table lists the settings applicable to association rules models.

Table 10-3 Association Rules Models Settings

Setting Name	Setting Value	Description
ASSO_ABS_ERROR	0<ASSO_ABS_ERRORMAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE)	Specifies the absolute error for the association rules sampling. A smaller value of ASSO_ABS_ERROR obtains a larger sample size that gives accurate results but takes longer to compute. Set a reasonable value for ASSO_ABS_ERROR, such as the default value, to avoid too large a sample size. The default value is 0.5 * MAX(ASSO_MIN_SUPPORT, ASSO_MIN_CONFIDENCE).
ASSO_AGGREGATES	NULL	Specifies the columns to aggregate. It is a comma separated list of strings containing the names of the columns for aggregation. The number of columns in the list must be <= 10. You can set ASSO_AGGREGATES if you have specified a column name with ODMS_ITEM_ID_COLUMN_NAME. The data table must have valid column names such as ITEM_ID and CASE_ID which are derived from ODMS_ITEM_ID_COLUMN_NAME. An item value is not mandatory. The default value is NULL. For each item, you may supply several columns to aggregate. However, doing so requires more memory to buffer the extra data and also affects performance because of the larger input data set and increased operations.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-3 (Cont.) Association Rules Models Settings

Setting Name	Setting Value	Description
ASSO_ANT_IN_RULES	NULL	Sets Including Rules for the antecedent: it is a comma separated list of strings, at least one of which must appear in the antecedent part of each reported association rule. The default value is NULL.
ASSO_ANT_EX_RULES	NULL	Sets Excluding Rules for the antecedent: it is a comma separated list of strings, none of which can appear in the antecedent part of each reported association rule. The default value is NULL.
ASSO_CONF_LEVEL	0 ASSO_CONF_LEVEL 1	Specifies the confidence level for an association rules sample. A larger value of ASSO_CONF_LEVEL obtains a larger sample size. Any value between 0.9 and 1 is suitable. The default value is 0.95.
ASSO_CONS_IN_RULES	NULL	Sets Including Rules for the consequent: it is a comma separated list of strings, at least one of which must appear in the consequent part of each reported association rule. The default value is NULL.
ASSO_CONS_EX_RULES	NULL	Sets Excluding Rules for the consequent: it is a comma separated list of strings, none of which can appear in the consequent part of a reported association rule. You can use the excluding rule to reduce the data that must be stored, but you may be required to build extra models for executing different Including or Excluding Rules. The default value is NULL.
ASSO_EX_RULES	NULL	Sets Excluding Rules applied for each association rule: it is a comma separated list of strings that cannot appear in an association rule. No rule can contain any item in the list. The default value is NULL.
ASSO_IN_RULES	NULL	Sets Including Rules applied for each association rule: it is a comma separated list of strings, at least one of which must appear in each reported association rule, either as antecedent or as consequent The default value NULL, which specifies that filtering is not applied.
ASSO_MAX_RULE_LENGTH	TO_CHAR(2<= *numeric_expr* <=20)	Maximum rule length for association rules. The default value is 4.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-3 (Cont.) Association Rules Models Settings

Setting Name	Setting Value	Description
ASSO_MIN_CONFIDENCE	TO_CHAR(0<= *numeric_expr* <=1)	Minimum confidence for association rules. The default value is 0.1.
ASSO_MIN_REV_CONFIDENCE	TO_CHAR(0<= *numeric_expr* <=1)	Sets the Minimum Reverse Confidence that each rule should satisfy. The Reverse Confidence of a rule is defined as the number of transactions in which the rule occurs divided by the number of transactions in which the consequent occurs. The value is real number between 0 and 1. The default value is 0.
ASSO_MIN_SUPPORT	TO_CHAR(0<= *numeric_expr* <=1)	Minimum support for association rules. The default value is 0.1.
ASSO_MIN_SUPPORT_INT	TO_CHAR(0<= *numeric_expr* <=1)	Minimum absolute support that each rule must satisfy. The value must be an integer. The default value is 1.
ASSO_CONS_EX_RULES		
ODMS_ITEM_ID_COLUMN_NAME	*column_name*	The name of a column that contains the items in a transaction. When you specify this setting, the algorithm expects the data to be presented in native transactional format, consisting of two columns: • Case ID, either categorical or numeric • Item ID, either categorical or numeric
ODMS_ITEM_VALUE_COLUMN_ NAME	*column_name*	The name of a column that contains a value associated with each item in a transaction. Use this setting only when you have specified a value for ODMS_ITEM_ID_COLUMN_NAME, indicating that the data is presented in native transactional format. If you also use ASSO_AGGREGATES, then the build data must include the following three columns and the columns specified in the AGGREGATES setting. • Case ID, either categorical or numeric • Item ID, either categorical or numeric, specified by ODMS_ITEM_ID_COLUMN_NAME • Item value, either categorical or numeric, specified by ODMS_ITEM_VALUE_COLUMN_ NAME If ASSO_AGGREGATES, Case ID, and Item ID columns are present, then the Item Value column may or may not appear. The Item Value column may specify information such as the number of items (for example, three apples) or the type of the item (for example, macintosh apples).

Copyright © 2024, 2026, Oracle and/or its affiliates.

See Also

- About Model Settings
- Shared Settings

Example 10-8 Using the oml.ar Class

This example uses methods of the `oml.ar` class.

```
import pandas as pd
from sklearn import datasets
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species']))

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training data.
train_dat = oml.sync(table = 'IRIS')

# Specify settings.
setting = {'asso_min_support':'0.1', 'asso_min_confidence':'0.1'}

# Create an AR model object.
ar_mod = oml.ar(**setting)

# Fit the model according to the training data and parameter
# settings.
ar_mod = ar_mod.fit(train_dat)

# Show details of the model.
ar_mod
```

Listing for This Example

```
>>> import pandas as pd
>>> from sklearn import datasets
>>> import oml
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training data.
... train_dat = oml.sync(table = 'IRIS')
>>>
>>> # Specify settings.
... setting = {'asso_min_support':'0.1', 'asso_min_confidence':'0.1'}
>>>
>>> # Create an AR model object.
... ar_mod = oml.ar(**setting)
>>>
>>> # Fit the model according to the training data and parameter
... # settings.
>>> ar_mod = ar_mod.fit(train_dat)
>>>
>>> # Show details of the model.
... ar_mod

Algorithm Name: Association Rules

Mining Function: ASSOCIATION

Settings:
                   setting name                    setting value
0                     ALGO_NAME  ALGO_APRIORI_ASSOCIATION_RULES
1          ASSO_MAX_RULE_LENGTH                                4
2           ASSO_MIN_CONFIDENCE                              0.1
3       ASSO_MIN_REV_CONFIDENCE                                0
4              ASSO_MIN_SUPPORT                              0.1
5          ASSO_MIN_SUPPORT_INT                                1
6                  ODMS_DETAILS                     ODMS_ENABLE
7  ODMS_MISSING_VALUE_TREATMENT         ODMS_MISSING_VALUE_AUTO
8                 ODMS_SAMPLING           ODMS_SAMPLING_DISABLE
9                     PREP_AUTO                               ON

Global Statistics:
     attribute name  attribute value
0     ITEMSET_COUNT         6.000000
1       MAX_SUPPORT         0.333333
2          NUM_ROWS       150.000000
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
3        RULE_COUNT         2.000000
4 TRANSACTION_COUNT       150.000000

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Itemsets:

   ITEMSET_ID   SUPPORT  NUMBER_OF_ITEMS    ITEM_NAME          ITEM_VALUE
0           1  0.193333                1  Petal_Width  .20000000000000001
1           2  0.173333                1  Sepal_Width                   3
2           3  0.333333                1      Species              setosa
3           4  0.333333                1      Species          versicolor
4           5  0.333333                1      Species           virginica
5           6  0.193333                2  Petal_Width  .20000000000000001
6           6  0.193333                2      Species              setosa

Rules:

   RULE_ID  NUMBER_OF_ITEMS     LHS_NAME           LHS_VALUE     RHS_NAME  \
0        1                2      Species              setosa  Petal_Width
1        2                2  Petal_Width  .20000000000000001      Species

  RHS_VALUE   SUPPORT  CONFIDENCE  REVCONFIDENCE  LIFT
0      None  0.186667        0.58           1.00     3
1      None  0.186667        1.00           0.58     3
```

10.9 Decision Tree

The `oml.dt` class uses the Decision Tree algorithm for classification.

Decision Tree models are classification models that contain axis-parallel rules. A rule is a conditional statement that can be understood by humans and may be used within a database to identify a set of records.

A decision tree predicts a target value by asking a sequence of questions. At a given stage in the sequence, the question that is asked depends upon the answers to the previous questions. The goal is to ask questions that, taken together, uniquely identify specific target values. Graphically, this process forms a tree structure.

During the training process, the Decision Tree algorithm must repeatedly find the most efficient way to split a set of cases (records) into two child nodes. The `oml.dt` class offers two homogeneity metrics, gini and entropy, for calculating the splits. The default metric is gini.

For information on the `oml.dt` class attributes and methods, invoke `help(oml.dt)` or see Oracle Machine Learning for Python API Reference.

Settings for a Decision Tree Model

The following table lists settings that apply to Decision Tree models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-4 Decision Tree Model Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	*table_name*	The name of a table that stores a cost matrix for the algorithm to use in building and applying the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table. • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
CLAS_MAX_SUP_BINS	2 <= *a number* <= 2147483647	Specifies the maximum number of bins for each attribute. The default value is 32.
CLAS_WEIGHTS_BALANCED	ON OFF	Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.
TREE_IMPURITY_METRIC	TREE_IMPURITY_ENTROPY TREE_IMPURITY_GINI	Tree impurity metric for a Decision Tree model. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with a metric. Decision trees can use either gini (TREE_IMPURITY_GINI) or entropy (TREE_IMPURITY_ENTROPY) as the purity metric. By default, the algorithm uses TREE_IMPURITY_GINI.
TREE_TERM_MAX_DEPTH	2 <= *a number* <= 100	Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). The default is 7.
TREE_TERM_MINPCT_NODE	0< = *a number* <= 10	The minimum number of training rows in a node expressed as a percentage of the rows in the training data. The default value is 0.05, indicating 0.05%.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-4 (Cont.) Decision Tree Model Settings

Setting Name	Setting Value	Description
TREE_TERM_MINPCT_SPLIT	0 < *a number* <= 20	Minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. The default value is 0.1, indicating 0.1%.
TREE_TERM_MINREC_NODE	*A number* >= 0	Minimum number of rows in a node. The default value is 10.
TREE_TERM_MINREC_SPLIT	*A number* > 1	Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if the number of records is below this value. The default value is 20.

See Also

- About Model Settings
- Shared Settings

Example 10-9 Using the oml.dt Class

This example demonstrates the use of various methods of the `oml.dt` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('COST_MATRIX')
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create  training and test data.
dat = oml.sync(table = 'IRIS').split()
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a cost matrix table in the database.
cost_matrix = [['setosa', 'setosa', 0],
               ['setosa', 'virginica', 0.2],
               ['setosa', 'versicolor', 0.8],
               ['virginica', 'virginica', 0],
               ['virginica', 'setosa', 0.5],
               ['virginica', 'versicolor', 0.5],
               ['versicolor', 'versicolor', 0],
               ['versicolor', 'setosa', 0.4],
               ['versicolor', 'virginica', 0.6]]
cost_matrix = oml.create(
  pd.DataFrame(cost_matrix,
               columns = ['ACTUAL_TARGET_VALUE',
                          'PREDICTED_TARGET_VALUE', 'COST']),
               table = 'COST_MATRIX')

# Specify settings.
setting = {'TREE_TERM_MAX_DEPTH':'2'}

# Create a DT model object.
dt_mod = oml.dt(**setting)

# Fit the DT model according to the training data and parameter
# settings.
dt_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

# Use the model to make predictions on the test data.
dt_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Petal_Length',
                                                'Species']])

# Return the prediction probability.
dt_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Species']],
               proba = True)

# Make predictions and return the probability for each class
# on new data.
dt_mod.predict_proba(test_dat.drop('Species'),
                     supplemental_cols = test_dat[:,
                       ['Sepal_Length',
                        'Species']]).sort_values(by = ['Sepal_Length',
                                                       'Species'])

dt_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> try:
...    oml.drop('COST_MATRIX')
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # Create a cost matrix table in the database.
... cost_matrix = [['setosa', 'setosa', 0],
...                ['setosa', 'virginica', 0.2],
...                ['setosa', 'versicolor', 0.8],
...                ['virginica', 'virginica', 0],
...                ['virginica', 'setosa', 0.5],
...                ['virginica', 'versicolor', 0.5],
...                ['versicolor', 'versicolor', 0],
...                ['versicolor', 'setosa', 0.4],
...                ['versicolor', 'virginica', 0.6]]
>>> cost_matrix = oml.create(
...  pd.DataFrame(cost_matrix,
...               columns = ['ACTUAL_TARGET_VALUE',
...                          'PREDICTED_TARGET_VALUE',
...                          'COST']),
...               table = 'COST_MATRIX')
>>>
>>> # Specify settings.
... setting = {'TREE_TERM_MAX_DEPTH':'2'}
>>>
>>> # Create a DT model object.
... dt_mod = oml.dt(**setting)
>>>
>>> # Fit the DT model according to the training data and parameter
... # settings.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> dt_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: Species

Settings:
                      setting name            setting value
0                        ALGO_NAME       ALGO_DECISION_TREE
1             CLAS_COST_TABLE_NAME "OML_USER"."COST_MATRIX"
2                CLAS_MAX_SUP_BINS                       32
3            CLAS_WEIGHTS_BALANCED                      OFF
4                     ODMS_DETAILS              ODMS_ENABLE
5    ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
6                    ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
7                        PREP_AUTO                       ON
8             TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
9              TREE_TERM_MAX_DEPTH                        2
10           TREE_TERM_MINPCT_NODE                      .05
11          TREE_TERM_MINPCT_SPLIT                       .1
12           TREE_TERM_MINREC_NODE                       10
13          TREE_TERM_MINREC_SPLIT                       20

Global Statistics:
   attribute name  attribute value
0        NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width

Partition: NO

Distributions:

   NODE_ID TARGET_VALUE  TARGET_COUNT
0        0       setosa            36
1        0   versicolor            35
2        0    virginica            33
3        1       setosa            36
4        2   versicolor            35
5        2    virginica            33

Nodes:

   parent  node.id  row.count  prediction  \
0     0.0        1         36      setosa
1     0.0        2         68  versicolor
2     NaN        0        104      setosa

                                        split  \
0  (Petal_Length <=(2.4500000000000002E+000))
1   (Petal_Length >(2.4500000000000002E+000))
2                                        None
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                                    surrogate  \
0  Petal_Width <=(8.0000000000000004E-001))
1   Petal_Width >(8.0000000000000004E-001))
2                                       None

                                    full.splits
0  (Petal_Length <=(2.4500000000000002E+000))
1   (Petal_Length >(2.4500000000000002E+000))
2                                           (
>>>
>>> # Use the model to make predictions on the test data.
... dt_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
0            4.9          3.0           1.4      setosa      setosa
1            4.9          3.1           1.5      setosa      setosa
2            4.8          3.4           1.6      setosa      setosa
3            5.8          4.0           1.2      setosa      setosa
...          ...          ...           ...         ...         ...
44           6.7          3.3           5.7   virginica  versicolor
45           6.7          3.0           5.2   virginica  versicolor
46           6.5          3.0           5.2   virginica  versicolor
47           5.9          3.0           5.1   virginica  versicolor
>>>
>>> # Return the prediction probability.
... dt_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']],
...                proba = True)
    Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
0            4.9          3.0      setosa      setosa     1.000000
1            4.9          3.1      setosa      setosa     1.000000
2            4.8          3.4      setosa      setosa     1.000000
3            5.8          4.0      setosa      setosa     1.000000
...          ...          ...         ...         ...          ...
44           6.7          3.3   virginica  versicolor     0.514706
45           6.7          3.0   virginica  versicolor     0.514706
46           6.5          3.0   virginica  versicolor     0.514706
47           5.9          3.0   virginica  versicolor     0.514706

>>> # Make predictions and return the probability for each class
>>> # on new data.
>>> dt_mod.predict_proba(test_dat.drop('Species'),
...                      supplemental_cols = test_dat[:,
...                        ['Sepal_Length',
...                         'Species']]).sort_values(by = ['Sepal_Length',
...                                                       'Species'])
    Sepal_Length     Species  PROBABILITY_OF_SETOSA  \
0            4.4      setosa                    1.0
1            4.4      setosa                    1.0
2            4.5      setosa                    1.0
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
3            4.8      setosa                   1.0
...          ...         ...                   ...
42           6.7   virginica                   0.0
43           6.9  versicolor                   0.0
44           6.9   virginica                   0.0
45           7.0  versicolor                   0.0

    PROBABILITY_OF_VERSICOLOR  PROBABILITY_OF_VIRGINICA
0                    0.000000                  0.000000
1                    0.000000                  0.000000
2                    0.000000                  0.000000
3                    0.000000                  0.000000
...                       ...                       ...
42                   0.514706                  0.485294
43                   0.514706                  0.485294
44                   0.514706                  0.485294
45                   0.514706                  0.485294
>>>
>>> dt_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.645833
```

10.10 Expectation Maximization

The `oml.em` class uses the Expectation Maximization (EM) algorithm to create a clustering model.

EM is a density estimation algorithm that performs probabilistic clustering. In density estimation, the goal is to construct a density function that captures how a given population is distributed. The density estimate is based on observed data that represents a sample of the population.

EM is enhanced to resolve some challenges in its standard form. EM is well established as a distribution-based algorithm. The Oracle Machine Learning for SQL implementation includes significant enhancements, such as scalable processing of large volumes of data and automatic parameter initialization. For more information, see Oracle Machine Learning for SQL Concepts Guide..

For information on the `oml.em` class methods, invoke `help(oml.em)` or see Oracle Machine Learning for Python API Reference..

Settings for an Expectation Maximization Model

The following table lists settings for data preparation and analysis for `EM` models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-5 Expectation Maximization Settings for Data Preparation and Analysis

Setting Name	Setting Value	Description
EMCS_ATTRIBUTE_FILTER	EMCS_ATTR_FILTER_ENABLE EMCS_ATTR_FILTER_DISABLE	Whether or not to include uncorrelated attributes in the model. When EMCS_ATTRIBUTE_FILTER is enabled, uncorrelated attributes are not included. **Note** This setting applies only to attributes that are not nested. The default value is system-determined.
EMCS_MAX_NUM_ATTR_2D	TO_CHAR(*numeric_expr* >= 1)	Maximum number of correlated attributes to include in the model. **Note** This setting applies only to attributes that are not nested (2D). The default value is 50.
EMCS_NUM_DISTRIBUTION	EMCS_NUM_DISTR_BERNOULLI EMCS_NUM_DISTR_GAUSSIAN EMCS_NUM_DISTR_SYSTEM	The distribution for modeling numeric attributes. Applies to the input table or view as a whole and does not allow per-attribute specifications. The options include Bernoulli, Gaussian, or system-determined distribution. When Bernoulli or Gaussian distribution is chosen, all numeric attributes are modeled using the same type of distribution. When the distribution is system-determined, individual attributes may use different distributions (either Bernoulli or Gaussian), depending on the data. The default value is EMCS_NUM_DISTR_SYSTEM.
EMCS_NUM_EQUIWIDTH_BINS	TO_CHAR(1 <*numeric_expr* <= 255)	Number of equi-width bins that will be used for gathering cluster statistics for numeric columns. The default value is 11.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-5 (Cont.) Expectation Maximization Settings for Data Preparation and Analysis

Setting Name	Setting Value	Description
EMCS_NUM_PROJECTIONS	TO_CHAR(*numeric_expr* >= 1)	Specifies the number of projections to use for each nested column. If a column has fewer distinct attributes than the specified number of projections, then the data is not projected. The setting applies to all nested columns. The default value is 50.
EMCS_NUM_QUANTILE_BINS	TO_CHAR(1 < *numeric_expr* <= 255)	Specifies the number of quantile bins to use for modeling numeric columns with multivalued Bernoulli distributions. The default value is system-determined.
EMCS_NUM_TOPN_BINS	TO_CHAR(1 < *numeric_expr* <= 255)	Specifies the number of top-N bins to use for modeling categorical columns with multivalued Bernoulli distributions. The default value is system-determined.
EMCS_OUTLIER_RATE **Note** Available only in Oracle AI Database 26ai.	TO_CHAR(0 < numeric_expr < 1)	The desired rate of outliers in the training data. The setting can be used only for EM Anomaly. Default is 0.05.

The following table lists settings for learning for EM models.

Table 10-6 Expectation Maximization Settings for Learning

Setting Name	Setting Value	Description
EMCS_CONVERGENCE_CRITERION	EMCS_CONV_CRIT_HELDASIDE EMCS_CONV_CRIT_BIC	The convergence criterion for EM. The convergence criterion may be based on a held-aside data set or it may be Bayesian Information Criterion. The default value is system determined.
EMCS_LOGLIKE_IMPROVEMENT	TO_CHAR(0 < *numeric_expr* < 1)	When the convergence criterion is based on a held-aside data set (EMCS_CONVERGENCE_CRITERION = EMCS_CONV_CRIT_HELDASIDE), this setting specifies the percentage improvement in the value of the log likelihood function that is required for adding a new component to the model.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-6 (Cont.) Expectation Maximization Settings for Learning

Setting Name	Setting Value	Description
EMCS_MODEL_SEARCH	EMCS_MODEL_SEARCH_ENABLE EMCS_MODEL_SEARCH_DISABLE	Enables model search in EM where different model sizes are explored and the best size is selected. The default value is EMCS_MODEL_SEARCH_DISABLE.
EMCS_NUM_COMPONENTS	TO_CHAR(*numeric_expr* >= 1)	Maximum number of components in the model. If model search is enabled, the algorithm automatically determines the number of components based on improvements in the likelihood function or based on regularization, up to the specified maximum. The number of components must be greater than or equal to the number of clusters. The default value is 20.
EMCS_NUM_ITERATIONS	TO_CHAR(*numeric_expr* >= 1)	Specifies the maximum number of iterations in the EM algorithm. The default value is 100.
EMCS_RANDOM_SEED	Non-negative integer	Controls the seed of the random generator used in EM. The default value is 0.
EMCS_REMOVE_COMPONENTS	EMCS_REMOVE_COMPS_ENABLE EMCS_REMOVE_COMPS_DISABLE	Allows the EM algorithm to remove a small component from the solution. The default value is EMCS_REMOVE_COMPS_ENABLE.

The following table lists the settings for component clustering for EM models.

Table 10-7 Expectation Maximization Settings for Component Clustering

Setting Name	Setting Value	Description
CLUS_NUM_CLUSTERS	*TO_CHAR(numeric_expr* >= 1)	The maximum number of leaf clusters generated by the algorithm. The algorithm may return fewer clusters than the specified number, depending on the data. but it cannot return more clusters than the number of components, which is governed by algorithm-specific settings. (See Table 10-6.) Depending on these settings, there may be fewer clusters than components. If component clustering is disabled, then the number of clusters equals the number of components. The default value is system-determined.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-7 (Cont.) Expectation Maximization Settings for Component Clustering

Setting Name	Setting Value	Description
EMCS_CLUSTER_COMPONENTS	EMCS_CLUSTER_COMP_ENABLE EMCS_CLUSTER_COMP_DISABLE	Enables or disables the grouping of EM components into high-level clusters. When disabled, the components themselves are treated as clusters. When component clustering is enabled, model scoring through the SQL CLUSTER function produces assignments to the higher level clusters. When clustering is disabled, the CLUSTER function produces assignments to the original components. The default value is EMCS_CLUSTER_COMP_ENABLE.
EMCS_CLUSTER_THRESH	TO_CHAR(*numeric_expr* >= 1)	Dissimilarity threshold that controls the clustering of EM components. When the dissimilarity measure is less than the threshold, the components are combined into a single cluster. A lower threshold may produce more clusters that are more compact. A higher threshold may produce fewer clusters that are more spread out. The default value is 2.
EMCS_LINKAGE_FUNCTION	EMCS_LINKAGE_SINGLE EMCS_LINKAGE_AVERAGE EMCS_LINKAGE_COMPLETE	Allows the specification of a linkage function for the agglomerative clustering step. EMCS_LINKAGE_SINGLE uses the nearest distance within the branch. The clusters tend to be larger and have arbitrary shapes. EMCS_LINKAGE_AVERAGE uses the average distance within the branch. There is less chaining effect and the clusters are more compact. EMCS_LINKAGE_COMPLETE uses the maximum distance within the branch. The clusters are smaller and require strong component overlap. The default value is EMCS_LINKAGE_SINGLE.

The following table lists the settings for cluster statistics for EM models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-8 Expectation Maximization Settings for Cluster Statistics

Setting Name	Setting Value	Description
EMCS_CLUSTER_STATISTICS	EMCS_CLUS_STATS_ENABLE EMCS_CLUS_STATS_DISABLE	Enables or disables the gathering of descriptive statistics for clusters (centroids, histograms, and rules). When statistics are disabled, model size is reduced. The default value is EMCS_CLUS_STATS_ENABLE.
EMCS_MIN_PCT_ATTR_SUPPORT	TO_CHAR(0 < *numeric_expr* < 1)	Minimum support required for including an attribute in the cluster rule. The support is the percentage of the data rows assigned to a cluster that must have non-null values for the attribute. The default value is 0.1.

See Also

- About Model Settings
- Shared Settings

Example 10-10 Using the oml.em Class

This example creates an EM model and uses some of the methods of the `oml.em` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
test_dat = dat[1]

# Specify settings.
setting = {'emcs_num_iterations': 100}

# Create an EM model object
em_mod = oml.em(n_clusters = 2, **setting)

# Fit the EM model according to the training data and parameter
# settings.
em_mod = em_mod.fit(train_dat)

# Show details of the model.
em_mod

# Use the model to make predictions on the test data.
em_mod.predict(test_dat)

# Make predictions and return the probability for each class
# on new data.
em_mod.predict_proba(test_dat,
  supplemental_cols = test_dat[:,
    ['Sepal_Length', 'Sepal_Width',
     'Petal_Length']]).sort_values(by = ['Sepal_Length',
       'Sepal_Width',  'Petal_Length',
       'PROBABILITY_OF_2', 'PROBABILITY_OF_3'])

# Change the random seed and refit the model.
em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... setting = {'emcs_num_iterations': 100}
>>>
>>> # Create an EM model object.
... em_mod = oml.em(n_clusters = 2, **setting)
>>>
>>> # Fit the EM model according to the training data and parameter
... # settings.
>>> em_mod = em_mod.fit(train_dat)
>>>
>>> # Show details of the model.
... em_mod

Algorithm Name: Expectation Maximization

Mining Function: CLUSTERING

Settings:
                     setting name                   setting value
0                       ALGO_NAME  ALGO_EXPECTATION_MAXIMIZATION
1             CLUS_NUM_CLUSTERS                               2
2        EMCS_CLUSTER_COMPONENTS       EMCS_CLUSTER_COMP_ENABLE
3        EMCS_CLUSTER_STATISTICS         EMCS_CLUS_STATS_ENABLE
4            EMCS_CLUSTER_THRESH                               2
5          EMCS_LINKAGE_FUNCTION            EMCS_LINKAGE_SINGLE
6       EMCS_LOGLIKE_IMPROVEMENT                            .001
7           EMCS_MAX_NUM_ATTR_2D                              50
8      EMCS_MIN_PCT_ATTR_SUPPORT                              .1
9               EMCS_MODEL_SEARCH      EMCS_MODEL_SEARCH_DISABLE
10           EMCS_NUM_COMPONENTS                              20
11         EMCS_NUM_DISTRIBUTION          EMCS_NUM_DISTR_SYSTEM
12       EMCS_NUM_EQUIWIDTH_BINS                              11
13           EMCS_NUM_ITERATIONS                             100
14          EMCS_NUM_PROJECTIONS                              50
15              EMCS_RANDOM_SEED                               0
16        EMCS_REMOVE_COMPONENTS       EMCS_REMOVE_COMPS_ENABLE
17                   ODMS_DETAILS                   ODMS_ENABLE
18  ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
19                 ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
20                     PREP_AUTO                              ON

Computed Settings:
                setting name             setting value
0       EMCS_ATTRIBUTE_FILTER  EMCS_ATTR_FILTER_DISABLE
1  EMCS_CONVERGENCE_CRITERION        EMCS_CONV_CRIT_BIC
2      EMCS_NUM_QUANTILE_BINS                         3
3          EMCS_NUM_TOPN_BINS                         3

Global Statistics:
       attribute name  attribute value
0           CONVERGED              YES
1       LOGLIKELIHOOD        -2.10044
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
2        NUM_CLUSTERS                2
3      NUM_COMPONENTS                8
4            NUM_ROWS              104
5         RANDOM_SEED                0
6  REMOVED_COMPONENTS               12

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

   CLUSTER_ID  CLUSTER_NAME  RECORD_COUNT  PARENT  TREE_LEVEL \
0           1             1           104     NaN           1
1           2             2            68     1.0           2
2           3             3            36     1.0           2
  LEFT_CHILD_ID  RIGHT_CHILD_ID
0           2.0             3.0
1           NaN             NaN
2           NaN             NaN

Taxonomy:

   PARENT_CLUSTER_ID  CHILD_CLUSTER_ID
0                  1               2.0
1                  1               3.0
2                  2               NaN
3                  3               NaN

Centroids:

    CLUSTER_ID ATTRIBUTE_NAME      MEAN  MODE_VALUE  VARIANCE
0            1   Petal_Length  3.721154        None  3.234694
1            1    Petal_Width  1.155769        None  0.567539
2            1   Sepal_Length  5.831731        None  0.753255
3            1    Sepal_Width  3.074038        None  0.221358
4            1        Species       NaN      setosa       NaN
5            2   Petal_Length  4.902941        None  0.860588
6            2    Petal_Width  1.635294        None  0.191572
7            2   Sepal_Length  6.266176        None  0.545555
8            2    Sepal_Width  2.854412        None  0.128786
9            2        Species       NaN  versicolor       NaN
10           3   Petal_Length  1.488889        None  0.033016
11           3    Petal_Width  0.250000        None  0.012857
12           3   Sepal_Length  5.011111        None  0.113016
13           3    Sepal_Width  3.488889        None  0.134159
14           3        Species       NaN      setosa       NaN

Leaf Cluster Counts:

   CLUSTER_ID  CNT
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
0           2   68
1           3   36

Attribute Importance:

  ATTRIBUTE_NAME  ATTRIBUTE_IMPORTANCE_VALUE  ATTRIBUTE_RANK
0   Petal_Length                    0.558311               2
1    Petal_Width                    0.556300               3
2   Sepal_Length                    0.469978               4
3    Sepal_Width                    0.196211               5
4        Species                    0.612463               1

Components:

   COMPONENT_ID  CLUSTER_ID  PRIOR_PROBABILITY
0             1           2           0.115366
1             2           2           0.079158
2             3           3           0.113448
3             4           2           0.148059
4             5           3           0.126979
5             6           2           0.134402
6             7           3           0.105727
7             8           2           0.176860

Cluster Hists:

     cluster.id            variable  bin.id  lower.bound  upper.bound  \
0             1       Petal_Length       1         1.00         1.59
1             1       Petal_Length       2         1.59         2.18
2             1       Petal_Length       3         2.18         2.77
3             1       Petal_Length       4         2.77         3.36
...         ...                ...     ...          ...          ...
137           3        Sepal_Width      11          NaN          NaN
138           3    Species:'Other'       1          NaN          NaN
139           3     Species:setosa       2          NaN          NaN
140           3  Species:versicolor       3          NaN          NaN

         label  count
0       1:1.59     25
1    1.59:2.18     11
2    2.18:2.77      0
3    2.77:3.36      3
...        ...    ...
137          :      0
138          :      0
139          :     36
140          :      0

[141 rows x 7 columns]

Rules:

    cluster.id  rhs.support  rhs.conf  lhr.support  lhs.conf       lhs.var  \
0            1          104  1.000000           93  0.892157   Sepal_Width
1            1          104  1.000000           93  0.892157   Sepal_Width
2            1          104  1.000000           99  0.892157  Petal_Length
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
3            1          104  1.000000           99  0.892157  Petal_Length
...        ...          ...       ...          ...       ...           ...
26           3           36  0.346154           36  0.972222  Petal_Length
27           3           36  0.346154           36  0.972222  Sepal_Length
28           3           36  0.346154           36  0.972222  Sepal_Length
29           3           36  0.346154           36  0.972222       Species

    lhs.var.support  lhs.var.conf             predicate
0                93      0.400000    Sepal_Width <= 3.92
1                93      0.400000     Sepal_Width > 2.48
2                93      0.222222   Petal_Length <= 6.31
3                93      0.222222      Petal_Length >= 1
...             ...           ...                    ...
26               35      0.134398      Petal_Length >= 1
27               35      0.094194    Sepal_Length <= 5.74
28               35      0.094194     Sepal_Length >= 4.3
29               35      0.281684        Species = setosa

[30 rows x 9 columns]

>>> # Use the model to make predictions on the test data.
... em_mod.predict(test_dat)
    CLUSTER_ID
0            3
1            3
2            3
3            3
...        ...
42           2
43           2
44           2
45           2

>>> # Make predictions and return the probability for each class
... # on new data.
>>> em_mod.predict_proba(test_dat,
...   supplemental_cols = test_dat[:,
...     ['Sepal_Length', 'Sepal_Width',
...      'Petal_Length']]).sort_values(by = ['Sepal_Length',
...        'Sepal_Width',  'Petal_Length',
...        'PROBABILITY_OF_2', 'PROBABILITY_OF_3'])
    Sepal_Length  Sepal_Width  Petal_Length  PROBABILITY_OF_2  \
0            4.4          3.0           1.3      4.680788e-20
1            4.4          3.2           1.3      1.052071e-20
2            4.5          2.3           1.3      7.751240e-06
3            4.8          3.4           1.6      5.363418e-19
...          ...          ...           ...               ...
43           6.9          3.1           4.9      1.000000e+00
44           6.9          3.1           5.4      1.000000e+00
45           7.0          3.2           4.7      1.000000e+00

    PROBABILITY_OF_3
0       1.000000e+00
1       1.000000e+00
2       9.999922e-01
3       1.000000e+00
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...              ...
43     3.295578e-97
44    6.438740e-137
45     3.853925e-89

>>>
>>> # Change the random seed and refit the model.
... em_mod.set_params(EMCS_RANDOM_SEED = '5').fit(train_dat)

Algorithm Name: Expectation Maximization

Mining Function: CLUSTERING

Settings:
                    setting name                    setting value
0                      ALGO_NAME  ALGO_EXPECTATION_MAXIMIZATION
1              CLUS_NUM_CLUSTERS                               2
2        EMCS_CLUSTER_COMPONENTS       EMCS_CLUSTER_COMP_ENABLE
3        EMCS_CLUSTER_STATISTICS         EMCS_CLUS_STATS_ENABLE
4            EMCS_CLUSTER_THRESH                               2
5          EMCS_LINKAGE_FUNCTION            EMCS_LINKAGE_SINGLE
6       EMCS_LOGLIKE_IMPROVEMENT                            .001
7           EMCS_MAX_NUM_ATTR_2D                              50
8      EMCS_MIN_PCT_ATTR_SUPPORT                              .1
9               EMCS_MODEL_SEARCH      EMCS_MODEL_SEARCH_DISABLE
10           EMCS_NUM_COMPONENTS                              20
11         EMCS_NUM_DISTRIBUTION          EMCS_NUM_DISTR_SYSTEM
12       EMCS_NUM_EQUIWIDTH_BINS                              11
13           EMCS_NUM_ITERATIONS                             100
14          EMCS_NUM_PROJECTIONS                              50
15              EMCS_RANDOM_SEED                               5
16        EMCS_REMOVE_COMPONENTS       EMCS_REMOVE_COMPS_ENABLE
17                   ODMS_DETAILS                    ODMS_ENABLE
18  ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
19                 ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
20                     PREP_AUTO                              ON

Computed Settings:
                 setting name             setting value
0       EMCS_ATTRIBUTE_FILTER  EMCS_ATTR_FILTER_DISABLE
1  EMCS_CONVERGENCE_CRITERION        EMCS_CONV_CRIT_BIC
2      EMCS_NUM_QUANTILE_BINS                         3
3          EMCS_NUM_TOPN_BINS                         3

Global Statistics:
       attribute name  attribute value
0           CONVERGED             YES
1       LOGLIKELIHOOD        -1.75777
2        NUM_CLUSTERS               2
3      NUM_COMPONENTS               9
4            NUM_ROWS             104
5         RANDOM_SEED               5
6  REMOVED_COMPONENTS              11

Attributes:
Petal_Length
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

   CLUSTER_ID  CLUSTER_NAME  RECORD_COUNT  PARENT  TREE_LEVEL  LEFT_CHILD_ID
\
0           1             1           104     NaN           1
2.0
1           2             2            36     1.0           2
NaN
2           3             3            68     1.0           2
NaN

   RIGHT_CHILD_ID
0             3.0
1             NaN
2             NaN

Taxonomy:

   PARENT_CLUSTER_ID  CHILD_CLUSTER_ID
0                  1               2.0
1                  1               3.0
2                  2               NaN
3                  3               NaN

Centroids:

    CLUSTER_ID ATTRIBUTE_NAME      MEAN  MODE_VALUE  VARIANCE
0            1   Petal_Length  3.721154        None  3.234694
1            1    Petal_Width  1.155769        None  0.567539
2            1   Sepal_Length  5.831731        None  0.753255
3            1    Sepal_Width  3.074038        None  0.221358
4            1        Species       NaN      setosa       NaN
5            2   Petal_Length  1.488889        None  0.033016
6            2    Petal_Width  0.250000        None  0.012857
7            2   Sepal_Length  5.011111        None  0.113016
8            2    Sepal_Width  3.488889        None  0.134159
9            2        Species       NaN      setosa       NaN
10           3   Petal_Length  4.902941        None  0.860588
11           3    Petal_Width  1.635294        None  0.191572
12           3   Sepal_Length  6.266176        None  0.545555
13           3    Sepal_Width  2.854412        None  0.128786
14           3        Species       NaN  versicolor       NaN

Leaf Cluster Counts:

   CLUSTER_ID  CNT
0           2   36
1           3   68
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Attribute Importance:

  ATTRIBUTE_NAME  ATTRIBUTE_IMPORTANCE_VALUE  ATTRIBUTE_RANK
0   Petal_Length                   0.558311               2
1    Petal_Width                   0.556300               3
2   Sepal_Length                   0.469978               4
3    Sepal_Width                   0.196211               5
4        Species                   0.612463               1

Components:

   COMPONENT_ID  CLUSTER_ID  PRIOR_PROBABILITY
0             1           2           0.113452
1             2           2           0.105727
2             3           3           0.114202
3             4           3           0.086285
4             5           3           0.067294
5             6           2           0.124365
6             7           3           0.126975
7             8           3           0.105761
8             9           3           0.155939

Cluster Hists:

     cluster.id            variable  bin.id  lower.bound  upper.bound  \
0             1       Petal_Length       1         1.00         1.59
1             1       Petal_Length       2         1.59         2.18
2             1       Petal_Length       3         2.18         2.77
3             1       Petal_Length       4         2.77         3.36
...         ...                ...     ...          ...          ...
137           3        Sepal_Width      11          NaN          NaN
138           3    Species:'Other'       1          NaN          NaN
139           3     Species:setosa       3          NaN          NaN
140           3  Species:versicolor       2          NaN          NaN

         label  count
0       1:1.59     25
1    1.59:2.18     11
2    2.18:2.77      0
3    2.77:3.36      3
...        ...    ...
137          :      0
138          :     33
139          :      0
140          :     35

[141 rows x 7 columns]

Rules:

    cluster.id  rhs.support  rhs.conf  lhr.support  lhs.conf       lhs.var  \
0            1          104  1.000000           93  0.894231   Sepal_Width
1            1          104  1.000000           93  0.894231   Sepal_Width
2            1          104  1.000000           99  0.894231  Petal_Length
3            1          104  1.000000           99  0.894231  Petal_Length
...        ...          ...       ...          ...       ...           ...
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
26           3           68  0.653846           68  0.955882  Sepal_Length
27           3           68  0.653846           68  0.955882  Sepal_Length
28           3           68  0.653846           68  0.955882       Species
29           3           68  0.653846           68  0.955882       Species

    lhs.var.support  lhs.var.conf              predicate
0                93      0.400000    Sepal_Width <= 3.92
1                93      0.400000     Sepal_Width > 2.48
2                93      0.222222   Petal_Length <= 6.31
3                93      0.222222      Petal_Length >= 1
...             ...           ...                    ...
26               65      0.026013    Sepal_Length <= 7.9
27               65      0.026013    Sepal_Length > 4.66
28               65      0.125809     Species IN 'Other'
29               65      0.125809  Species IN versicolor
```

10.11 Explicit Semantic Analysis

The `oml.esa` class extracts text-based features from a corpus of documents and performs document similarity comparisons.

Explicit Semantic Analysis (ESA) is an unsupervised algorithm for feature extraction. ESA does not discover latent features but instead uses explicit features based on an existing knowledge base.

Explicit knowledge often exists in text form. Multiple knowledge bases are available as collections of text documents. These knowledge bases can be generic, such as Wikipedia, or domain-specific. Data preparation transforms the text into vectors that capture attribute-concept associations.

ESA uses concepts of an existing knowledge base as features rather than latent features derived by latent semantic analysis methods such as Singular Value Decomposition and Latent Dirichlet Allocation. Each row, for example, in a document in the training data maps to a feature, that is, a concept. ESA has multiple applications in the area of text processing, most notably semantic relatedness (similarity) and explicit topic modeling. Text similarity use cases might involve, for example, resume matching, searching for similar blog postings, and so on.

While projecting a document to the ESA topic space produces a high-dimensional sparse vector, it is unsuitable as an input to other machine learning algorithms. Starting from Oracle AI Database 26ai, embeddings are added to address this issue. For more information about the embeddings, see Oracle Machine Learning for SQL Concepts Guide.

For information on the `oml.esa` class attributes and methods, invoke `help(oml.esa)` or see Oracle Machine Learning for Python API Reference.

Settings for an Explicit Semantic Analysis Model

The following table lists settings for ESA models.

Table 10-9 Explicit Semantic Analysis Settings

Setting Name	Setting Value	Description
`ESAS_MIN_ITEMS`	A non-negative number	Determines the minimum number of non-zero entries required in an input row. The default value is 100 for text input and 0 for non-text input.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-9 (Cont.) Explicit Semantic Analysis Settings

Setting Name	Setting Value	Description
ESAS_TOPN_FEATURES	A positive integer	Controls the maximum number of features per attribute. The default value is 1000.
ESAS_VALUE_THRESHOLD	A non-negative number	Sets the threshold to a small value for attribute weights in the transformed build data. The default value is 1e-8.
FEAT_NUM_FEATURES	TO_CHAR(*numeric_expr* >=1)	The number of features to extract. The default value is estimated by the algorithm. If the matrix rank is smaller than this number, then fewer features are returned.
ESAS_EMBEDDINGS **Note** Available only in Oracle AI Database 26ai.	ESAS_EMBEDDINGS_ENABLE ESAS_EMBEDDINGS_DISABLE	This setting applies to feature extraction models. The default value is ESAS_EMBEDDINGS_DISABLE. When you set ESAS_EMBEDDINGS_ENABLE: • ESA generates embeddings during scoring • The FEATURE_ID of the generated embeddings is of the datatype NUMBER • The CASE_ID_COLUMN_NAME argument of the DBMS_DATA_MINING.CREATE_MODEL and DBMS_DATA_MINING.CREATE_MODEL2 function is optional.
ESAS_EMBEDDING_SIZE **Note** Available only in Oracle AI Database 26ai.	A positive integer less than or equal to 4096	This setting applies to feature extraction models. It specifies the size of the vectors representing embeddings. You can set this parameter only if you have enabled ESAS_EMBEDDINGS. The default size is 1024. If this value is less than the number of distinct features in the training set, then the actual number of explicit features is used as the size of embedding vectors instead.

See Also

- About Model Settings
- Shared Settings

Example 10-11 Using the oml.esa Class

This example creates an ESA model and uses some of the methods of the `oml.esa` class.

```
import oml
from oml import cursor
import pandas as pd
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Create training data and test data.
dat = oml.push(pd.DataFrame(
  {'COMMENTS':['Aids in Africa: Planning for a long war',
     'Mars rover maneuvers for rim shot',
     'Mars express confirms presence of water at Mars south pole',
     'NASA announces major Mars rover finding',
     'Drug access, Asia threat in focus at AIDS summit',
     'NASA Mars Odyssey THEMIS image: typical crater',
     'Road blocks for Aids'],
     'YEAR':['2017', '2018', '2017', '2017', '2018', '2018', '2018'],
     'ID':[1,2,3,4,5,6,7]})).split(ratio=(0.7,0.3), seed = 1234)
train_dat = dat[0]
test_dat = dat[1]

# Specify settings.
cur = cursor()
cur.execute("Begin ctx_ddl.create_policy('DMDEMO_ESA_POLICY'); End;")
cur.close()

odm_settings = {'odms_text_policy_name': 'DMDEMO_ESA_POLICY',
                '"ODMS_TEXT_MIN_DOCUMENTS"': 1,
                '"ESAS_MIN_ITEMS"': 1}

ctx_settings = {'COMMENTS':
                'TEXT(POLICY_NAME:DMDEMO_ESA_POLICY)(TOKEN_TYPE:STEM)'}

# Create an oml ESA model object.
esa_mod = oml.esa(**odm_settings)

# Fit the ESA model according to the training data and parameter settings.
esa_mod = esa_mod.fit(train_dat, case_id = 'ID',
                      ctx_settings = ctx_settings)

# Show model details.
esa_mod

# Use the model to make predictions on test data.
esa_mod.predict(test_dat,
                supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])

esa_mod.transform(test_dat,
  supplemental_cols = test_dat[:, ['ID', 'COMMENTS']],
                                topN = 2).sort_values(by = ['ID'])

esa_mod.feature_compare(test_dat,
                        compare_cols = 'COMMENTS',
                        supplemental_cols = ['ID'])

esa_mod.feature_compare(test_dat,
                        compare_cols = ['COMMENTS', 'YEAR'],
                        supplemental_cols = ['ID'])

# Change the setting parameter and refit the model.
new_setting = {'ESAS_VALUE_THRESHOLD': '0.01',
               'ODMS_TEXT_MAX_FEATURES': '2',
               'ESAS_TOPN_FEATURES': '2'}
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
esa_mod.set_params(**new_setting).fit(train_dat, 'ID', case_id = 'ID',
                   ctx_settings = ctx_settings)

cur = cursor()
cur.execute("Begin ctx_ddl.drop_policy('DMDEMO_ESA_POLICY'); End;")
cur.close()
```

Listing for This Example

```
>>> import oml
>>> from oml import cursor
>>> import pandas as pd
>>>
>>> # Create training data and test data.
... dat = oml.push(pd.DataFrame(
...   {'COMMENTS':['Aids in Africa: Planning for a long war',
...      'Mars rover maneuvers for rim shot',
...      'Mars express confirms presence of water at Mars south pole',
...      'NASA announces major Mars rover finding',
...      'Drug access, Asia threat in focus at AIDS summit',
...      'NASA Mars Odyssey THEMIS image: typical crater',
...      'Road blocks for Aids'],
...      'YEAR':['2017', '2018', '2017', '2017', '2018', '2018', '2018'],
...      'ID':[1,2,3,4,5,6,7]})).split(ratio=(0.7,0.3), seed = 1234)
>>> train_dat = dat[0]
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... cur = cursor()
>>> cur.execute("Begin ctx_ddl.create_policy('DMDEMO_ESA_POLICY'); End;")
>>> cur.close()
>>>
>>> odm_settings = {'odms_text_policy_name': 'DMDEMO_ESA_POLICY',
...                 '"ODMS_TEXT_MIN_DOCUMENTS"': 1,
...                 '"ESAS_MIN_ITEMS"': 1}
>>>
>>> ctx_settings = {'COMMENTS':
...                 'TEXT(POLICY_NAME:DMDEMO_ESA_POLICY)(TOKEN_TYPE:STEM)'}
>>>
>>> # Create an oml ESA model object.
... esa_mod = oml.esa(**odm_settings)
>>>
>>> # Fit the ESA model according to the training data and parameter settings.
... esa_mod = esa_mod.fit(train_dat, case_id = 'ID',
...                       ctx_settings =  ctx_settings)
>>>
>>> # Show model details.
... esa_mod

Algorithm Name: Explicit Semantic Analysis

Mining Function: FEATURE_EXTRACTION

Settings:
                    setting name                 setting value
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
0                     ALGO_NAME  ALGO_EXPLICIT_SEMANTIC_ANALYS
1                 ESAS_MIN_ITEMS                             1
2              ESAS_TOPN_FEATURES                          1000
3            ESAS_VALUE_THRESHOLD                     .00000001
4                    ODMS_DETAILS                   ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
6                   ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
7         ODMS_TEXT_MAX_FEATURES                        300000
8        ODMS_TEXT_MIN_DOCUMENTS                             1
9          ODMS_TEXT_POLICY_NAME              DMDEMO_ESA_POLICY
10                    PREP_AUTO                            ON

Global Statistics:
   attribute name  attribute value
0        NUM_ROWS                4

Attributes:
COMMENTS
YEAR

Partition: NO

Features:

     FEATURE_ID      ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
 0            1     COMMENTS.AFRICA            None     0.342997
 1            1       COMMENTS.AIDS            None     0.171499
 2            1       COMMENTS.LONG            None     0.342997
 3            1   COMMENTS.PLANNING            None     0.342997
...         ...                 ...             ...          ...
 24           6    COMMENTS.ODYSSEY            None     0.282843
 25           6     COMMENTS.THEMIS            None     0.282843
 26           6    COMMENTS.TYPICAL            None     0.282843
 27           6                YEAR            2018     0.707107



>>> # Use the model to make predictions on test data.
... esa_mod.predict(test_dat,
...                 supplemental_cols = test_dat[:, ['ID', 'COMMENTS']])
   ID                                          COMMENTS  FEATURE_ID
0   4           NASA announces major Mars rover finding           3
1   6    NASA Mars Odyssey THEMIS image: typical crater           2
2   7                             Road blocks for Aids           5
>>>
>>> esa_mod.transform(test_dat,
...   supplemental_cols = test_dat[:, ['ID', 'COMMENTS']],
...                                 topN = 2).sort_values(by = ['ID'])
                                          COMMENTS  TOP_1  TOP_1_VAL  \
 0   4           NASA announces major Mars rover finding      3   0.647065
 1   6    NASA Mars Odyssey THEMIS image: typical crater      2   0.766237
 2   7                             Road blocks for Aids      5   0.759125

    TOP_2  TOP_2_VAL
 0      1   0.590565
 1      2   0.616672
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
 2      2   0.632604
>>>
>>> esa_mod.feature_compare(test_dat,
                            compare_cols = 'COMMENTS',
                            supplemental_cols = ['ID'])
   ID_A  ID_B  SIMILARITY
0     4     6    0.946469
1     4     7    0.871994
2     6     7    0.954565

>>> esa_mod.feature_compare(test_dat,
...                         compare_cols = ['COMMENTS', 'YEAR'],
...                         supplemental_cols = ['ID'])
    ID_A  ID_B  SIMILARITY
 0     4     6    0.467644
 1     4     7    0.377144
 2     6     7    O.952857

>>> # Change the setting parameter and refit the model.
... new_setting = {'ESAS_VALUE_THRESHOLD': '0.01',
...                'ODMS_TEXT_MAX_FEATURES': '2',
...                'ESAS_TOPN_FEATURES': '2'}
>>> esa_mod.set_params(**new_setting).fit(train_dat, case_id = 'ID',
...                    ctx_settings = ctx_settings)

Algorithm Name: Explicit Semantic Analysis

Mining Function: FEATURE_EXTRACTION

Settings:
                    setting name                    setting value
0                      ALGO_NAME  ALGO_EXPLICIT_SEMANTIC_ANALYS
1                ESAS_MIN_ITEMS                               1
2             ESAS_TOPN_FEATURES                               2
3           ESAS_VALUE_THRESHOLD                            0.01
4                   ODMS_DETAILS                     ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
6                  ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
7         ODMS_TEXT_MAX_FEATURES                               2
8        ODMS_TEXT_MIN_DOCUMENTS                               1
9          ODMS_TEXT_POLICY_NAME              DMDEMO_ESA_POLICY
10                     PREP_AUTO                              ON

Global Statistics:
   attribute name  attribute value
0        NUM_ROWS                4

Attributes:
COMMENTS
YEAR

Partition: NO

Features:

   FEATURE_ID  ATTRIBUTE_NAME  ATTRIBUTE_VALUE  COEFFICIENT
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
0           1   COMMENTS.AIDS             None      0.707107
1           1       YEAR                  2017      0.707107
2           2   COMMENTS.MARS             None      0.707107
3           2       YEAR                  2018      0.707107
4           3   COMMENTS.MARS             None      0.707107
5           3       YEAR                  2017      0.707107
6           5   COMMENTS.AIDS             None      0.707107
7           5       YEAR                  2018      0.707107

>>>
>>> cur = cursor()
>>> cur.execute("Begin ctx_ddl.drop_policy('DMDEMO_ESA_POLICY'); End;")
>>> cur.close()
```

10.12 Generalized Linear Model

The `oml.glm` class builds a Generalized Linear Model (GLM) model.

GLM models include and extend the class of linear models. They relax the restrictions on linear models, which are often violated in practice. For example, binary (yes/no or 0/1) responses do not have the same variance across classes.

GLM is a parametric modeling technique. Parametric models make assumptions about the distribution of the data. When the assumptions are met, parametric models can be more efficient than non-parametric models.

The challenge in developing models of this type involves assessing the extent to which the assumptions are met. For this reason, quality diagnostics are key to developing quality parametric models.

In addition to the classical weighted least squares estimation for linear regression and iteratively re-weighted least squares estimation for logistic regression, both solved through Cholesky decomposition and matrix inversion, Oracle Machine Learning GLM provides a conjugate gradient-based optimization algorithm that does not require matrix inversion and is very well suited to high-dimensional data. The choice of algorithm is handled internally and is transparent to the user.

GLM can be used to build classification or regression models as follows:

- **Classification**: Binary logistic regression is the GLM classification algorithm. The algorithm uses the logit link function and the binomial variance function.
- **Regression:** Linear regression is the GLM regression algorithm. The algorithm assumes no target transformation and constant variance over the range of target values.

The following table provides the link functions used in GLM:

GLM Function	Default Link Function	Other Supported Link Functions
Linear regression (gaussian)	identity	none
Logistic regression (binomial)	logit	probit, cloglog, cauchit, and binomial variance

For more information about the link functions, see Oracle Machine Learning for SQL Concepts Guide.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The `oml.glm` class allows you to build two different types of models. Some arguments apply to classification models only and some to regression models only.

For information on the `oml.glm` class attributes and methods, invoke `help(oml.glm)` or see Oracle Machine Learning for Python API Reference.

Settings for a Generalized Linear Model

The following table lists the settings that apply to GLM models.

Table 10-10 Generalized Linear Model Settings

Setting Name	Setting Value	Description
`CLAS_COST_TABLE_NAME`	*table_name*	The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table. • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
`CLAS_WEIGHTS_BALANCED`	`ON` `OFF`	Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is `OFF`.
`CLAS_WEIGHTS_TABLE_NAME`	*table_name*	The name of a table that stores weighting information for individual target values in GLM logistic regression models. The weights are used by the algorithm to bias the model in favor of higher weighted classes. The class weights table is user-created. The following are the column requirements for the table. • Column Name: TARGET_VALUE Data Type: Valid target data type • Column Name: CLASS_WEIGHT Data Type: NUMBER
`GLMS_BATCH_ROWS`	`0` or a positive integer.	Number of rows in a batch used by the SGD solver. The value of this parameter sets the size of the batch for the SGD solver. An input of 0 triggers a data-driven batch size estimate. The default value is `2000`.
`GLMS_CONF_LEVEL`	`TO_CHAR(0< numeric_expr <1)`	The confidence level for coefficient confidence intervals. The default confidence level is `0.95`.
`GLMS_CONV_TOLERANCE`	The range is `(0, 1)` non-inclusive.	Convergence tolerance setting of the GLM algorithm. The default value is system-determined.
`GLMS_FTR_GEN_METHOD`	`GLMS_FTR_GEN_CUBIC` `GLMS_FTR_GEN_QUADRATIC`	Whether feature generation is cubic or quadratic. When you enable feature generation, the algorithm automatically chooses the most appropriate feature generation method based on the data.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-10 (Cont.) Generalized Linear Model Settings

Setting Name	Setting Value	Description
GLMS_FTR_GENERATION	GLMS_FTR_GENERATION_ENABLE GLMS_FTR_GENERATION_DISABLE	Whether or not feature generation is enabled for GLM. By default, feature generation is not enabled. **Note** Note: Feature generation can only be enabled when feature selection is also enabled.
GLMS_FTR_SEL_CRIT	GLMS_FTR_SEL_AIC GLMS_FTR_SEL_ALPHA_INV GLMS_FTR_SEL_RIC GLMS_FTR_SEL_SBIC	Feature selection penalty criterion for adding a feature to the model. When feature selection is enabled, the algorithm automatically chooses the penalty criterion based on the data.
GLMS_FTR_SELECTION	GLMS_FTR_SELECTION_DISABLE	Enable or disable feature selection for GLM. By default, feature selection is not enabled.
GLMS_MAX_FEATURES	TO_CHAR(0 < *numeric_expr* <= 2000)	When feature selection is enabled, this setting specifies the maximum number of features that can be selected for the final model. By default, the algorithm limits the number of features to ensure sufficient memory.
GLMS_NUM_ITERATIONS	A positive integer.	Maximum number of iterations for the GLM algorithm. The default value is system-determined.
GLMS_PRUNE_MODEL	GLMS_PRUNE_MODEL_ENABLE GLMS_PRUNE_MODEL_DISABLE	When feature selection is enabled, the algorithm automatically performs pruning based on the data.
GLMS_REFERENCE_CLASS_NAME	*target_value*	The target value used as the reference class in a binary logistic regression model. Probabilities are produced for the other class. By default, the algorithm chooses the value with the highest prevalence (the most cases) for the reference class.
GLMS_RIDGE_REGRESSION	GLMS_RIDGE_REG_ENABLE GLMS_RIDGE_REG_DISABLE	Enable or disable ridge regression. Ridge applies to both regression and classification machine learning functions. When ridge is enabled, prediction bounds are not produced by the PREDICTION_BOUNDS SQL function.
GLMS_RIDGE_VALUE	TO_CHAR(*numeric_expr* > 0)	The value of the ridge parameter. Use this setting only when you have configured the algorithm to use ridge regression. If ridge regression is enabled internally by the algorithm, then the ridge parameter is determined by the algorithm.
GLMS_ROW_DIAGNOSTICS	GLMS_ROW_DIAG_ENABLE GLMS_ROW_DIAG_DISABLE	Enable or disable row diagnostics. By default, row diagnostics are disabled.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-10 (Cont.) Generalized Linear Model Settings

Setting Name	Setting Value	Description
GLMS_SOLVER	GLMS_SOLVER_CHOL GLMS_SOLVER_LBFGS_ADMM GLMS_SOLVER_QR GLMS_SOLVER_SGD	Specifies the GLM solver. You cannot select the solver if GLMS_FTR_SELECTION setting is enabled. The default value is system determined. The GLMS_SOLVER_CHOL solver uses Cholesky decomposition. The GLMS_SOLVER_SGD solver uses stochastic gradient descent.
GLMS_SPARSE_SOLVER	GLMS_SPARSE_SOLVER_ENABLE GLMS_SPARSE_SOLVER_DISABLE	Enable or disable the use of a sparse solver if it is available. The default value is GLMS_SPARSE_SOLVER_DISABLE.
ODMS_ROW_WEIGHT_COLUMN_NAME	*column_name*	The name of a column in the training data that contains a weighting factor for the rows. The column datatype must be NUMBER. You can use row weights as a compact representation of repeated rows, as in the design of experiments where a specific configuration is repeated several times. You can also use row weights to emphasize certain rows during model construction. For example, to bias the model towards rows that are more recent and away from potentially obsolete data.
GLMS_LINK_FUNCTION **Note** Available only in Oracle Database 23ai.	GLMS_IDENTITY_LINK GLMS_LOGIT_LINK GLMS_PROBIT_LINK GLMS_CLOGLOG_LINK GLMS_CAUCHIT_LINK	This setting allows the user to specify the link function for building a GLM model. The link functions are specific to the mining function. For classification, the following are applicable: • GLMS_LOGIT_LINK (default) • GLMS_PROBIT_LINK • GLMS_CLOGLOG_LINK • GLMS_CAUCHIT_LINK For regression, the following is applicable: • GLMS_IDENTITY_LINK (default)

See Also

- About Model Settings
- Shared Settings

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 10-12 Using the oml.glm Class

This example demonstrates the use of various methods of the `oml.glm` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Petal_Width')
train_y = dat[0]['Petal_Width']
test_dat = dat[1]

# Specify settings.
setting = {'GLMS_SOLVER': 'dbms_data_mining.GLMS_SOLVER_QR'}

# Create a GLM model object.
glm_mod = oml.glm("regression", **setting)

# Fit the GLM model according to the training data and parameter
# settings.
glm_mod = glm_mod.fit(train_x, train_y)

# Show the model details.
glm_mod

# Use the model to make predictions on the test data.
glm_mod.predict(test_dat.drop('Petal_Width'),
                supplemental_cols = test_dat[:,
                  ['Sepal_Length', 'Sepal_Width',
                   'Petal_Length', 'Species']])

# Return the prediction probability.
glm_mod.predict(test_dat.drop('Petal_Width'),
                supplemental_cols = test_dat[:,
                  ['Sepal_Length', 'Sepal_Width',
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                   'Petal_Length', 'Species']],
                proba = True)

glm_mod.score(test_dat.drop('Petal_Width'),
              test_dat[:, ['Petal_Width']])

# Change the parameter setting and refit the model.
new_setting = {'GLMS_SOLVER': 'GLMS_SOLVER_SGD'}
glm_mod.set_params(**new_setting).fit(train_x, train_y)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Petal_Width')
>>> train_y = dat[0]['Petal_Width']
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... setting = {'GLMS_SOLVER': 'dbms_data_mining.GLMS_SOLVER_QR'}
>>>
>>> # Create a GLM model object.
... glm_mod = oml.glm("regression", **setting)
>>>
>>> # Fit the GLM model according to the training data and parameter
... # settings.
>>> glm_mod = glm_mod.fit(train_x, train_y)
>>>
>>> # Show the model details.
... glm_mod

Algorithm Name: Generalized Linear Model
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Mining Function: REGRESSION

Target: Petal_Width

Settings:
                setting name                  setting value
0                     ALGO_NAME  ALGO_GENERALIZED_LINEAR_MODEL
1               GLMS_CONF_LEVEL                            .95
2           GLMS_FTR_GENERATION    GLMS_FTR_GENERATION_DISABLE
3            GLMS_FTR_SELECTION     GLMS_FTR_SELECTION_DISABLE
4                   GLMS_SOLVER                GLMS_SOLVER_QR
5                  ODMS_DETAILS                   ODMS_ENABLE
6  ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
7                 ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                            ON

Computed Settings:
         setting name            setting value
0    GLMS_CONV_TOLERANCE  .0000050000000000000004
1    GLMS_NUM_ITERATIONS                      30
2  GLMS_RIDGE_REGRESSION    GLMS_RIDGE_REG_ENABLE

Global Statistics:
            attribute name  attribute value
0        ADJUSTED_R_SQUARE         0.949634
1                      AIC         -363.888
2                COEFF_VAR          14.6284
3                CONVERGED              YES
4       CORRECTED_TOTAL_DF              103
5         CORRECTED_TOT_SS          58.4565
6            DEPENDENT_MEAN          1.15577
7                 ERROR_DF               98
8        ERROR_MEAN_SQUARE         0.028585
9        ERROR_SUM_SQUARES          2.80131
10                 F_VALUE          389.405
11                   GMSEP         0.030347
12              HOCKING_SP         0.000295
13                     J_P         0.030234
14               MODEL_DF                5
15         MODEL_F_P_VALUE                0
16       MODEL_MEAN_SQUARE           11.131
17       MODEL_SUM_SQUARES          55.6552
18              NUM_PARAMS                6
19                NUM_ROWS              104
20         RANK_DEFICIENCY                0
21            ROOT_MEAN_SQ          0.16907
22                    R_SQ         0.952079
23                    SBIC         -348.021
24 VALID_COVARIANCE_MATRIX              YES
[1 rows x 25 columns]

Attributes:
Petal_Length
Sepal_Length
Sepal_Width
Species
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Partition: NO

Coefficients:

          name       level  estimate
0   (Intercept)        None -0.600603
1  Petal_Length        None  0.239775
2  Sepal_Length        None -0.078338
3   Sepal_Width        None  0.253996
4       Species  versicolor  0.652420
5       Species   virginica  1.010438

Fit Details:
                        name         value
0         ADJUSTED_R_SQUARE  9.496338e-01
1                       AIC -3.638876e+02
2                 COEFF_VAR  1.462838e+01
3        CORRECTED_TOTAL_DF  1.030000e+02
...
21             ROOT_MEAN_SQ  1.690704e-01
22                     R_SQ  9.520788e-01
23                     SBIC -3.480213e+02
24  VALID_COVARIANCE_MATRIX  1.000000e+00

Rank:

6

Deviance:

2.801309

AIC:

-364

Null Deviance:

58.456538

DF Residual:

98.0

DF Null:

103.0

Converged:

True

>>>
>>> # Use the model to make predictions on the test data.
... glm_mod.predict(test_dat.drop('Petal_Width'),
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...                 supplemental_cols = test_dat[:,
...                   ['Sepal_Length', 'Sepal_Width',
...                    'Petal_Length', 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
0            4.9          3.0           1.4      setosa    0.113215
1            4.9          3.1           1.5      setosa    0.162592
2            4.8          3.4           1.6      setosa    0.270602
3            5.8          4.0           1.2      setosa    0.248752
...           ...          ...           ...         ...         ...
42           6.7          3.3           5.7   virginica    2.89876
43           6.7          3.0           5.2   virginica    1.893790
44           6.5          3.0           5.2   virginica    1.909457
45           5.9          3.0           5.1   virginica    1.932483

>>> # Return the prediction probability.
... glm_mod.predict(test_dat.drop('Petal_Width'),
...                 supplemental_cols = test_dat[:,
...                   ['Sepal_Length', 'Sepal_Width',
...                    'Petal_Length', 'Species']]),
...                 proba = True)
    Sepal_Length  Sepal_Width     Species  PREDICTION
0            4.9          3.0      setosa    0.113215
1            4.9          3.1      setosa    0.162592
2            4.8          3.4      setosa    0.270602
3            5.8          4.0      setosa    0.248752
...          ...          ...         ...         ...
42           6.7          3.3   virginica    2.089876
43           6.7          3.0   virginica    1.893790
44           6.5          3.0   virginica    1.909457
45           5.9          3.0   virginica    1.932483
>>>
>>> glm_mod.score(test_dat.drop('Petal_Width'),
...               test_dat[:, ['Petal_Width']])
0.951252
>>>
>>> # Change the parameter setting and refit the model.
... new_setting = {'GLMS_SOLVER': 'GLMS_SOLVER_SGD'}
>>> glm_mod.set_params(**new_setting).fit(train_x, train_y)

Algorithm Name: Generalized Linear Model

Mining Function: REGRESSION

Target: Petal_Width

Settings:
                   setting name                   setting value
0                     ALGO_NAME  ALGO_GENERALIZED_LINEAR_MODEL
1                GLMS_CONF_LEVEL                             .95
2            GLMS_FTR_GENERATION    GLMS_FTR_GENERATION_DISABLE
3             GLMS_FTR_SELECTION     GLMS_FTR_SELECTION_DISABLE
4                    GLMS_SOLVER                GLMS_SOLVER_SGD
5                   ODMS_DETAILS                   ODMS_ENABLE
6  ODMS_MISSING_VALUE_TREATMENT        ODMS_MISSING_VALUE_AUTO
7                  ODMS_SAMPLING          ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                             ON
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Computed Settings:
            setting name          setting value
0        GLMS_BATCH_ROWS                   2000
1    GLMS_CONV_TOLERANCE                  .0001
2    GLMS_NUM_ITERATIONS                   500
3  GLMS_RIDGE_REGRESSION  GLMS_RIDGE_REG_ENABLE
4       GLMS_RIDGE_VALUE                    .01

Global Statistics:
            attribute name  attribute value
0        ADJUSTED_R_SQUARE          0.94175
1                      AIC         -348.764
2               COEFF_VAR          15.7316
3               CONVERGED               NO
4       CORRECTED_TOTAL_DF              103
5         CORRECTED_TOT_SS          58.4565
6            DEPENDENT_MEAN          1.15577
7                 ERROR_DF               98
8        ERROR_MEAN_SQUARE         0.033059
9        ERROR_SUM_SQUARES          3.23979
10                F_VALUE          324.347
11                  GMSEP         0.035097
12              HOCKING_SP         0.000341
13                    J_P         0.034966
14               MODEL_DF                5
15         MODEL_F_P_VALUE                0
16       MODEL_MEAN_SQUARE          10.7226
17       MODEL_SUM_SQUARES           53.613
18              NUM_PARAMS                6
19                NUM_ROWS              104
20         RANK_DEFICIENCY                0
21            ROOT_MEAN_SQ          0.181821
22                    R_SQ          0.944578
23                    SBIC         -332.898
24 VALID_COVARIANCE_MATRIX               NO

[1 rows x 25 columns]

Attributes:
Petal_Length
Sepal_Length
Sepal_Width
Species

Partition: NO

Coefficients:

           name       level  estimate
0   (Intercept)        None -0.338046
1  Petal_Length        None  0.378658
2  Sepal_Length        None -0.084440
3   Sepal_Width        None  0.137150
4       Species  versicolor  0.151916
5       Species   virginica  0.337535
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Fit Details:

                        name         value
0         ADJUSTED_R_SQUARE  9.417502e-01
1                        AIC -3.487639e+02
2                  COEFF_VAR  1.573164e+01
3        CORRECTED_TOTAL_DF  1.030000e+02
...                      ...           ...
21             ROOT_MEAN_SQ  1.818215e-01
22                     R_SQ  9.445778e-01
23                     SBIC -3.328975e+02
24  VALID_COVARIANCE_MATRIX  0.000000e+00

Rank:

6

Deviance:

3.239787

AIC:

-349

Null Deviance:

58.456538

Prior Weights:

1

DF Residual:

98.0

DF Null:

103.0

Converged:

False
```

10.13 *k*-Means

The `oml.km` class uses the *k*-Means (KM) algorithm, which is a hierarchical, distance-based clustering algorithm that partitions data into a specified number of clusters.

The algorithm has the following features:

- Several distance functions: Euclidean, Cosine, and Fast Cosine distance functions. The default is Euclidean.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- For each cluster, the algorithm returns the centroid, a histogram for each attribute, and a rule describing the hyperbox that encloses the majority of the data assigned to the cluster. The centroid reports the mode for categorical attributes and the mean and variance for numeric attributes.

For information on the `oml.km` class attributes and methods, invoke `help(oml.km)` or see Oracle Machine Learning for Python API Reference.

Settings for a *k*-Means Model

The following table lists the settings that apply to KM models.

Table 10-11 *k*-Means Model Settings

Setting Name	Setting Value	Description
CLUS_NUM_CLUSTERS	*TO_CHAR(numeric_expr* >= 1)	The maximum number of leaf clusters generated by the algorithm. The algorithm produces the specified number of clusters unless there are fewer distinct data points. The default value is 10.
KMNS_CONV_TOLERANCE	TO_CHAR(0< *numeric_expr* <1)	Minimum Convergence Tolerance for *k*-Means. The algorithm iterates until the minimum Convergence Tolerance is satisfied or until the maximum number of iterations, specified in KMNS_ITERATIONS, is reached. Decreasing the Convergence Tolerance produces a more accurate solution but may result in longer run times. The default Convergence Tolerance is 0.001.
KMNS_DETAILS	KMNS_DETAILS_ALL KMNS_DETAILS_HIERARCHY KMNS_DETAILS_NONE	Determines the level of cluster detail that is computed during the build. KMNS_DETAILS_ALL: Cluster hierarchy, record counts, descriptive statistics (means, variances, modes, histograms, and rules) are computed. KMNS_DETAILS_HIERARCHY: Cluster hierarchy and cluster record counts are computed. This is the default value. KMNS_DETAILS_NONE: No cluster details are computed. Only the scoring information is persisted.
KMNS_DISTANCE	KMNS_COSINE KMNS_EUCLIDEAN	Distance function for *k*-Means. The default distance function is KMNS_EUCLIDEAN.
KMNS_ITERATIONS	TO_CHAR(*positive_numeric_expr*)	Maximum number of iterations for *k*-Means. The algorithm iterates until either the maximum number of iterations is reached or the minimum Convergence Tolerance, specified in KMNS_CONV_TOLERANCE, is satisfied. The default number of iterations is 20.
KMNS_MIN_PCT_ATTR_SUPPORT	TO_CHAR(0< = *numeric_expr* <= 1)	Minimum percentage of attribute values that must be non-null in order for the attribute to be included in the rule description for the cluster. If the data is sparse or includes many missing values, a minimum support that is too high can cause very short rules or even empty rules. The default minimum support is 0.1.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-11 (Cont.) *k*-Means Model Settings

Setting Name	Setting Value	Description
KMNS_NUM_BINS	TO_CHAR(*numeric_expr* > 0)	Number of bins in the attribute histogram produced by *k*-Means. The bin boundaries for each attribute are computed globally on the entire training data set. The binning method is equi-width. All attributes have the same number of bins with the exception of attributes with a single value, which have only one bin. The default number of histogram bins is 11.
KMNS_RANDOM_SEED	Non-negative integer	Controls the seed of the random generator used during the *k*-Means initialization. It must be a non-negative integer value. The default value is 0.
KMNS_SPLIT_CRITERION	KMNS_SIZE KMNS_VARIANCE	Split criterion for *k*-Means. The split criterion controls the initialization of new *k*-Means clusters. The algorithm builds a binary tree and adds one new cluster at a time. When the split criterion is based on size, the new cluster is placed in the area where the largest current cluster is located. When the split criterion is based on the variance, the new cluster is placed in the area of the most spread-out cluster. The default split criterion is the KMNS_VARIANCE.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-11 (Cont.) *k*-Means Model Settings

Setting Name	Setting Value	Description
KMNS_WINSORIZE	KMNS_WINSORIZE_ENABLE KMNS_WINSORIZE_DISABLE	To winsorize data, enable or disable this parameter. Data is restricted in a window size of six standard deviations around the mean value when winsorize is enabled. This functionality can be used with AUTO_DATA_PREP turned ON and OFF. The values outside the range are replaced with the ends of the interval. Winsorize is not enabled by default.

Note

Available only in Oracle AI Database 26ai.

Note

Winsorize is only available when the KMNS_EUCLIDEAN distance function is used. An exception is raised if Winsorize is enabled and other distance functions are set.

See Also

- About Model Settings
- Shared Settings

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 10-13 Using the oml.km Class

This example creates a KM model and uses methods of it. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets


# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
test_dat = dat[1]

# Specify settings.
setting = {'kmns_iterations': 20}

# Create a KM model object and fit it.
km_mod = oml.km(n_clusters = 3, **setting).fit(train_dat)

# Show model details.
km_mod

# Use the model to make predictions on the test data.
km_mod.predict(test_dat,
               supplemental_cols =
                  test_dat[:, ['Sepal_Length', 'Sepal_Width',
                               'Petal_Length', 'Species']])
km_mod.predict_proba(test_dat,
                     supplemental_cols =
                       test_dat[:, ['Species']]).sort_values(by =
                                    ['Species', 'PROBABILITY_OF_3'])

km_mod.transform(test_dat)

km_mod.score(test_dat)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
>>>
>>> # Specify settings.
... setting = {'kmns_iterations': 20}
>>>
>>> # Create a KM model object and fit it.
... km_mod = omlkm(n_clusters = 3, **setting).fit(train_dat)
>>>
>>> # Show model details.
... km_mod

Algorithm Name: K-Means

Mining Function: CLUSTERING

Settings:
                     setting name            setting value
0                       ALGO_NAME              ALGO_KMEANS
1              CLUS_NUM_CLUSTERS                        3
2            KMNS_CONV_TOLERANCE                     .001
3                   KMNS_DETAILS   KMNS_DETAILS_HIERARCHY
4                  KMNS_DISTANCE          KMNS_EUCLIDEAN
5                KMNS_ITERATIONS                       20
6      KMNS_MIN_PCT_ATTR_SUPPORT                       .1
7                   KMNS_NUM_BINS                       11
8                KMNS_RANDOM_SEED                        0
9           KMNS_SPLIT_CRITERION            KMNS_VARIANCE
10                  ODMS_DETAILS              ODMS_ENABLE
11  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
12                ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
13                    PREP_AUTO                       ON

Global Statistics:
   attribute name  attribute value
0       CONVERGED              YES
1        NUM_ROWS            104.0


Attributes: Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Clusters:

   CLUSTER_ID  ROW_CNT  PARENT_CLUSTER_ID  TREE_LEVEL  DISPERSION
0           1      104                NaN           1    0.986153
1           2       68                1.0           2    1.102147
2           3       36                1.0           2    0.767052
3           4       37                2.0           3    1.015669
4           5       31                2.0           3    1.205363

Taxonomy:

   PARENT_CLUSTER_ID  CHILD_CLUSTER_ID
0                  1               2.0
1                  1               3.0
2                  2               4.0
3                  2               5.0
4                  3               NaN
5                  4               NaN
6                  5               NaN

Leaf Cluster Counts:

   CLUSTER_ID  CNT
0           3   50
1           4   53
2           5   47
>>>
>>> # Use the model to make predictions on the test data.
... km_mod.predict(test_dat, ['Sepal_Length', 'Sepal_Width',
...                           'Petal_Length', 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  CLUSTER_ID
0            4.9          3.0           1.4      setosa           3
1            4.9          3.1           1.5      setosa           3
2            4.8          3.4           1.6      setosa           3
3            5.8          4.0           1.2      setosa           3
...          ...          ...           ...         ...         ...
38           6.4          2.8           5.6   virginica           5
39           6.9          3.1           5.4   virginica           5
40           6.7          3.1           5.6   virginica           5
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
41           5.8          2.7           5.1   virginica           5
>>>
>>> km_mod.predict_proba(test_dat,
...                      supplemental_cols =
...                        test_dat[:, ['Species']]).sort_values(by =
...                                      ['Species', 'PROBABILITY_OF_3'])
       Species  PROBABILITY_OF_3  PROBABILITY_OF_4  PROBABILITY_OF_5
0       setosa          0.791267          0.208494          0.000240
1       setosa          0.971498          0.028350          0.000152
2       setosa          0.981020          0.018499          0.000481
3       setosa          0.981907          0.017989          0.000104
...        ...               ...               ...               ...
42   virginica          0.000655          0.316671          0.682674
43   virginica          0.001036          0.413744          0.585220
44   virginica          0.001036          0.413744          0.585220
45   virginica          0.002452          0.305021          0.692527
>>>
>>> km_mod.transform(test_dat)
    CLUSTER_DISTANCE
0           1.050234
1           0.859817
2           0.321065
3           1.427080
...              ...
42          0.837757
43          0.479313
44          0.448562
45          1.123587
>>>
>>> km_mod.score(test_dat)
-47.487712
```

10.14 Naive Bayes

The `oml.nb` class creates a Naive Bayes (NB) model for classification.

The Naive Bayes algorithm is based on conditional probabilities. Naive Bayes looks at the historical data and calculates conditional probabilities for the target values by observing the frequency of attribute values and of combinations of attribute values.

Naive Bayes assumes that each predictor is conditionally independent of the others. (Bayes' Theorem requires that the predictors be independent.)

For information on the `oml.nb` class attributes and methods, invoke `help(oml.nb)` or see Oracle Machine Learning for Python API Reference.

Settings for a Naive Bayes Model

The following table lists the settings that apply to NB models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-12 Naive Bayes Model Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	*table_name*	The name of a table that stores a cost matrix for the algorithm to use in building the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table. • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
CLAS_MAX_SUP_BINS	2 <= *a number* <= 2147483647	Specifies the maximum number of bins for each attribute. The default value is 32.
CLAS_PRIORS_TABLE_NAME	*table_name*	The name of a table that stores prior probabilities to offset differences in distribution between the build data and the scoring data. The priors table is user-created. The following are the column requirements for the table. • Column Name: TARGET_VALUE Data Type: Valid target data type • Column Name: PRIOR_PROBABILITY Data Type: NUMBER
CLAS_WEIGHTS_BALANCED	ON OFF	Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.
NABS_PAIRWISE_THRESHOLD	TO_CHAR(0 <= *numeric_expr* <= 1)	Value of the pairwise threshold for the NB algorithm. The default value is 0.
NABS_SINGLETON_THRESHOLD	TO_CHAR(0 <= *numeric_expr* <= 1)	Value of the singleton threshold for the NB algorithm. The default value is 0.

See Also

- About Model Settings
- Shared Settings

Example 10-14 Using the oml.nb Class

This example creates an NB model and uses some of the methods of the `oml.nb` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop(table = 'NB_PRIOR_PROBABILITY_DEMO')
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()

train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# User specified settings.
setting = {'CLAS_WEIGHTS_BALANCED': 'ON'}

# Create an oml NB model object.
nb_mod = oml.nb(**setting)

# Fit the NB model according to the training data and parameter
# settings.
nb_mod = nb_mod.fit(train_x, train_y)

# Show details of the model.
nb_mod

# Create a priors table in the database.
priors = {'setosa': 0.2, 'versicolor': 0.3, 'virginica': 0.5}
priors = oml.create(pd.DataFrame(list(priors.items()),
                        columns = ['TARGET_VALUE',
                                   'PRIOR_PROBABILITY']),
                        table = 'NB_PRIOR_PROBABILITY_DEMO')

# Change the setting parameter and refit the model
# with a user-defined prior table.
new_setting = {'CLAS_WEIGHTS_BALANCED': 'OFF'}
nb_mod = nb_mod.set_params(**new_setting).fit(train_x,
                                              train_y,
                                              priors = priors)
nb_mod

# Use the model to make predictions on test data.
nb_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                                                'Sepal_Width',
                                                'Petal_Length',
                                                'Species']])
# Return the prediction probability.
nb_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Species']],
               proba = True)


# Return the top two most influencial attributes of the highest
# probability class.
nb_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Petal_Length',
                                                'Species']],
               topN_attrs = 2)

# Make predictions and return the probability for each class
# on new data.
nb_mod.predict_proba(test_dat.drop('Species'),
                     supplemental_cols = test_dat[:,
                       ['Sepal_Length',
                        'Species']]).sort_values(by =
                          ['Sepal_Length',
                           'Species',
                           'PROBABILITY_OF_setosa',
                           'PROBABILITY_OF_versicolor'])

# Make predictions on new data and return the mean accuracy.
nb_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop(table = 'NB_PRIOR_PROBABILITY_DEMO')
...    oml.drop('IRIS')
... except:
...    pass
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
>>> dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # User specified settings.
... setting = {'CLAS_WEIGHTS_BALANCED': 'ON'}
>>>
>>> # Create an oml NB model object.
... nb_mod = oml.nb(**setting)
>>>
>>> # Fit the NB model according to the training data and parameter
... # settings.
>>> nb_mod = nb_mod.fit(train_x, train_y)
>>>
>>> # Show details of the model.
... nb_mod

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name            setting value
0                      ALGO_NAME         ALGO_NAIVE_BAYES
1          CLAS_WEIGHTS_BALANCED                       ON
2        NABS_PAIRWISE_THRESHOLD                        0
3       NABS_SINGLETON_THRESHOLD                        0
4                   ODMS_DETAILS             ODMS_ENABLE
5  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
6                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
7                      PREP_AUTO                       ON

Global Statistics:
   attribute name  attribute value
0        NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Priors:

   TARGET_NAME TARGET_VALUE  PRIOR_PROBABILITY  COUNT
0      Species       setosa           0.333333     36
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
1     Species   versicolor           0.333333     35
2     Species    virginica           0.333333     33

Conditionals:

    TARGET_NAME TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME
ATTRIBUTE_VALUE  \
0       Species       setosa   Petal_Length              None       ( ;
1.05]
1       Species       setosa   Petal_Length              None     (1.05; 1.2]
2       Species       setosa   Petal_Length              None     (1.2; 1.35]
3       Species       setosa   Petal_Length              None    (1.35; 1.45]
...         ...          ...            ...               ...             ...

152     Species    virginica    Sepal_Width              None    (3.25; 3.35]
153     Species    virginica    Sepal_Width              None    (3.35; 3.45]
154     Species    virginica    Sepal_Width              None    (3.55; 3.65]
155     Species    virginica    Sepal_Width              None    (3.75; 3.85]

     CONDITIONAL_PROBABILITY  COUNT
0                   0.027778      1
1                   0.027778      1
2                   0.083333      3
3                   0.277778     10
...                      ...    ...
152                 0.030303      1
153                 0.060606      2
154                 0.030303      1
155                 0.060606      2

[156 rows x 7 columns]

>>> # Create a priors table in the database.
... priors = {'setosa': 0.2, 'versicolor': 0.3, 'virginica': 0.5}
>>> priors = oml.create(pd.DataFrame(list(priors.items()),
...                         columns = ['TARGET_VALUE',
...                                    'PRIOR_PROBABILITY']),
...                         table = 'NB_PRIOR_PROBABILITY_DEMO')
>>>
>>> # Change the setting parameter and refit the model
... # with a user-defined prior table.
... new_setting = {'CLAS_WEIGHTS_BALANCED': 'OFF'}
>>> nb_mod = nb_mod.set_params(**new_setting).fit(train_x,
...                                               train_y,
...                                               priors = priors)
>>> nb_mod

Algorithm Name: Naive Bayes

Mining Function: CLASSIFICATION

Target: Species

Settings:
                   setting name                         setting value
0                     ALGO_NAME                    ALGO_NAIVE_BAYES
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
1        CLAS_PRIORS_TABLE_NAME "OML_USER"."NB_PRIOR_PROBABILITY_DEMO"
2         CLAS_WEIGHTS_BALANCED                                    OFF
3       NABS_PAIRWISE_THRESHOLD                                      0
4      NABS_SINGLETON_THRESHOLD                                      0
5                  ODMS_DETAILS                          ODMS_ENABLE
6  ODMS_MISSING_VALUE_TREATMENT              ODMS_MISSING_VALUE_AUTO
7                 ODMS_SAMPLING                ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                                   ON

Global Statistics:
   attribute name  attribute value
0        NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

Priors:

  TARGET_NAME TARGET_VALUE  PRIOR_PROBABILITY  COUNT
0     Species       setosa                0.2     36
1     Species   versicolor                0.3     35
2     Species    virginica                0.5     33

Conditionals:

    TARGET_NAME TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME
ATTRIBUTE_VALUE  \
0       Species       setosa   Petal_Length              None       ( ; 1.05]
1       Species       setosa   Petal_Length              None     (1.05; 1.2]
2       Species       setosa   Petal_Length              None     (1.2; 1.35]
3       Species       setosa   Petal_Length              None    (1.35; 1.45]
...         ...          ...            ...               ...             ...
152     Species    virginica    Sepal_Width              None    (3.25; 3.35]
153     Species    virginica    Sepal_Width              None    (3.35; 3.45]
154     Species    virginica    Sepal_Width              None    (3.55; 3.65]
155     Species    virginica    Sepal_Width              None    (3.75; 3.85]

     CONDITIONAL_PROBABILITY  COUNT
0                   0.027778      1
1                   0.027778      1
2                   0.083333      3
3                   0.277778     10
...                      ...    ...
152                 0.030303      1
153                 0.060606      2
154                 0.030303      1
155                 0.060606      2

[156 rows x 7 columns]

>>> # Use the model to make predictions on test data.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
... nb_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
0            4.9          3.0           1.4      setosa      setosa
1            4.9          3.1           1.5      setosa      setosa
2            4.8          3.4           1.6      setosa      setosa
3            5.8          4.0           1.2      setosa      setosa
...          ...          ...           ...         ...         ...
42           6.7          3.3           5.7   virginica   virginica
43           6.7          3.0           5.2   virginica   virginica
44           6.5          3.0           5.2   virginica   virginica
45           5.9          3.0           5.1   virginica   virginica

>>> # Return the prediction probability.
>>> nb_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']],
...                proba = True)
    Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
0            4.9          3.0      setosa      setosa     1.000000
1            4.9          3.1      setosa      setosa     1.000000
2            4.8          3.4      setosa      setosa     1.000000
3            5.8          4.0      setosa      setosa     1.000000
...           ...          ...         ...         ...          ...
42           6.7          3.3   virginica   virginica     1.000000
43           6.7          3.0   virginica   virginica     0.953848
44           6.5          3.0   virginica   virginica     1.000000
45           5.9          3.0   virginica   virginica     0.932334

>>> # Return the top two most influencial attributes of the highest
... # probability class.
>>> nb_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']],
...                topN_attrs = 2)
  Sepal_Length  Sepal_Width Petal_Length    Species PREDICTION \
0          4.9          3.0          1.4     setosa     setosa
1          4.9          3.1          1.5     setosa     setosa
2          4.8          3.4          1.6     setosa     setosa
3          5.8          4.0          1.2     setosa     setosa
... ... ... ... ... ...
42         6.7          3.3          5.7  virginica  virginica
43         6.7          3.0          5.2  virginica  virginica
44         6.5          3.0          5.2  virginica  virginica
45         5.9          3.0          5.1  virginica  virginica
                                  TOP_N_ATTRIBUTES
0 <Details algorithm="Naive Bayes" class="setosa...
1 <Details algorithm="Naive Bayes" class="setosa...
2 <Details algorithm="Naive Bayes" class="setosa...
3 <Details algorithm="Naive Bayes" class="setosa...
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...
42 <Details algorithm="Naive Bayes" class="virgin...
43 <Details algorithm="Naive Bayes" class="virgin...
44 <Details algorithm="Naive Bayes" class="virgin...
45 <Details algorithm="Naive Bayes" class="virgin...

>>> # Make predictions and return the probability for each class
... # on new data.
>>> nb_mod.predict_proba(test_dat.drop('Species'),
...                      supplemental_cols = test_dat[:,
...                        ['Sepal_Length',
...                         'Species']]).sort_values(by =
...                           ['Sepal_Length',
...                            'Species',
...                            'PROBABILITY_OF_setosa,
...                            'PROBABILITY_OF_versicolor'])
    Sepal_Length     Species  PROBABILITY_OF_SETOSA  \
0            4.4      setosa           1.000000e+00
1            4.4      setosa           1.000000e+00
2            4.5      setosa           1.000000e+00
3            4.8      setosa           1.000000e+00
...          ...         ...                    ...
42           6.7   virginica           1.412132e-13
43           6.9  versicolor           5.295492e-20
44           6.9   virginica           5.295492e-20
45           7.0  versicolor           6.189014e-14

     PROBABILITY_OF_VERSICOLOR  PROBABILITY_OF_VIRGINICA
0                9.327306e-21              7.868301e-20
1                3.497737e-20              1.032715e-19
2                2.238553e-13              2.360490e-19
3                6.995487e-22              2.950617e-21
...                       ...                       ...
42               4.741700e-13              1.000000e+00
43               1.778141e-07              9.999998e-01
44               2.963565e-20              1.000000e+00
45               4.156340e-01              5.843660e-01

>>> # Make predictions on new data and return the mean accuracy.
... nb_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.934783
```

10.15 Neural Network

The `oml.nn` class creates a Neural Network (NN) model for classification and regression.

Neural Network models can be used to capture intricate nonlinear relationships between inputs and outputs or to find patterns in data.

The `oml.nn` class methods build a feed-forward neural network for regression on `oml.DataFrame` data. It supports multiple hidden layers with a specifiable number of nodes. Each layer can have one of several activation functions.

The output layer is a single numeric or binary categorical target. The output layer can have any of the activation functions. It has the linear activation function by default.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Modeling with the `oml.nn` class is well-suited for noisy and complex data such as sensor data. Problems that such data might have are the following:

- Potentially many (numeric) predictors, for example, pixel values
- The target may be discrete-valued, real-valued, or a vector of such values
- Training data may contain errors – robust to noise
- Fast scoring
- Model transparency is not required; models difficult to interpret

Typical steps in Neural Network modeling are the following:

1. Specifying the architecture
2. Preparing the data
3. Building the model
4. Specifying the stopping criteria: iterations, error on a validation set within tolerance
5. Viewing statistical results from the model
6. Improving the model

For information on the `oml.nn` class attributes and methods, invoke `help(oml.nn)` or `help(oml.hist)`, or see Oracle Machine Learning for Python API Reference.

Settings for a Neural Network Model

The following table lists settings for NN models.

Table 10-13 Neural Network Models Settings

Setting Name	Setting Value	Description
`CLAS_COST_TABLE_NAME`	*table_name*	The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table. • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
`CLAS_WEIGHTS_BALANCED`	`ON` `OFF`	Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is `OFF`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-13 (Cont.) Neural Network Models Settings

Setting Name	Setting Value	Description
NNET_ACTIVATIONS	A list of the following strings: • "NNET_ACTIVATIONS_ARCTAN" • "NNET_ACTIVATIONS_BIPOLAR_SIG" • "NNET_ACTIVATIONS_LINEAR" • "NNET_ACTIVATIONS_LOG_SIG" • "NNET_ACTIVATIONS_TANH"	Defines the activation function for the hidden layers. For example, '"NNET_ACTIVATIONS_BIPOLAR_SIG", "NNET_ACTIVATIONS_TANH"'. Different layers can have different activation functions. The default value is "NNET_ACTIVATIONS_LOG_SIG". The number of activation functions must be consistent with NNET_HIDDEN_LAYERS and NNET_NODES_PER_LAYER. **Note** All quotes are single and two single quotes are used to escape a single quote in SQL statements.
NNET_HELDASIDE_MAX_FAIL	A positive integer	With NNET_REGULARIZER_HELDASIDE, the training process is stopped early if the network performance on the validation data fails to improve or remains the same for NNET_HELDASIDE_MAX_FAIL epochs in a row. The default value is 6.
NNET_HELDASIDE_RATIO	0 <= *numeric_expr* <= 1	Defines the held ratio for the held-aside method. The default value is 0.25.
NNET_HIDDEN_LAYERS	A non-negative integer	Defines the topology by number of hidden layers. The default value is 1.
NNET_ITERATIONS	A positive integer	Specifies the maximum number of iterations in the Neural Network algorithm. The default value is 200.
NNET_NODES_PER_LAYER	A list of positive integers	Defines the topology by number of nodes per layer. Different layers can have different number of nodes. The value should be a comma separated list non-negative integers. For example, '10, 20, 5'. The setting values must be consistent with NNET_HIDDEN_LAYERS. The default number of nodes per layer is the number of attributes or 50 (if the number of attributes > 50).
NNET_REG_LAMBDA	TO_CHAR(*numeric_expr* >= 0)	Defines the L2 regularization parameter lambda. This can not be set together with NNET_REGULARIZER_HELDASIDE. The default value is 1.
NNET_REGULARIZER	NNET_REGULARIZER_HELDASIDE NNET_REGULARIZER_L2 NNET_REGULARIZER_NONE	Regularization setting for the Neural Network algorithm. If the total number of training rows is greater than 50000, then the default is NNET_REGULARIZER_HELDASIDE. If the total number of training rows is less than or equal to 50000, then the default is NNET_REGULARIZER_NONE.
NNET_SOLVER	NNET_SOLVER_ADAM NNET_SOLVER_LBFGS	Specifies the method of optimization. The default value is NNET_SOLVER_LBFGS.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-13 (Cont.) Neural Network Models Settings

Setting Name	Setting Value	Description
NNET_TOLERANCE	TO_CHAR(0 < *numeric_expr* < 1)	Defines the convergence tolerance setting of the Neural Network algorithm. The default value is 0.000001.
NNET_WEIGHT_LOWER_BOUND	A real number	Specifies the lower bound of the region where weights are randomly initialized. NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND must be set together. Setting one and not setting the other raises an error. NNET_WEIGHT_LOWER_BOUND must not be greater than NNET_WEIGHT_UPPER_BOUND. The default value is –sqrt(6/(l_nodes+r_nodes)). The value of l_nodes for: • input layer dense attributes is (1+number of dense attributes) • input layer sparse attributes is number of sparse attributes • each hidden layer is (1+number of nodes in that hidden layer) The value of r_nodes is the number of nodes in the layer that the weight is connecting to.
NNET_WEIGHT_UPPER_BOUND	A real number	Specifies the upper bound of the region where weights are initialized. It should be set in pairs with NNET_WEIGHT_LOWER_BOUND and its value must not be smaller than the value of NNET_WEIGHT_LOWER_BOUND. If not specified, the values of NNET_WEIGHT_LOWER_BOUND and NNET_WEIGHT_UPPER_BOUND are system determined. The default value is sqrt(6/(l_nodes+r_nodes)). See NNET_WEIGHT_LOWER_BOUND.
ODMS_RANDOM_SEED	A non-negative integer	Controls the random number seed used by the hash function to generate a random number with uniform distribution. The default values is 0.

See Also

- About Model Settings
- Shared Settings

Example 10-15 Building a Neural Network Model

This example creates an NN model and uses some of the methods of the oml.nn class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                              'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a Neural Network model object.
nn_mod = oml.nn(nnet_hidden_layers = 1,
                nnet_activations= "'NNET_ACTIVATIONS_LOG_SIG'",
                NNET_NODES_PER_LAYER= '30')

# Fit the NN model according to the training data and parameter
# settings.
nn_mod = nn_mod.fit(train_x, train_y)

# Show details of the model.
nn_mod

# Use the model to make predictions on test data.
nn_mod.predict(test_dat.drop('Species'),
    supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                     'Petal_Length', 'Species']])

nn_mod.predict(test_dat.drop('Species'),
    supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
                                  'Species']], proba = True)

nn_mod.predict_proba(test_dat.drop('Species'),
    supplemental_cols = test_dat[:, ['Sepal_Length',
      'Species']]).sort_values(by = ['Sepal_Length', 'Species',
         'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])

nn_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

# Change the setting parameter and refit the model.
new_setting = {'NNET_NODES_PER_LAYER': '50'}
nn_mod.set_params(**new_setting).fit(train_x, train_y)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # Create a Neural Network model object.
... nn_mod = oml.nn(nnet_hidden_layers = 1,
...                 nnet_activations= "'NNET_ACTIVATIONS_LOG_SIG'",
...                 NNET_NODES_PER_LAYER= '30')
>>>
>>> # Fit the NN model according to the training data and parameter
... # settings.
... nn_mod = nn_mod.fit(train_x, train_y)
>>>
>>> # Show details of the model.
... nn_mod

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name              setting value
0                      ALGO_NAME         ALGO_NEURAL_NETWORK
1          CLAS_WEIGHTS_BALANCED                         OFF
2       LBFGS_GRADIENT_TOLERANCE                .000000001
3            LBFGS_HISTORY_DEPTH                          20
4            LBFGS_SCALE_HESSIAN  LBFGS_SCALE_HESSIAN_ENABLE
5               NNET_ACTIVATIONS  'NNET_ACTIVATIONS_LOG_SIG'
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
6        NNET_HELDASIDE_MAX_FAIL                           6
7           NNET_HELDASIDE_RATIO                         .25
8             NNET_HIDDEN_LAYERS                           1
9                NNET_ITERATIONS                         200
10          NNET_NODES_PER_LAYER                          30
11                NNET_TOLERANCE                      .000001
12                  ODMS_DETAILS                  ODMS_ENABLE
13  ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
14              ODMS_RANDOM_SEED                           0
15                 ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
16                     PREP_AUTO                          ON

Computed Settings:
       setting name          setting value
0  NNET_REGULARIZER  NNET_REGULARIZER_NONE

Global Statistics:
   attribute name  attribute value
0       CONVERGED              YES
1      ITERATIONS             60.0
2      LOSS_VALUE              0.0
3        NUM_ROWS            102.0


Attributes:
Sepal_Length
Sepal_Width
Petal_Length
Petal_Width

Partition: NO

Topology:

    HIDDEN_LAYER_ID  NUM_NODE       ACTIVATION_FUNCTION
 0                0        30  NNET_ACTIVATIONS_LOG_SIG

Weights:

      LAYER  IDX_FROM  IDX_TO ATTRIBUTE_NAME ATTRIBUTE_SUBNAME
ATTRIBUTE_VALUE  \
 0        0       0.0       0   Petal_Length              None
None
 1        0       0.0       1   Petal_Length              None
None
 2        0       0.0       2   Petal_Length              None
None
 3        0       0.0       3   Petal_Length              None
None
 ...    ...       ...     ...            ...               ...             ...

 239      1      29.0       2           None              None
None
 240      1       NaN       0           None              None
None
 241      1       NaN       1           None              None
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
None
 242      1       NaN       2           None             None
None

     TARGET_VALUE     WEIGHT
 0           None -39.836487
 1           None  32.604824
 2           None   0.953903
 3           None   0.714064
 ...          ...        ...
 239    virginica -22.650606
 240       setosa   2.402457
 241   versicolor   7.647615
 242    virginica  -9.493982

[243 rows x 8 columns]



>>> # Use the model to make predictions on test data.
... nn_mod.predict(test_dat.drop('Species'),
...     supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
...                                      'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
 0            4.9          3.0           1.4      setosa      setosa
 1            4.9          3.1           1.5      setosa      setosa
 2            4.8          3.4           1.6      setosa      setosa
 3            5.8          4.0           1.2      setosa      setosa
...           ...          ...           ...         ...         ...
 44           6.7          3.3           5.7   virginica   virginica
 45           6.7          3.0           5.2   virginica   virginica
 46           6.5          3.0           5.2   virginica   virginica
 47           5.9          3.0           5.1   virginica   virginica

>>> nn_mod.predict(test_dat.drop('Species'),
...     supplemental_cols = test_dat[:, ['Sepal_Length', 'Sepal_Width',
...                                      'Species']], proba = True)
     Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
 0            4.9          3.0      setosa      setosa     1.000000
 1            4.9          3.1      setosa      setosa     1.000000
 2            4.8          3.4      setosa      setosa     1.000000
 3            5.8          4.0      setosa      setosa     1.000000
...           ...          ...         ...         ...          ...
 44           6.7          3.3   virginica   virginica     1.000000
 45           6.7          3.0   virginica   virginica     1.000000
 46           6.5          3.0   virginica   virginica     1.000000
 47           5.9          3.0   virginica   virginica     1.000000

>>> nn_mod.predict_proba(test_dat.drop('Species'),
...     supplemental_cols = test_dat[:, ['Sepal_Length',
...       'Species']]).sort_values(by = ['Sepal_Length', 'Species',
...         'PROBABILITY_OF_setosa', 'PROBABILITY_OF_versicolor'])
     Sepal_Length     Species  PROBABILITY_OF_SETOSA  \
 0            4.4      setosa           1.000000e+00
 1            4.4      setosa           1.000000e+00
 2            4.5      setosa           1.000000e+00
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
 3            4.8      setosa           1.000000e+00
...           ...         ...                    ...
 44           6.7   virginica          4.567318e-218
 45           6.9  versicolor          3.028266e-177
 46           6.9   virginica          1.203417e-215
 47           7.0  versicolor          3.382837e-148

    PROBABILITY_OF_VERSICOLOR  PROBABILITY_OF_VIRGINICA
 0               3.491272e-67             3.459448e-283
 1               8.038930e-58             2.883999e-288
 2               5.273544e-64             2.243282e-293
 3               1.332150e-78             2.040723e-283
...                      ...                       ...
 44              1.328042e-36              1.000000e+00
 45              1.000000e+00              5.063405e-55
 46              4.000953e-31              1.000000e+00
 47              1.000000e+00             2.593761e-121

>>> nn_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
 0.9375

>>> # Change the setting parameter and refit the model.
... new_setting = {'NNET_NODES_PER_LAYER': '50'}
>>> nn_mod.set_params(**new_setting).fit(train_x, train_y)

Algorithm Name: Neural Network

Mining Function: CLASSIFICATION

Target: Species

Settings:
                     setting name              setting value
0                       ALGO_NAME        ALGO_NEURAL_NETWORK
1           CLAS_WEIGHTS_BALANCED                        OFF
2        LBFGS_GRADIENT_TOLERANCE                  .000000001
3             LBFGS_HISTORY_DEPTH                         20
4             LBFGS_SCALE_HESSIAN  LBFGS_SCALE_HESSIAN_ENABLE
5                NNET_ACTIVATIONS  'NNET_ACTIVATIONS_LOG_SIG'
6         NNET_HELDASIDE_MAX_FAIL                           6
7            NNET_HELDASIDE_RATIO                         .25
8              NNET_HIDDEN_LAYERS                           1
9                 NNET_ITERATIONS                         200
10          NNET_NODES_PER_LAYER                          50
11                NNET_TOLERANCE                     .000001
12                  ODMS_DETAILS                ODMS_ENABLE
13  ODMS_MISSING_VALUE_TREATMENT    ODMS_MISSING_VALUE_AUTO
14               ODMS_RANDOM_SEED                           0
15                  ODMS_SAMPLING      ODMS_SAMPLING_DISABLE
16                      PREP_AUTO                         ON

Computed Settings:
       setting name          setting value
0  NNET_REGULARIZER  NNET_REGULARIZER_NONE

Global Statistics:
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
   attribute name  attribute value
0       CONVERGED              YES
1      ITERATIONS             68.0
2      LOSS_VALUE              0.0
3        NUM_ROWS            102.0

Attributes:
Sepal_Length
Sepal_Width
Petal_Length
Petal_Width

Partition: NO

Topology:

    HIDDEN_LAYER_ID  NUM_NODE       ACTIVATION_FUNCTION
 0                0        50  NNET_ACTIVATIONS_LOG_SIG

Weights:

      LAYER  IDX_FROM  IDX_TO ATTRIBUTE_NAME ATTRIBUTE_SUBNAME
ATTRIBUTE_VALUE  \
 0        0       0.0       0   Petal_Length              None
None
 1        0       0.0       1   Petal_Length              None
None
 2        0       0.0       2   Petal_Length              None
None
 3        0       0.0       3   Petal_Length              None
None
 ...    ...       ...     ...            ...               ...              ...

 399      1      49.0       2           None              None
None
 400      1       NaN       0           None              None
None
 401      1       NaN       1           None              None
None
 402      1       NaN       2           None              None
None

     TARGET_VALUE     WEIGHT
 0           None  10.606389
 1           None -37.256485
 2           None -14.263772
 3           None -17.945173
 ...          ...        ...
 399    virginica -22.179815
 400       setosa  -6.452953
 401   versicolor  13.186332
 402    virginica  -6.973605

[403 rows x 8 columns]
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

10.16 ONNX

The `oml.onnx` class allows you to import your own ONNX-format model, load it in the database, and score data using the prediction operators of Oracle Machine Learning.

You can provide an onnx file (.onnx) along with its associated metadata to load your ONNX-format model in the database and score using the existing data in the database. You can also access your already existing ONNX model in the database. The python module, `onnx`, accepts ONNX-format models with the following characteristics:

- **Techniques**: The pretrained models must use one of the following machine learning techniques:
  - Regression
  - Classification
  - Clustering
  - Embedding
- **Data types**: The model inputs must be one of the following Python types:
  - Numerical (float, int8, int16, int32, int64, uint8, uint16, uint32, uint64)
  - String
- **Input types**: The model's inputs must be two-dimensional vectors of the shape [batch_size, n]. In this structure, the first dimension specifies the size of each batch, while the second dimension defines the size of each individual input.

For information on the `oml.onnx` class attributes and methods, invoke `help(oml.onnx)` or see Oracle Machine Learning for Python API Reference.

ONNX-format Model Examples

1. Load an ONNX-format model into the database:

   Use the onnx module to load your ONNX-format models into the database. You need to provide the onnx file (.onnx) along with its associated metadata to build the model in OML4Py.

   > **Note**
   >
   > If the model's name is not provided when loading it to the database, a system-generated model name will be used. For example `model.load2db()`

```
from oml.algo import onnx
model = onnx(onnx_file="path_to_model.onnx",
         mining_function='CLASSIFICATION',
         classification_prob_output="output",
         model_input={"tensor_of_dim_4":
['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']},
         apply_softmax=True,
         labels=['setosa','versicolor','virginica'],
         description={"description":"This model is useful.","author":"John
Doe"},
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
default_on_null={'Sepal_Length':3.2,'Sepal_Width':2.5,'Petal_Length':46,'Pe
tal_Width':1.8})
model.load2db("model1")
```

2. Access an existing model:

   An existing ONNX model in the database can be accessed using the onnx module. In the following example, a model named classif_auto owned by omluser is accessed.

   ```
   from oml.algo import onnx
   model = onnx(model_name="classif_auto", model_owner="omluser")
   ```

   Once the model is loaded, you can view the model information by entering `model`.

3. Score data using an ONNX-format models:

   In this example, an ONNX-format model loaded in the database is used for scoring:

   ```
   model.predict(oml_iris.drop('Species'),
                     supplemental_cols=oml_iris[:,['Sepal_Length',
   'Sepal_Width', 'Species']], proba=True)
   ```

Final Example: Putting it All Together

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

x = pd.DataFrame(iris.data,
                  columns = ['Sepal_Length','Sepal_Width',
                             'Petal_Length','Petal_Width'])

y = pd.DataFrame(list(map(lambda x:
                           {0: 'setosa', 1: 'versicolor',
                            2:'virginica'}[x], iris.target)),
                  columns = ['Species'])

 try:
     oml.drop('IRIS')
 except:
     pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
model = oml.algo.onnx(onnx_file="model.onnx",
mining_function='CLASSIFICATION', classification_prob_output="output",
         model_input={"tensor_of_dim_4":
['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']},
         apply_softmax=True, labels=['setosa','versicolor','virginica'],
         description={"description":"This model is useful.","author":"John
Doe"},

default_on_null={'Sepal_Length':3.2,'Sepal_Width':2.5,'Petal_Length':4.6,'Peta
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
l_Width':1.8})

model.load2db("model1")
model.predict(oml_iris.drop('Species'),
               supplemental_cols=oml_iris[:,['Sepal_Length', 'Sepal_Width',
'Species']], proba=True)
```

```
Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
0            4.9          3.0      setosa      setosa     1.000000
1            4.9          3.1      setosa      setosa     1.000000
2            4.8          3.4      setosa      setosa     1.000000
3            5.8          4.0      setosa      setosa     1.000000
...          ...          ...         ...         ...          ...
44           6.7          3.3   virginica  versicolor     0.514706
45           6.7          3.0   virginica  versicolor     0.514706
46           6.5          3.0   virginica  versicolor     0.514706
47           5.9          3.0   virginica  versicolor     0.514706
```

```
model.predict_proba(oml_iris.drop('Species'),
                    supplemental_cols = oml_iris[:,['Sepal_Length',
'Species']])
```

```
Sepal_Length     Species  PROBABILITY_OF_SETOSA  \
0            4.4      setosa                    1.0
1            4.4      setosa                    1.0
2            4.5      setosa                    1.0
3            4.8      setosa                    1.0
...          ...         ...                    ...
42           6.7   virginica                    0.0
43           6.9  versicolor                    0.0
44           6.9   virginica                    0.0
45           7.0  versicolor                    0.0

    PROBABILITY_OF_VERSICOLOR  PROBABILITY_OF_VIRGINICA
0                    0.000000                 0.000000
1                    0.000000                 0.000000
2                    0.000000                 0.000000
3                    0.000000                 0.000000
...                       ...                      ...
42                   0.514706                 0.485294
43                   0.514706                 0.485294
44                   0.514706                 0.485294
45                   0.514706                 0.485294
```

Related Topics

- About ONNX

Copyright © 2024, 2026, Oracle and/or its affiliates.

10.17 Random Forest

The `oml.rf` class creates a Random Forest (RF) model that provides an ensemble learning technique for classification.

By combining the ideas of bagging and random selection of variables, the Random Forest algorithm produces a collection of decision trees with controlled variance while avoiding overfitting, which is a common problem for decision trees.

For information on the `oml.rf` class attributes and methods, invoke `help(oml.rf)` or see Oracle Machine Learning for Python API Reference.

Settings for a Random Forest Model

The following table lists settings for RF models.

Table 10-14 Random Forest Model Settings

Setting Name	Setting Value	Description
`CLAS_COST_TABLE_NAME`	*table_name*	The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table. • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
`CLAS_MAX_SUP_BINS`	`2 <= a number <= 254`	Specifies the maximum number of bins for each attribute. The default value is 32.
`CLAS_WEIGHTS_BALANCED`	`ON` `OFF`	Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is `OFF`.
`ODMS_RANDOM_SEED`	A non-negative integer	Controls the random number seed used by the hash function to generate a random number with uniform distribution. The default values is `0`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-14 (Cont.) Random Forest Model Settings

Setting Name	Setting Value	Description
RFOR_MTRY	*A number* >= 0	Size of the random subset of columns to consider when choosing a split at a node. For each node, the size of the pool remains the same but the specific candidate columns change. The default is half of the columns in the model signature. The special value 0 indicates that the candidate pool includes all columns.
RFOR_NUM_TREES	1 <= *a number* <= 65535	Number of trees in the forest The default value is 20.
RFOR_SAMPLING_RATIO	0 < *a fraction* <= 1	Fraction of the training data to be randomly sampled for use in the construction of an individual tree. The default is half of the number of rows in the training data.
TREE_IMPURITY_METRIC	TREE_IMPURITY_ENTROPY TREE_IMPURITY_GINI	Tree impurity metric for a decision tree model. Tree algorithms seek the best test question for splitting data at each node. The best splitter and split value are those that result in the largest increase in target value homogeneity (purity) for the entities in the node. Purity is measured in accordance with a metric. Decision trees can use either gini (TREE_IMPURITY_GINI) or entropy (TREE_IMPURITY_ENTROPY) as the purity metric. By default, the algorithm uses TREE_IMPURITY_GINI.
TREE_TERM_MAX_DEPTH	2 <= *a number* <= 100	Criteria for splits: maximum tree depth (the maximum number of nodes between the root and any leaf node, including the leaf node). The default is 16.
TREE_TERM_MINPCT_NODE	0 <= *a number* <= 10	The minimum number of training rows in a node expressed as a percentage of the rows in the training data. The default value is 0.05, indicating 0.05%.
TREE_TERM_MINPCT_SPLIT	0 < *a number* <= 20	Minimum number of rows required to consider splitting a node expressed as a percentage of the training rows. The default value is 0.1, indicating 0.1%.
TREE_TERM_MINREC_NODE	*A number* >= 0	Minimum number of rows in a node. The default value is 10.
TREE_TERM_MINREC_SPLIT	*A number* > 1	Criteria for splits: minimum number of records in a parent node expressed as a value. No split is attempted if the number of records is below this value. The default value is 20.

Copyright © 2024, 2026, Oracle and/or its affiliates.

See Also

- About Model Settings
- Shared Settings

Example 10-16 Using the oml.rf Class

This example creates an RF model and uses some of the methods of the `oml.rf` class.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
    oml.drop(table = 'RF_COST')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create a cost matrix table in the database.
cost_matrix = [['setosa', 'setosa', 0],
               ['setosa', 'virginica', 0.2],
               ['setosa', 'versicolor', 0.8],
               ['virginica', 'virginica', 0],
               ['virginica', 'setosa', 0.5],
               ['virginica', 'versicolor', 0.5],
               ['versicolor', 'versicolor', 0],
               ['versicolor', 'setosa', 0.4],
               ['versicolor', 'virginica', 0.6]]
cost_matrix = \
  oml.create(pd.DataFrame(cost_matrix,
                          columns = ['ACTUAL_TARGET_VALUE',
                                     'PREDICTED_TARGET_VALUE',
                                     'COST']),
             table = 'RF_COST')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Create an RF model object.
rf_mod = oml.rf(tree_term_max_depth = '2')

# Fit the RF model according to the training data and parameter
# settings.
rf_mod = rf_mod.fit(train_x, train_y, cost_matrix = cost_matrix)

# Show details of the model.
rf_mod

# Use the model to make predictions on the test data.
rf_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Petal_Length',
                                                'Species']])

# Return the prediction probability.
rf_mod.predict(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Sepal_Width',
                                                'Species']],
               proba = True)


# Return the top two most influencial attributes of the highest
# probability class.
rf_mod.predict_proba(test_dat.drop('Species'),
               supplemental_cols = test_dat[:, ['Sepal_Length',
                                                'Species']],
               topN = 2).sort_values(by = ['Sepal_Length', 'Species'])

rf_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])

# Reset TREE_TERM_MAX_DEPTH and refit the model.
rf_mod.set_params(tree_term_max_depth = '3').fit(train_x, train_y,
cost_matrix)
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> try:
...    oml.drop('IRIS')
...    oml.drop(table = 'RF_COST')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # Create a cost matrix table in the database.
... cost_matrix = [['setosa', 'setosa', 0],
...                ['setosa', 'virginica', 0.2],
...                ['setosa', 'versicolor', 0.8],
...                ['virginica', 'virginica', 0],
...                ['virginica', 'setosa', 0.5],
...                ['virginica', 'versicolor', 0.5],
...                ['versicolor', 'versicolor', 0],
...                ['versicolor', 'setosa', 0.4],
...                ['versicolor', 'virginica', 0.6]]
>>> cost_matrix = \
...   oml.create(pd.DataFrame(cost_matrix,
...                           columns = ['ACTUAL_TARGET_VALUE',
...                                      'PREDICTED_TARGET_VALUE',
...                                      'COST']),
...              table = 'RF_COST')
>>>
>>> # Create an RF model object.
... rf_mod = oml.rf(tree_term_max_depth = '2')
>>>
>>> # Fit the RF model according to the training data and parameter
... # settings.
>>> rf_mod = rf_mod.fit(train_x, train_y, cost_matrix = cost_matrix)
>>>
>>> # Show details of the model.
... rf_mod

Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species

Settings:
                    setting name            setting value
0                      ALGO_NAME       ALGO_RANDOM_FOREST
1           CLAS_COST_TABLE_NAME     "OML_USER"."RF_COST"
2              CLAS_MAX_SUP_BINS                       32
3          CLAS_WEIGHTS_BALANCED                      OFF
4                   ODMS_DETAILS             ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
6               ODMS_RANDOM_SEED                         0
7                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
8                      PREP_AUTO                       ON
9                 RFOR_NUM_TREES                       20
10           RFOR_SAMPLING_RATIO                       .5
11          TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
12           TREE_TERM_MAX_DEPTH                        2
13         TREE_TERM_MINPCT_NODE                      .05
14        TREE_TERM_MINPCT_SPLIT                       .1
15         TREE_TERM_MINREC_NODE                       10
16        TREE_TERM_MINREC_SPLIT                       20

Computed Settings:
  setting name setting value
0    RFOR_MTRY             2

Global Statistics:
   attribute name  attribute value
0       AVG_DEPTH               2
1   AVG_NODECOUNT               3
2       MAX_DEPTH               2
3   MAX_NODECOUNT               2
4       MIN_DEPTH               2
5   MIN_NODECOUNT               2
6        NUM_ROWS             104

Attributes:
Petal_Length
Petal_Width
Sepal_Length

Partition: NO

Importance:

   ATTRIBUTE_NAME ATTRIBUTE_SUBNAME  ATTRIBUTE_IMPORTANCE
 0   Petal_Length              None              0.329971
 1    Petal_Width              None              0.296799
 2   Sepal_Length              None              0.037309
 3    Sepal_Width              None              0.000000

>>> # Use the model to make predictions on the test data.
... rf_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Petal_Length',
...                                                 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
 0            4.9          3.0           1.4      setosa      setosa
 1            4.9          3.1           1.5      setosa      setosa
 2            4.8          3.4           1.6      setosa      setosa
 3            5.8          4.0           1.2      setosa      setosa
...           ...          ...           ...         ...         ...
42            6.7          3.3           5.7   virginica   virginica
43            6.7          3.0           5.2   virginica   virginica
44            6.5          3.0           5.2   virginica   virginica
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
45            5.9          3.0           5.1   virginica   virginica

>>> # Return the prediction probability.
... rf_mod.predict(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Sepal_Width',
...                                                 'Species']],
...                proba = True)
     Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
 0            4.9          3.0      setosa      setosa     0.989130
 1            4.9          3.1      setosa      setosa     0.989130
 2            4.8          3.4      setosa      setosa     0.989130
 3            5.8          4.0      setosa      setosa     0.950000
...           ...          ...         ...         ...          ...
42            6.7          3.3   virginica   virginica     0.501016
43            6.7          3.0   virginica   virginica     0.501016
44            6.5          3.0   virginica   virginica     0.501016
45            5.9          3.0   virginica   virginica     0.501016

>>> # Return the top two most influencial attributes of the highest
... # probability class.
>>> rf_mod.predict_proba(test_dat.drop('Species'),
...                supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                 'Species']],
...                topN = 2).sort_values(by = ['Sepal_Length', 'Species'])
     Sepal_Length     Species       TOP_1  TOP_1_VAL       TOP_2  TOP_2_VAL
 0            4.4      setosa      setosa   0.989130  versicolor   0.010870
 1            4.4      setosa      setosa   0.989130  versicolor   0.010870
 2            4.5      setosa      setosa   0.989130  versicolor   0.010870
 3            4.8      setosa      setosa   0.989130  versicolor   0.010870
...           ...         ...         ...        ...         ...        ...
42            6.7   virginica   virginica   0.501016  versicolor   0.498984
43            6.9  versicolor   virginica   0.501016  versicolor   0.498984
44            6.9   virginica   virginica   0.501016  versicolor   0.498984
45            7.0  versicolor   virginica   0.501016  versicolor   0.498984

>>> rf_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.76087

>>> # Reset TREE_TERM_MAX_DEPTH and refit the model.
... rf_mod.set_params(tree_term_max_depth = '3').fit(train_x, train_y,
cost_matrix)

Algorithm Name: Random Forest

Mining Function: CLASSIFICATION

Target: Species
Settings:
                    setting name            setting value
0                      ALGO_NAME       ALGO_RANDOM_FOREST
1           CLAS_COST_TABLE_NAME     "OML_USER"."RF_COST"
2              CLAS_MAX_SUP_BINS                       32
3          CLAS_WEIGHTS_BALANCED                      OFF
4                   ODMS_DETAILS             ODMS_ENABLE
5   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
6               ODMS_RANDOM_SEED                         0
7                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
8                      PREP_AUTO                        ON
9                RFOR_NUM_TREES                        20
10           RFOR_SAMPLING_RATIO                        .5
11          TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
12           TREE_TERM_MAX_DEPTH                         3
13         TREE_TERM_MINPCT_NODE                       .05
14        TREE_TERM_MINPCT_SPLIT                        .1
15         TREE_TERM_MINREC_NODE                        10
16        TREE_TERM_MINREC_SPLIT                        20

Computed Settings:
  setting name setting value
0    RFOR_MTRY             2

Global Statistics:
   attribute name  attribute value
0       AVG_DEPTH                3
1   AVG_NODECOUNT                5
2       MAX_DEPTH                3
3   MAX_NODECOUNT                6
4       MIN_DEPTH                3
5   MIN_NODECOUNT                4
6        NUM_ROWS              104

Attributes:
Petal_Length
Petal_Width
Sepal_Length

Partition: NO

Importance:

  ATTRIBUTE_NAME ATTRIBUTE_SUBNAME  ATTRIBUTE_IMPORTANCE
0   Petal_Length              None              0.501022
1    Petal_Width              None              0.568170
2   Sepal_Length              None              0.091617
3    Sepal_Width              None              0.000000
```

10.18 Singular Value Decomposition

Use the `oml.svd` class to build a model for feature extraction.

The `oml.svd` class creates a model that uses the Singular Value Decomposition (SVD) algorithm for feature extraction. SVD performs orthogonal linear transformations that capture the underlying variance of the data by decomposing a rectangular matrix into three matrices: U, V, and D. Columns of matrix V contain the right singular vectors and columns of matrix U contain the left singular vectors. Matrix D is a diagonal matrix and its singular values reflect the amount of data variance captured by the bases.

The `SVDS_MAX_NUM_FEATURES` constant specifies the maximum number of features supported by SVD. The value of the constant is 2500.

Copyright © 2024, 2026, Oracle and/or its affiliates.

For information on the `oml.svd` class attributes and methods, invoke `help(oml.svd)` or see Oracle Machine Learning for Python API Reference.

Settings for a Singular Value Decomposition Model

Table 10-15 Singular Value Decomposition Model Settings

Setting Name	Setting Value	Description
FEAT_NUM_FEATURES	TO_CHAR(*numeric_expr* >=1)	The number of features to extract. The default value is estimated by the algorithm. If the matrix rank is smaller than this number, fewer features are returned.
SVDS_OVER_SAMPLING	Range [1, 5000].	Configures the number of columns in the sampling matrix used by the Stochastic SVD solver. The number of columns in this matrix is equal to the requested number of features plus the oversampling setting. TSVDS_SOLVER must be set to SVDS_SOLVER_SSVD or SVDS_SOLVER_STEIGEN.
SVDS_POWER_ITERATIONS	Range [0, 20].	Improves the accuracy of the SSVD solver. The default value is 2. SVDS_SOLVER must be set to SVDS_SOLVER_SSVD or SVDS_SOLVER_STEIGEN.
SVDS_RANDOM_SEED	Range [0 - 4,294,967,296]	The random seed value for initializing the sampling matrix used by the Stochastic SVD solver. The default value is 0. SVDS_SOLVER must be set to SVDS_SOLVER_SSVD or SVDS_SOLVER_STEIGEN
SVDS_SCORING_MODE	SVDS_SCORING_PCA SVDS_SCORING_SVD	Whether to use SVD or PCA scoring for the model. When the build data is scored with SVD, the projections are the same as the U matrix. When the build data is scored with PCA, the projections are the product of the U and D matrices. The default value is SVDS_SCORING_SVD.
SVDS_SOLVER	SVDS_SOLVER_STEIGEN SVDS_SOLVER_SSVD SVDS_SOLVER_TSEIGEN SVDS_SOLVER_TSSVD	Specifies the solver to be used for computing SVD of the data. For PCA, the solver setting indicates the type of SVD solver used to compute the PCA for the data. When this setting is not specified, the solver type selection is data driven. If the number of attributes is greater than 3240, then the default wide solver is used. Otherwise, the default narrow solver is selected. The following are the group of solvers: • Narrow data solvers: for matrices with up to 11500 attributes (TSEIGEN) or up to 8100 attributes (TSSVD). • Wide data solvers: for matrices up to 1 million attributes. For narrow data solvers: • Tall-Skinny SVD uses QR computation TSVD (SVDS_SOLVER_TSSVD) • Tall-Skinny SVD uses eigenvalue computation, TSEIGEN (SVDS_SOLVER_TSEIGEN), which is the default solver for narrow data. For wide data solvers: • Stochastic SVD uses QR computation SSVD (SVDS_SOLVER_SSVD), is the default solver for wide data solvers. • Stochastic SVD uses eigenvalue computations, STEIGEN (SVDS_SOLVER_STEIGEN).
SVDS_TOLERANCE	Range [0, 1]	Defines the minimum value for the eigenvalue of a feature as a share of the first eigenvalue to not prune. Use this setting to prune features. The default value is data driven.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-15 (Cont.) Singular Value Decomposition Model Settings

Setting Name	Setting Value	Description
SVDS_U_MATRIX_OUTPUT	SVDS_U_MATRIX_ENABLE SVDS_U_MATRIX_DISABLE	Specifies whether to persist the U matrix produced by SVD. The U matrix in SVD has as many rows as the number of rows in the build data. To avoid creating a large model, the U matrix is persisted only when SVDS_U_MATRIX_OUTPUT is enabled. When SVDS_U_MATRIX_OUTPUT is enabled, the build data must include a case ID. If no case ID is present and the U matrix is requested, then an exception is raised. The default value is SVDS_U_MATRIX_DISABLE.

See Also

- About Model Settings
- Shared Settings

Example 10-17 Using the oml.svd Class

This example uses some of the methods of the `oml.svd` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_dat = dat[0]
test_dat = dat[1]

# Create an SVD model object.
svd_mod = oml.svd(ODMS_DETAILS = 'ODMS_ENABLE')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Fit the model according to the training data and parameter
# settings.
svd_mod = svd_mod.fit(train_dat)

# Show the model details.
svd_mod

# Use the model to make predictions on the test data.
svd_mod.predict(test_dat,
                supplemental_cols = test_dat[:,
                                             ['Sepal_Length',
                                              'Sepal_Width',
                                              'Petal_Length',
                                              'Species']])

# Perform dimensionality reduction and return values for the two
# features that have the highest topN values.
svd_mod.transform(test_dat,
  supplemental_cols = test_dat[:, ['Sepal_Length']],
    topN = 2).sort_values(by = ['Sepal_Length',
                                'TOP_1',
                                'TOP_1_VAL'])
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]
>>>
>>> # Create an SVD model object.
... svd_mod = oml.svd(ODMS_DETAILS = 'ODMS_ENABLE')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>>
>>> # Fit the model according to the training data and parameter
... # settings.
>>> svd_mod = svd_mod.fit(train_dat)
>>>
>>> # Show the model details.
... svd_mod

Algorithm Name: Singular Value Decomposition

Mining Function: FEATURE_EXTRACTION

Settings:
               setting name               setting value
0                     ALGO_NAME  ALGO_SINGULAR_VALUE_DECOMP
1                   ODMS_DETAILS                 ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                          ON
5             SVDS_SCORING_MODE             SVDS_SCORING_SVD
6          SVDS_U_MATRIX_OUTPUT       SVDS_U_MATRIX_DISABLE

Computed Settings:
        setting name                     setting value
0  FEAT_NUM_FEATURES                                 8
1        SVDS_SOLVER               SVDS_SOLVER_TSEIGEN
2     SVDS_TOLERANCE  .00000000000002464695114667847 5

Global Statistics:
    attribute name  attribute value
0   NUM_COMPONENTS                8
1         NUM_ROWS              111
2 SUGGESTED_CUTOFF                1

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

Features:

   FEATURE_ID ATTRIBUTE_NAME ATTRIBUTE_VALUE     VALUE
0           1             ID            None  0.996297
1           1   Petal_Length            None  0.046646
2           1    Petal_Width            None  0.015917
3           1   Sepal_Length            None  0.063312
... ... ... ... ...
60          8   Sepal_Width             None -0.030620
61          8       Species          setosa  0.431543
62          8       Species      versicolor  0.566418
63          8       Species       virginica  0.699261
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
[64 rows x 4 columns]

D:

   FEATURE_ID      VALUE
0           1  886.737809
1           2   32.736792
2           3   10.043389
3           4    5.270496
4           5    2.708602
5           6    1.652340
6           7    0.938640
7           8    0.452170

V:

        '1'       '2'       '3'       '4'       '5'       '6'       '7' \
0  0.001332  0.156581 -0.317375  0.113462 -0.154414 -0.113058  0.799390
1  0.003692  0.052289  0.316295  0.733040  0.190746  0.022285 -0.046406
2  0.005267 -0.051498 -0.052111  0.527881 -0.066995  0.046461 -0.469396
3  0.015917  0.008741  0.263614  0.244811  0.460445  0.767503  0.262966
4  0.030208  0.550384 -0.358277  0.041807  0.689962 -0.261815 -0.143258
5  0.046646  0.189325  0.766663  0.326363  0.079611 -0.479070  0.177661
6  0.063312  0.790864  0.097964 -0.051230 -0.490804  0.312159 -0.131337
7  0.996297 -0.076079 -0.035940 -0.017429 -0.000960 -0.001908  0.001755
        '8'
0  0.431543
1  0.566418
2  0.699261
3  0.005000
4 -0.030620
5 -0.016932
6 -0.052185
7 -0.001415

>>> # Use the model to make predictions on the test data.
>>> svd_mod.predict(test_dat,
                    supplemental_cols = test_dat[:,
...                                              ['Sepal_Length',
...                                               'Sepal_Width',
...                                               'Petal_Length',
...                                               'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  FEATURE_ID
0            5.0          3.6           1.4    setosa           2
1            5.0          3.4           1.5    setosa           2
2            4.4          2.9           1.4    setosa           8
3            4.9          3.1           1.5    setosa           2
... ... ... ... ... ...
35           6.9          3.1           5.4 virginica           1
36           5.8          2.7           5.1 virginica           1
37           6.2          3.4           5.4 virginica           5
38           5.9          3.0           5.1 virginica           1

>>> # Perform dimensionality reduction and return values for the two
... # features that have the highest topN values.
>>> svd_mod.transform(test_dat,
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...   supplemental_cols = test_dat[:, ['Sepal_Length']],
...     topN = 2).sort_values(by = ['Sepal_Length',
...                                 'TOP_1',
...                                 'TOP_1_VAL'])
    Sepal_Length  TOP_1  TOP_1_VAL  TOP_2  TOP_2_VAL
0            4.4      7   0.153125      3  -0.130778
1            4.4      8   0.171819      2   0.147070
2            4.8      2   0.159324      6  -0.085194
3            4.8      7   0.157187      3  -0.141668
... ... ... ... ... ...
35           7.2      6  -0.167688      1   0.142545
36           7.2      7  -0.176290      6  -0.175527
37           7.6      4   0.205779      3   0.141533
38           7.9      8  -0.253194      7  -0.166967
```

10.19 Support Vector Machine

The `oml.svm` class creates a Support Vector Machine (SVM) model for classification, regression, or anomaly detection.

SVM is a powerful, state-of-the-art algorithm with strong theoretical foundations based on the Vapnik-Chervonenkis theory. SVM has strong regularization properties. Regularization refers to the generalization of the model to new data.

SVM models have a functional form similar to neural networks and radial basis functions, which are both popular machine learning techniques.

SVM can be used to solve the following problems:

- **Classification**: SVM classification is based on decision planes that define decision boundaries. A decision plane is one that separates a set of objects having different class memberships. SVM finds the vectors (“support vectors") that define the separators that give the widest separation of classes.

  SVM classification supports both binary and multiclass targets.
- **Regression**: SVM uses an epsilon-insensitive loss function to solve regression problems.

  SVM regression tries to find a continuous function such that the maximum number of data points lie within the epsilon-wide insensitivity tube. Predictions falling within epsilon distance of the true target value are not interpreted as errors.
- **Anomaly Detection**: Anomaly detection identifies unusual cases in data that is seemingly homogeneous. Anomaly detection is an important tool for detecting fraud, network intrusion, and other rare events that may have great significance but are hard to find.

  Anomaly detection is implemented as one-class SVM classification. An anomaly detection model predicts whether a data point is typical for a given distribution or not.

The `oml.svm` class builds each of these three different types of models. Some arguments apply to classification models only, some to regression models only, and some to anomaly detection models only.

For information on the `oml.svm` class attributes and methods, invoke `help(oml.svm)` or see Oracle Machine Learning for Python API Reference.

Support Vector Machine Model Settings

The following table lists settings for SVM models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-16 Support Vector Machine Settings

Setting Name	Setting Value	Description
CLAS_COST_TABLE_NAME	*table_name*	The name of a table that stores a cost matrix for the algorithm to use in scoring the model. The cost matrix specifies the costs associated with misclassifications. The cost matrix table is user-created. The following are the column requirements for the table. • Column Name: ACTUAL_TARGET_VALUE Data Type: Valid target data type • Column Name: PREDICTED_TARGET_VALUE Data Type: Valid target data type • Column Name: COST Data Type: NUMBER
CLAS_WEIGHTS_BALANCED	ON OFF	Indicates whether the algorithm must create a model that balances the target distribution. This setting is most relevant in the presence of rare targets, as balancing the distribution may enable better average accuracy (average of per-class accuracy) instead of overall accuracy (which favors the dominant class). The default value is OFF.
CLAS_WEIGHTS_TABLE_NAME	*table_name*	The name of a table that stores weighting information for individual target values in GLM logistic regression models. The weights are used by the algorithm to bias the model in favor of higher weighted classes. The class weights table is user-created. The following are the column requirements for the table. • Column Name: TARGET_VALUE Data Type: Valid target data type • Column Name: CLASS_WEIGHT Data Type: NUMBER
SVMS_BATCH_ROWS	Positive integer	Sets the size of the batch for the SGD solver. This setting applies to SVM models with linear kernel. An input of 0 triggers a data driven batch size estimate. The default value is 20000.
SVMS_COMPLEXITY_FACTOR	TO_CHAR(*numeric_expr* >0)	Regularization setting that balances the complexity of the model against model robustness to achieve good generalization on new data. SVM uses a data-driven approach to finding the complexity factor. Value of complexity factor for SVM algorithm (both Classification and Regression). Default value estimated from the data by the algorithm.
SVMS_CONV_TOLERANCE	TO_CHAR(*numeric_expr* >0)	Convergence tolerance for SVM algorithm. Default is 0.0001.
SVMS_EPSILON	TO_CHAR(*numeric_expr* >0)	Regularization setting for regression, similar to complexity factor. Epsilon specifies the allowable residuals, or noise, in the data. Value of epsilon factor for SVM regression. Default is 0.1.
SVMS_KERNEL_FUNCTION	SVMS_GAUSSIAN SVMS_LINEAR	Kernel for Support Vector Machine. Linear or Gaussian. The default value is SVMS_LINEAR.
SVMS_NUM_ITERATIONS	Positive integer	Sets an upper limit on the number of SVM iterations. The default is system determined because it depends on the SVM solver.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-16 (Cont.) Support Vector Machine Settings

Setting Name	Setting Value	Description
SVMS_NUM_PIVOTS	Range [1; 10000]	Sets an upper limit on the number of pivots used in the Incomplete Cholesky decomposition. It can be set only for non-linear kernels. The default value is 200.
SVMS_OUTLIER_RATE	TO_CHAR(*0< numeric_expr <1*)	The desired rate of outliers in the training data. Valid for One-Class SVM models only (Anomaly Detection). The default value is 0.01.
SVMS_REGULARIZER	SVMS_REGULARIZER_L1 SVMS_REGULARIZER_L2	Controls the type of regularization that the SGD SVM solver uses. The setting applies only to linear SVM models. The default value is system determined because it depends on the potential model size.
SVMS_SOLVER	SVMS_SOLVER_SGD (Sub-Gradient Descend) SVMS_SOLVER_IPM (Interior Point Method)	Allows the user to choose the SVM solver. The SGD solver cannot be selected if the kernel is non-linear. The default value is system determined.
SVMS_STD_DEV	TO_CHAR(*numeric_expr >0*)	Controls the spread of the Gaussian kernel function. SVM uses a data-driven approach to find a standard deviation value that is on the same scale as distances between typical cases. Value of standard deviation for SVM algorithm. This is applicable only for the Gaussian kernel. The default value is estimated from the data by the algorithm.

See Also

- About Model Settings
- Shared Settings

Example 10-18 Using the oml.svm Class

This example demonstrates the use of various methods of the `oml.svm` class. In the listing for this example, some of the output is not shown as indicated by ellipses.

```
import oml
import pandas as pd
from sklearn import datasets

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species']))

try:
    oml.drop('IRIS')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Create training and test data.
dat = oml.sync(table = 'IRIS').split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

# Create an SVM model object.
svm_mod = oml.svm('classification',
                  svms_kernel_function =
                  'dbms_data_mining.svms_linear')

# Fit the SVM Model according to the training data and parameter
# settings.
svm_mod.fit(train_x, train_y)

# Use the model to make predictions on test data.
svm_mod.predict(test_dat.drop('Species'),
                supplemental_cols = test_dat[:, ['Sepal_Length',
                                                 'Sepal_Width',
                                                 'Petal_Length',
                                                 'Species']])

# Return the prediction probability.
svm_mod.predict(test_dat.drop('Species'),
                supplemental_cols = test_dat[:, ['Sepal_Length',
                                                 'Sepal_Width',
                                                 'Species']],
                proba = True)
svm_mod.predict_proba(test_dat.drop('Species'),
  supplemental_cols = test_dat[:, ['Sepal_Length',
                                   'Sepal_Width',
                                   'Species']],
  topN = 1).sort_values(by = ['Sepal_Length', 'Sepal_Width'])

svm_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...                             2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> try:
...    oml.drop('IRIS')
... except:
...    pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Create training and test data.
... dat = oml.sync(table = 'IRIS').split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]
>>>
>>> # Create an SVM model object.
... svm_mod = oml.svm('classification',
...                   svms_kernel_function =
...                   'dbms_data_mining.svms_linear')
>>>
>>> # Fit the SVM model according to the training data and parameter
... # settings.
>>> svm_mod.fit(train_x, train_y)

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: Species

Settings:
           setting name                       setting value
0                     ALGO_NAME  ALGO_SUPPORT_VECTOR_MACHINES
1         CLAS_WEIGHTS_BALANCED                           OFF
2                  ODMS_DETAILS                   ODMS_ENABLE
3  ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
4                 ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
5                     PREP_AUTO                            ON
6           SVMS_CONV_TOLERANCE                         .0001
7          SVMS_KERNEL_FUNCTION                   SVMS_LINEAR

Computed Settings:
          setting name    setting value
0  SVMS_COMPLEXITY_FACTOR               10
1     SVMS_NUM_ITERATIONS               30
2             SVMS_SOLVER  SVMS_SOLVER_IPM

Global Statistics:
   attribute name  attribute value
0       CONVERGED              YES
1      ITERATIONS               14
2        NUM_ROWS              104

Attributes:
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

COEFFICIENTS:

   TARGET_VALUE ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE    COEF
0        setosa   Petal_Length              None            None -0.5809
1        setosa    Petal_Width              None            None -0.7736
2        setosa   Sepal_Length              None            None -0.1653
3        setosa    Sepal_Width              None            None  0.5689
4        setosa           None              None            None -0.7355
5    versicolor   Petal_Length              None            None  1.1304
6    versicolor    Petal_Width              None            None -0.3323
7    versicolor   Sepal_Length              None            None -0.8877
8    versicolor    Sepal_Width              None            None -1.2582
9    versicolor           None              None            None -0.9091
10    virginica   Petal_Length              None            None  4.6042
11    virginica    Petal_Width              None            None  4.0681
12    virginica   Sepal_Length              None            None -0.7985
13    virginica    Sepal_Width              None            None -0.4328
14    virginica           None              None            None -5.3180

>>> # Use the model to make predictions on test data.
... svm_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                  'Sepal_Width',
...                                                  'Petal_Length',
...                                                  'Species']])
    Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
0            4.9          3.0           1.4      setosa      setosa
1            4.9          3.1           1.5      setosa      setosa
2            4.8          3.4           1.6      setosa      setosa
3            5.8          4.0           1.2      setosa      setosa
...          ...          ...           ...         ...         ...
44           6.7          3.3           5.7   virginica   virginica
45           6.7          3.0           5.2   virginica   virginica
46           6.5          3.0           5.2   virginica   virginica
47           5.9          3.0           5.1   virginica   virginica

>>> # Return the prediction probability.
... svm_mod.predict(test_dat.drop('Species'),
...                 supplemental_cols = test_dat[:, ['Sepal_Length',
...                                                  'Sepal_Width',
...                                                  'Species']],
...                 proba = True)
    Sepal_Length  Sepal_Width     Species  PREDICTION  PROBABILITY
0            4.9          3.0      setosa      setosa     0.761886
1            4.9          3.1      setosa      setosa     0.805510
2            4.8          3.4      setosa      setosa     0.920317
3            5.8          4.0      setosa      setosa     0.998398
...          ...          ...         ...         ...          ...
44           6.7          3.3   virginica   virginica     0.927706
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
45           6.7          3.0   virginica   virginica     0.855353
46           6.5          3.0   virginica   virginica     0.799556
47           5.9          3.0   virginica   virginica     0.688024

>>> # Make predictions and return the probability for each class
... # on new data.
>>> svm_mod.predict_proba(test_dat.drop('Species'),
...   supplemental_cols = test_dat[:, ['Sepal_Length',
...                                    'Sepal_Width',
...                                    'Species']],
...   topN = 1).sort_values(by = ['Sepal_Length', 'Sepal_Width'])
    Sepal_Length  Sepal_Width     Species       TOP_1  TOP_1_VAL
0            4.4          3.0      setosa      setosa   0.698067
1            4.4          3.2      setosa      setosa   0.815643
2            4.5          2.3      setosa  versicolor   0.605105
3            4.8          3.4      setosa      setosa   0.920317
...          ...          ...         ...         ...        ...
44           6.7          3.3   virginica   virginica   0.927706
45           6.9          3.1  versicolor  versicolor   0.378391
46           6.9          3.1   virginica   virginica   0.881118
47           7.0          3.2  versicolor      setosa   0.586393

>>> svm_mod.score(test_dat.drop('Species'), test_dat[:, ['Species']])
0.895833
```

10.20 Non-Negative Matrix Factorization

The `oml.nmf` class creates a Non-Negative Matrix Factorization (NMF) model for feature extraction.

Each feature extracted by NMF is a linear combination of the original attribution set. Each feature has a set of non-negative coefficients, which are a measure of the weight of each attribute on the feature. If the argument `allow.negative.scores` is `TRUE`, then negative coefficients are allowed.

Settings for a Non-Negative Matrix Factorization Models

The following table lists settings that apply to Non-Negative Matrix Factorization models.

Table 10-17 Non-Negative Matrix Factorization Model Settings

Setting Name	Setting Value	Description
`NMFS_CONV_TOLERANCE`	`(0< numeric_expr <=0.5)`	Convergence tolerance for NMF algorithm Default is `0.05`
`NMFS_NONNEGATIVE_SCORING`	`NMFS_NONNEG_SCORING_ENABLE` `NMFS_NONNEG_SCORING_DISABLE`	Whether negative numbers should be allowed in scoring results. When set to `NMFS_NONNEG_SCORING_ENABLE`, negative feature values will be replaced with zeros. When set to `NMFS_NONNEG_SCORING_DISABLE`, negative feature values will be allowed. Default is `NMFS_NONNEG_SCORING_ENABLE`
`NMFS_NUM_ITERATIONS`	`(1 <= numeric_expr <=500)`	Number of iterations for NMF algorithm Default is `50`

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-17 (Cont.) Non-Negative Matrix Factorization Model Settings

Setting Name	Setting Value	Description
NMFS_RANDOM_SEED	(numeric_expr)	Random seed for NMF algorithm. Default is –1.

Example 10-19 Using the oml.nmf Class

This example creates an NMF model and uses some of the methods of the `oml.nmf` class.

```
import oml
import pandas as pd from sklearn import datasets
#For on-premises database follow the below command to connect to the database
oml.connect("<username>","<password>",dsn="dsn")

iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
'Petal_Length', 'Petal_Width'])
x.insert(0, "ID", range(1, len(x) + 1))
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
2:'virginica'}[x], iris.target)), columns = ['Species'])

z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training and test data sets.

train_dat, test_dat = oml.sync(table = "IRIS").split()

#Create a Non-Negative Matrix Factorization model using oml.nmf.

nmf_mod = oml.nmf()

#Fit the model to the training data.

nmf_mod = nmf_mod.fit(train_dat)

#Show the model details.

nmf_mod
#Use the model to make predictions on the test data, returning the
Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.

nmf_mod.predict(test_dat, supplemental_cols = test_dat[:, ['Sepal_Length',
'Sepal_Width', 'Petal_Length', 'Species']])

nmf_mod.transform(test_dat, supplemental_cols = test_dat[:,
['Sepal_Length']], topN = 2).sort_values(by = ['Sepal_Length', 'TOP_1',
'TOP_1_VAL'])

#Feature comparison

nmf_mod.feature_compare(test_dat, compare_cols = ["Sepal_Length",
"Petal_Length"], supplemental_cols = ["Species"])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
#Set new parameters and refit the model to produce U matrix output.

new_setting = {'nmfs_conv_tolerance':0.05}
nmf_mod2 = nmf_mod.set_params(**new_setting).fit(train_dat, case_id = "ID")
nmf_mod2
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets

>>> #For on-premises database follow the below command to connect to the
database
>>> oml.connect("<username>","<password>", dsn="<dsn>")

>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
'Petal_Length', 'Petal_Width'])
>>> x.insert(0, "ID", range(1, len(x) + 1))
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
2:'virginica'}[x], iris.target)), columns = ['Species'])

>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training and test data sets.

>>> dat = oml.sync(table = "IRIS").split()
>>> train_dat = dat[0]
>>> test_dat = dat[1]

#Create a Non-Negative Matrix Factorization model using oml.nmf.

>>> nmf_mod = oml.nmf()

#Fit the model to the training data.

>>> nmf_mod = nmf_mod.fit(train_dat)

#Show the model details.

>>> nmf_mod

Algorithm Name: Non-Negative Matrix Factorizationx

Mining Function: FEATURE_EXTRACTION

Settings:
                   setting name                    setting value
0                     ALGO_NAME  ALGO_NONNEGATIVE_MATRIX_FACTOR
1           NMFS_CONV_TOLERANCE                              .05
2      NMFS_NONNEGATIVE_SCORING      NMFS_NONNEG_SCORING_ENABLE
3           NMFS_NUM_ITERATIONS                              50
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
4              NMFS_RANDOM_SEED                              -1
5                  ODMS_DETAILS                      ODMS_ENABLE
6  ODMS_MISSING_VALUE_TREATMENT          ODMS_MISSING_VALUE_AUTO
7                 ODMS_SAMPLING            ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                              ON

Computed Settings:
              setting name setting value
0        FEAT_NUM_FEATURES             2
1      NMFS_NUM_ITERATIONS             2
2  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     CONV_ERROR       0.0444448
2     ITERATIONS               2
3       NUM_ROWS             111
4    SAMPLE_SIZE             111

Attributes:
ID
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

H:

    FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
0            1             1             ID            None     0.581551
1            1             1   Petal_Length            None     0.355323
2            1             1    Petal_Width            None     0.158492
3            1             1   Sepal_Length            None     0.656558
4            1             1    Sepal_Width            None     0.424101
5            1             1        Species          setosa     0.089560
6            1             1        Species      versicolor     0.534806
7            1             1        Species       virginica     0.539590
8            2             2             ID            None     0.344647
9            2             2   Petal_Length            None     0.506623
10           2             2    Petal_Width            None     0.650077
11           2             2   Sepal_Length            None     0.170237
12           2             2    Sepal_Width            None     0.248640
13           2             2        Species          setosa     0.249221
14           2             2        Species      versicolor     0.042316
15           2             2        Species       virginica     0.093861

W:

    FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
0            1             1             ID            None     0.288559
1            1             1   Petal_Length            None    -0.062579
2            1             1    Petal_Width            None    -0.370128
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
3            1            1   Sepal_Length            None     0.502382
4            1            1    Sepal_Width            None     0.212611
5            1            1        Species      versicolor     0.486970
6            1            1        Species          setosa    -0.113835
7            1            1        Species       virginica     0.450038
8            2            2             ID            None     0.119462
9            2            2   Petal_Length            None     0.578697
10           2            2    Petal_Width            None     0.982575
11           2            2   Sepal_Length            None    -0.238993
12           2            2    Sepal_Width            None     0.082511
13           2            2        Species          setosa     0.353453
14           2            2        Species      versicolor    -0.359264
15           2            2        Species       virginica    -0.275074


#Use the model to make predictions on the test data, returning the
Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.

>>> nmf_mod.predict(test_dat, supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length     Species  FEATURE_ID
 0            5.0          3.6           1.4      setosa           2
 1            5.0          3.4           1.5      setosa           2
 2            4.4          2.9           1.4      setosa           2
 3            4.9          3.1           1.5      setosa           2
...           ...          ...           ...         ...         ...
 35           6.9          3.1           5.4   virginica           2
 36           5.8          2.7           5.1   virginica           2
 37           6.2          3.4           5.4   virginica           2
 38           5.9          3.0           5.1   virginica           2

#Transform

>>> nmf_mod.transform(test_dat, supplemental_cols = test_dat[:,
['Sepal_Length']], topN = 2).sort_values(by = ['Sepal_Length', 'TOP_1',
'TOP_1_VAL'])
     Sepal_Length  TOP_1  TOP_1_VAL  TOP_2  TOP_2_VAL
 0            4.4      2   0.464041      1   0.000000
 1            4.4      2   0.482051      1   0.045518
 2            4.8      2   0.475169      1   0.083874
 3            4.8      2   0.510372      1   0.101880
...           ...    ...        ...    ...        ...
 35           7.2      1   0.915012      2   0.850330
 36           7.2      1   0.938112      2   0.745207
 37           7.6      2   0.980757      1   0.864508
 38           7.9      1   1.048287      2   0.947744

#Feature comparison

>>> nmf_mod.feature_compare(test_dat, compare_cols = ["Sepal_Length",
"Petal_Length"], supplemental_cols = ["Species"])
      Species_A  Species_B  SIMILARITY
 0       setosa     setosa    0.990134
 1       setosa     setosa    0.929516
 2       setosa     setosa    0.976885
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
 3       setosa     setosa    0.953770
...         ...        ...         ...
 737  virginica  virginica    0.849758
 738  virginica  virginica    0.944063
 739  virginica  virginica    0.983637
 740  virginica  virginica    0.958018

[741 rows x 3 columns]

#Set new parameters and refit tthe model to produce U matrix output.

>>> new_setting = {'nmfs_conv_tolerance':0.05}
>>> nmf_mod2 = nmf_mod.set_params(**new_setting).fit(train_dat, case_id =
"ID")
>>> nmf_mod2

Algorithm Name: Non-Negative Matrix Factorizationx

Mining Function: FEATURE_EXTRACTION

Settings:
                   setting name                    setting value
0                     ALGO_NAME  ALGO_NONNEGATIVE_MATRIX_FACTOR
1           NMFS_CONV_TOLERANCE                             0.05
2      NMFS_NONNEGATIVE_SCORING      NMFS_NONNEG_SCORING_ENABLE
3           NMFS_NUM_ITERATIONS                              50
4              NMFS_RANDOM_SEED                              -1
5                  ODMS_DETAILS                    ODMS_ENABLE
6  ODMS_MISSING_VALUE_TREATMENT         ODMS_MISSING_VALUE_AUTO
7                 ODMS_SAMPLING           ODMS_SAMPLING_DISABLE
8                     PREP_AUTO                              ON

Computed Settings:
              setting name setting value
0        FEAT_NUM_FEATURES             2
1      NMFS_NUM_ITERATIONS             8
2  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name attribute value
0      CONVERGED             YES
1     CONV_ERROR       0.0277253
2     ITERATIONS               8
3       NUM_ROWS             111
4    SAMPLE_SIZE             111

Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width
Species

Partition: NO

H:
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
    FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE   COEFFICIENT
0            1             1   Petal_Length            None  9.889792e-02
1            1             1    Petal_Width            None  1.060984e-01
2            1             1   Sepal_Length            None  1.947197e-01
3            1             1    Sepal_Width            None  5.099539e-01
4            1             1        Species          setosa  7.507257e-01
5            1             1        Species      versicolor  5.773815e-03
6            1             1        Species       virginica  8.136382e-02
7            2             2   Petal_Length            None  6.652922e-01
8            2             2    Petal_Width            None  6.571416e-01
9            2             2   Sepal_Length            None  5.702848e-01
10           2             2    Sepal_Width            None  2.420062e-01
11           2             2        Species          setosa  1.643131e-08
12           2             2        Species      versicolor  5.158020e-01
13           2             2        Species       virginica  4.948837e-01

W:

    FEATURE_ID  FEATURE_NAME ATTRIBUTE_NAME ATTRIBUTE_VALUE  COEFFICIENT
0            1             1   Petal_Length            None    -0.071259
1            1             1    Petal_Width            None    -0.059774
2            1             1   Sepal_Length            None     0.077608
3            1             1    Sepal_Width            None     0.571981
4            1             1        Species      versicolor    -0.144686
5            1             1        Species          setosa     0.947005
6            1             1        Species       virginica    -0.043170
7            2             2   Petal_Length            None     0.392684
8            2             2    Petal_Width            None     0.385395
9            2             2   Sepal_Length            None     0.304214
10           2             2    Sepal_Width            None     0.003195
11           2             2        Species          setosa    -0.221185
12           2             2        Species      versicolor     0.325338
13           2             2        Species       virginica     0.289804
```

10.21 Exponential Smoothing Method

The `oml.esm` function uses the Exponential Smoothing Method (ESM) algorithm to create a time series model.

Exponential Smoothing Methods have been widely used in forecasting for over half a century. It has applications at the strategic, tactical, and operation level. For example, at a strategic level, forecasting is used for projecting return on investment, growth and the effect of innovations. At a tactical level, forecasting is used for projecting costs, inventory requirements, and customer satisfaction. At an operational level, forecasting is used for setting targets and predicting quality and conformance with standards.

In its simplest form, Exponential Smoothing is a moving average method with a single parameter that models an exponentially decreasing effect of past levels on future values. With a variety of extensions, Exponential Smoothing covers a broader class of models than other well-known approaches, such as the Box-Jenkins auto-regressive integrated moving average (ARIMA) approach. Oracle Machine Learning implements Exponential Smoothing using a state-of-the-art state space method that incorporates a single source of error (SSOE) assumption that provides theoretical and performance advantages.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Multiple time series is a convenience operation for constructing input to a time series regression. Multiple time series builds multiple time series models with a common time interval for use as input to a time series regression. One of the time series models is identified as the target time series of interest. For more information about Multiple Time Series Models, see Oracle Machine Learning for SQL Concepts Guide.

The behavior of Exponential Smoothing is modified such that it searches for an acceptable time series model automatically. If you do not specify a model type (`EXSM_MODEL`), the default behavior is for the algorithm to automatically determine the model type. For more information, see Oracle Machine Learning for SQL Concepts Guide.

Settings for an ESM model

The following table lists settings for ESM models.

Table 10-18 ESM Model Settings

Setting Name	Setting Value	Description
`EXSM_MODEL`	It can take value in set *{EXSM_SIMPLE, EXSM_SIMPLE_MULT, EXSM_HOLT, EXSM_HOLT_DMP, EXSM_MUL_TRND, EXSM_MULTRD_DMP, EXSM_SEAS_ADD, EXSM_SEAS_MUL, EXSM_HW, EXSM_HW_DMP, EXSM_HW_ADDSEA, EXSM_DHW_ADDSEA, EXSM_HWMT, EXSM_HWMT_DMP}*	This setting specifies the model. `EXSM_SIMPLE`: Simple exponential smoothing model is applied. `EXSM_SIMPLE_MULT`: Simple exponential smoothing model with multiplicative error is applied. `EXSM_HOLT`: Holt linear exponential smoothing model is applied. `EXSM_HOLT_DMP`: Holt linear exponential smoothing model with damped trend is applied. `EXSM_MUL_TRND`: Exponential smoothing model with multiplicative trend is applied. `EXSM_MULTRD_DMP`: Exponential smoothing model with multiplicative damped trend is applied. `EXSM_SEAS_ADD`: Exponential smoothing with additive seasonality, but no trend, is applied. `EXSM_SEAS_MUL`:Exponential smoothing with multiplicative seasonality, but no trend, is applied. `EXSM_HW`: Holt-Winters triple exponential smoothing model, additive trend, multiplicative seasonality is applied. `EXSM_HW_DMP`: Holt-Winters multiplicative exponential smoothing model with damped trend, additive trend, multiplicative seasonality is applied. `EXSM_HW_ADDSEA`: Holt-Winters additive exponential smoothing model, additive trend, additive seasonality is applied. `EXSM_DHW_ADDSEA`: Holt-Winters additive exponential smoothing model with damped trend, additive trend, additive seasonality is applied. `EXSM_HWMT`: Holt-Winters multiplicative exponential smoothing model with multiplicative trend, multiplicative trend, multiplicative seasonality is applied. `EXSM_HWMT_DMP`: Holt-Winters multiplicative exponential smoothing model with damped multiplicative trend, multiplicative trend, multiplicative seasonality is applied. The default value is `EXSM_SIMPLE`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_SEASONALITY	positive integer > 1	This setting specifies a positive integer value as the length of seasonal cycle. The value specified must be larger than 1. For example, setting value 4 means that every group of four observations forms a seasonal cycle. This setting is only applicable and must be provided for models with seasonality, otherwise the model throws an error. When EXSM_INTERVAL is not set, this setting applies to the original input time series. When EXSM_INTERVAL is set, this setting applies to the accumulated time series.
EXSM_INTERVAL	It can take value in set *{EXSM_INTERVAL_YEAR, EXSM_INTERVAL_QTR, EXSM_INTERVAL_MONTH,EXSM_INTERVAL_WEEK, EXSM_INTERVAL_DAY, EXSM_INTERVAL_HOUR, EXSM_INTERVAL_MIN,EXSM_INTERVAL_SEC}*	This setting only applies and must be provided when the time column (case_id column) has datetime type. It specifies the spacing interval of the accumulated equally spaced time series. The model throws an error if the time column of input table is of datetime type and setting EXSM_INTERVAL is not provided. The model throws an error if the time column of input table is of oracle number type and setting EXSM_INTERVAL is provided.
EXSM_ACCUMULATE	It can take value in set *{EXSM_ACCU_TOTAL, EXSM_ACCU_STD, EXSM_ACCU_MAX, EXSM_ACCU_MIN, EXSM_ACCU_AVG, EXSM_ACCU_MEDIAN, EXSM_ACCU_COUNT}*	This setting only applies and must be provided when the time column has datetime type. It specifies how to generate the value of the accumulated time series from the input time series.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_SETMISSING	It can also specify an option taking value in set *{EXSM_MISS_MIN, EXSM_MISS_MAX, EXSM_MISS_AVG, EXSM_MISS_MEDIAN, EXSM_MISS_LAST, EXSM_MISS_FIRST, EXSM_MISS_PREV, EXSM_MISS_NEXT, EXSM_MISS_AUTO}*.	This setting specifies how to handle missing values, which may come from input data and/or the accumulation process of time series. You can specify either a number or an option. If a number is specified, all the missing values are set to that number. EXSM_MISS_MIN: Replaces missing value with minimum of the accumulated time series. EXSM_MISS_MAX: Replaces missing value with maximum of the accumulated time series. EXSM_MISS_AVG: Replaces missing value with average of the accumulated time series. EXSM_MISS_MEDIAN: Replaces missing value with median of the accumulated time series. EXSM_MISS_LAST: Replaces missing value with last non-missing value of the accumulated time series. EXSM_MISS_FIRST: Replaces missing value with first non-missing value of the accumulated time series. EXSM_MISS_PREV:Replaces missing value with the previous non-missing value of the accumulated time series. EXSM_MISS_NEXT:Replaces missing value with the next non-missing value of the accumulated time series. EXSM_MISS_AUTO:EXSM model treats the input data as an irregular (non-uniformly spaced) time series. If this setting is not provided, EXSM_MISS_AUTO is the default value. In such a case, the model treats the input time series as irregular time series, viewing missing values as gaps.
EXSM_PREDICTION_STEP	It must be set to a number between 1-30.	This setting specifies how many steps ahead the predictions are to be made. If it is not set, the default value is 1: the model gives one-step-ahead prediction. A value greater than 30 results in an error.
EXSM_CONFIDENCE_LEVEL	It must be a number between 0 and 1, exclusive.	This setting specifies the desired confidence level for prediction. The lower and upper bounds of the specified confidence interval is reported. If this setting is not specified, the default confidence level is 95%.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_OPT_CRITERION	It takes value in set *{EXSM_OPT_CRIT_LIK, EXSM_OPT_CRIT_MSE, EXSM_OPT_CRIT_AMSE, EXSM_OPT_CRIT_SIG, EXSM_OPT_CRIT_MAE}*.	This setting specifies the desired optimization criterion. The optimization criterion is useful as a diagnostic for comparing models' fit to the same data. EXSM_OPT_CRIT_LIK: Minus twice the log-likelihood of a model. EXSM_OPT_CRIT_MSE: Mean square error of a model. EXSM_OPT_CRIT_AMSE: Average mean square error over user-specified time window. EXSM_OPT_CRIT_SIG: Model's standard deviation of residuals. EXSM_OPT_CRIT_MAE: Mean absolute error of a model. The default value is EXSM_OPT_CRIT_LIK.
EXSM_NMSE	positive integer	This setting specifies the length of the window used in computing the error metric average mean square error (AMSE).

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_SERIES_LIST	Comma delimited list of time series columns	This setting allows you to forecast up to twenty predictor series in addition to the target series. The column names in EXSM_SERIES_LIST are enclosed in single quotes. It is important to note that the list is enclosed in single quotes, not the individual column names. For example: `INSERT INTO <settings_table_name VALUES(dbms_data_mining.exsm_series_list, '<column1>,<column2>,<column3>,<column4>' );` For the prefix DM$ to be added to the build and scoring data sets, column names must be less than 125 characters long.

Note

Available only in Oracle AI Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-18 (Cont.) ESM Model Settings

Setting Name	Setting Value	Description
EXSM_INITVL_OPTIMIZE	EXSM_INITVL_OPTIMIZE_ENABLE EXSM_INITVL_OPTIMIZE_DISABLE	The setting EXSM_INITVL_OPTIMIZE determines whether initial values are optimized during model build. The default value is EXSM_INITVL_OPTIMIZE_ENABLE. **Note** EXSM_INITVL_OPTIMIZE can only be set to EXSM_INITVL_OPTIMIZE_DISABLE if the user has set EXSM_MODEL to EXSM_HW or EXSM_HW_ADDSEA. If EXSM_MODEL is set to another model type or is not specified, error 40213 (conflicting settings) is thrown and the model is not built.

Note

Available only in Oracle AI Database 26ai.

Example 10-20 Using the oml.esm Class

This example creates an ESM model and uses some of the methods of the `oml.esm` class.

```
import oml
import pandas as pd

df = pd.DataFrame({'EVENT': ['A', 'B', 'C', 'D'],
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                    'START': ['2021-10-04 13:29:00', '2021-10-07 12:30:00',
                              '2021-10-15 04:20:00', '2021-10-18 15:45:03'],
                    'END':   ['2021-10-08 11:29:06', '2021-10-15 10:30:07',
                              '2021-10-29 05:50:15', '2021-10-22 15:40:03']})

df['START'] = pd.to_datetime(df['START'])
df['END'] = pd.to_datetime(df['END'])
df['DURATION'] = df['END'] - df['START']
df['HOURS'] = df['DURATION'] / pd.Timedelta(hours=1)
df['MINUTES'] = df['DURATION'] / pd.Timedelta(minutes=1)
#For on-premises database follow the below command to connect to the database#
oml.connect("<username>","<password>", dsn="<dsn>")
dat = oml.create(df, table='DF')
train_x = dat[:, 1]
train_y = dat[:, 4]

setting = {'EXSM_INTERVAL':'EXSM_INTERVAL_DAY'}
esm_mod = oml.esm(**setting).fit(train_x, train_y, time_seq = 'START')

esm_mod
train_x = dat[:, 4]
train_y = dat[:, 5]
esm_mod = oml.esm().fit(train_x, train_y, time_seq = 'HOURS')

esm_mod
```

Listing for This Example

```
Create pandas DataFrame with start and end dates for an event. Convert start
and end date columns to datetime, and create new columns that contain
timedelta between the start and end dates. Convert timedelta into total
number of hours and convert timedelta into total number of minutes.

>>> import oml
>>> import pandas as pd

>>> df = pd.DataFrame({'EVENT': ['A', 'B', 'C', 'D'],
                    'START': ['2021-10-04 13:29:00', '2021-10-07 12:30:00',
                              '2021-10-15 04:20:00', '2021-10-18 15:45:03'],
                    'END':   ['2021-10-08 11:29:06', '2021-10-15 10:30:07',
                              '2021-10-29 05:50:15', '2021-10-22 15:40:03']})

>>> df['START'] = pd.to_datetime(df['START'])
>>> df['END'] = pd.to_datetime(df['END'])
>>> df['DURATION'] = df['END'] - df['START']
>>> df['HOURS'] = df['DURATION'] / pd.Timedelta(hours=1)
>>> df['MINUTES'] = df['DURATION'] / pd.Timedelta(minutes=1)

>>> #For on-premises database follow the below command to connect to the
database#
>>> oml.connect("<username>","<password>", dsn="<dsn>")
>>> dat = oml.create(df, table='DF')


Using Datetime type
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> train_x = dat[:, 1]
>>> train_y = dat[:, 4]

>>> setting = {'EXSM_INTERVAL':'EXSM_INTERVAL_DAY'}
>>> esm_mod = oml.esm(**setting).fit(train_x, train_y, time_seq = 'START')

>>> esm_mod

Algorithm Name: Exponential Smoothing

Mining Function: TIME_SERIES

Target: HOURS

Settings:
                     setting name               setting value
0                       ALGO_NAME  ALGO_EXPONENTIAL_SMOOTHING
1                EXSM_ACCUMULATE             EXSM_ACCU_TOTAL
2           EXSM_CONFIDENCE_LEVEL                         .95
3                  EXSM_INTERVAL           EXSM_INTERVAL_DAY
4                       EXSM_NMSE                           3
5          EXSM_OPTIMIZATION_CRIT           EXSM_OPT_CRIT_LIK
6            EXSM_PREDICTION_STEP                           1
7                 EXSM_SETMISSING             EXSM_MISS_AUTO
8                      ODMS_BOXCOX          ODMS_BOXCOX_ENABLE
9                     ODMS_DETAILS                ODMS_ENABLE
10  ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
11                  ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
12                      PREP_AUTO                          ON

Computed Settings:
  setting name setting value
0   EXSM_MODEL   EXSM_SIMPLE

Global Statistics:
       attribute name attribute value
0   -2 LOG-LIKELIHOOD        -21.1618
1                 AIC         48.3236
2                AICC            None
3               ALPHA     0.000100034
4          ALPHA DISC          0.9999
5                AMSE         12175.3
6                 BIC         46.4825
7           CONVERGED             YES
8       INITIAL ALPHA     0.000100034
9       INITIAL LEVEL         179.353
10                MAE          84.403
11                MSE          9843.9
12           NUM_ROWS               4
13              SIGMA         140.313
14                STD         140.313

Attributes:

Partition: NO
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Prediction:

    TIME_SEQ       VALUE  PREDICTION      LOWER       UPPER
0 2021-10-04   94.001667  179.352705        NaN         NaN
1 2021-10-07  190.001944  179.344167        NaN         NaN
2 2021-10-15  337.504167  179.345233        NaN         NaN
3 2021-10-18   95.916667  179.361069        NaN         NaN
4 2021-10-19         NaN  179.352712 -95.656158  454.361582


Using Float type

>>> train_x = dat[:, 4]
>>> train_y = dat[:, 5]
>>> esm_mod = oml.esm().fit(train_x, train_y, time_seq = 'HOURS')

>>> esm_mod

Algorithm Name: Exponential Smoothing

Mining Function: TIME_SERIES

Target: MINUTES

Settings:
                     setting name               setting value
0                       ALGO_NAME  ALGO_EXPONENTIAL_SMOOTHING
1          EXSM_CONFIDENCE_LEVEL                         .95
2                       EXSM_NMSE                           3
3         EXSM_OPTIMIZATION_CRIT          EXSM_OPT_CRIT_LIK
4           EXSM_PREDICTION_STEP                           1
5                EXSM_SETMISSING             EXSM_MISS_AUTO
6                    ODMS_BOXCOX          ODMS_BOXCOX_ENABLE
7                   ODMS_DETAILS                ODMS_ENABLE
8   ODMS_MISSING_VALUE_TREATMENT     ODMS_MISSING_VALUE_AUTO
9                  ODMS_SAMPLING       ODMS_SAMPLING_DISABLE
10                     PREP_AUTO                          ON

Computed Settings:
  setting name setting value
0   EXSM_MODEL     EXSM_HOLT

Global Statistics:
       attribute name attribute value
0   -2 LOG-LIKELIHOOD         4.47424
1                 AIC         1.05153
2                AICC            None
3               ALPHA     0.000104161
4                AMSE       0.0190133
5                BETA     0.000104153
6                 BIC          -2.017
7           CONVERGED             YES
8       INITIAL LEVEL         8.00977
9       INITIAL TREND        0.452033
10             LAMBDA     4.08563e-05
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
11                MAE         1175.53
12                MSE       0.0266914
13           NUM_ROWS               4
14              SIGMA        0.188649
15                STD        0.188649

Attributes:

Partition: NO

Prediction:

   TIME_SEQ         VALUE    PREDICTION        LOWER         UPPER
0        94   5640.100000   4807.666451          NaN           NaN
1        95   5755.000000   7554.329741          NaN           NaN
2       190  11400.116667  11869.239245          NaN           NaN
3       337  20250.250000  18649.004898          NaN           NaN
4       338           NaN  29301.840039  19894.31833  41663.104953
```

10.22 XGBoost

The `oml.xgb` class supports the in-database scalable gradient tree boosting algorithm for both classification, regression specifications, ranking models, and survival models. It makes available the open source gradient boosting framework. It prepares the categorical encoding and missing value replacement from the OML infrastructure, calls the in-database XGBoost, builds and persists a model as a first-class database model object, and supports using the model for prediction.

You can use `oml.xgb` as a stand-alone predictor or incorporate it into real-world production pipelines for a wide range of problems such as ad click-through rate prediction, hazard risk prediction, web text classification, and so on.

The `oml.xgb` algorithm takes three types of parameters: general parameters, booster parameters, and task parameters. You set the parameters through the model settings. The algorithm supports most of the settings of the open source XGBoost project. For more information on the supported settings, see XGBoost parameters.

Through `oml.xgb`, OML4Py supports a number of different classification and regression specifications, ranking models, and survival models. Binary and multi-class models are supported under the classification machine learning technique while regression, ranking, count, and survival are supported under the regression machine learning technique.

`oml.xgb` also supports partitioned models and internalizes the data preparation.

XG Boost feature interaction constraints allow users to specify which variables can and cannot interact. By focusing on key interactions and eliminating noise, it aids in improving predicting performance. This, in turn, may lead to more generalized predictions. For more information about XG Boost feature interaction constraints, see Oracle Machine Learning for SQL Concepts Guide.

Settings for an XGBoost model

The following table lists settings that apply to XGBoost models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 XGBoost Model Settings

Setting Name	Setting Value	Description
`xgboost_booster`	A string that is one of the following: • `dart` • `gblinear` • `gbtree`	The booster to use: • `dart` • `gblinear` • `gbtree` The `dart` and `gbtree` boosters use tree-based models whereas `gblinear` uses linear functions. The default value is `gbtree`.
`xgboost_num_round`	A non-negative integer.	The number of rounds for boosting. The default value is `10`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
`xgboost_interaction_constraints`	`[[x0,x1,x2],[x0,x4],[x5,x6]]` for example, `xn` are feature names or columns **Note** Available only in Oracle AI Database 26ai.	This setting specifies permitted interactions in the model. Specify the constraints in the form of a nested list where each inner list is a group of features (column names) that are allowed to interact with each other. If a single column is passed in the interactions then, the input is ignored. Here, features x0, x1, and x2 are allowed to interact with each other but with no other feature. Similarly, x0 and x4 are allowed to interact with each other but with no other feature and so on. This setting is applicable to 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
xgboost_decrease_constraints	[x0,x1],[x4,x5]	This setting specifies the features (column names) that must obey the decreasing constraint. The feature names are separated by a comma. For example, setting value 'x4,x5' sets decreasing constraint on features x4 and x5. This setting applies to numeric columns and 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.

Note

Available only in Oracle AI Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
xgboost_increase_constraints	[x0,x1],[x0,x3]	This setting specifies the features (column names) that must obey the increasing constraint. The feature names are separated by a comma. For example, setting value 'x0,x3' sets increasing constraint on features x0 and x3. This setting is applicable to 2-Dimensional features. An error occurs if you pass columns of non-supported type and non-existing feature names.

Note

Available only in Oracle AI Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
`objective`	For a classification model, a string that is one of the following: • `binary:hinge` • `binary:logistic` • `multi:softmax` • `multi:softprob` For a regression model, a string that is one of the following: • `binary:logitraw` • `count:poisson` • `rank:map` • `rank:ndcg` • `rank:pairwise` • `reg:gamma` • `reg:logistic` • `reg:tweedie` • `survival:aft` • `survival:cox` • `reg:squarederror` • `reg:squaredlogerror` **Note** Available only in Oracle AI Database 26ai.	**Settings for a Classification model:** • `binary:hinge:` Hinge loss for binary classification. This setting makes predictions of 0 or 1, rather than producing probabilities. • `binary:logistic:` Logistic regression for binary classification. The output is the probability. • `multi:softmax:` Performs multiclass classification using the `softmax` objective; you must also set `num_class(number_of_classes)`. • `multi:softprob: :` Same as `softmax`, except the output is a vector of `ndata * nclass`, which can be further reshaped to an `ndata * nclass` matrix. The result contains the predicted probability of each data point belonging to each class. The default `objective` value for classification is `multi:softprob`. **Settings for a Regression model:** • `binary:logitraw:` Logistic regression for binary classification; the output is the score before logistic transformation. • `count:poisson:` Poisson regression for count data; the output is the mean of the Poisson distribution. The `max_delta_step` value is set to 0.7 by default in Poisson regression to safeguard optimization. • `rank:map:` Using `LambdaMART`, performs list-wise ranking in which the Mean Average Precision (MAP) is maximized. • `rank:ndcg:` Using `LambdaMART`, performs list-wise ranking in which the Normalized Discounted Cumulative Gain (NDCG) is maximized. • `rank:pairwise:` Performs ranking by minimizing the pairwise loss. • `reg:gamma:` Gamma regression with log-link; the output is the mean of the gamma distribution. This setting might be useful for any outcome that might be gamma-distributed, such as modeling insurance claims severity. • `reg:logistic:` Logistic regression. • `reg:tweedie:` Tweedie regression with log-link. This setting might be useful for any outcome that might be Tweedie-distributed, such as modeling total loss in insurance. • `survival:aft:` Applies the Accelerated Failure Time (AFT) model for censored survival time data. When you select this option, `eval_metric` uses `aft-nloglik` as the default value. • `survival:cox:` Cox regression for right-censored survival time data (negative values are considered right-censored). Predictions are returned on the hazard ratio scale (that is, as `HR =`

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
		`exp(marginal_prediction)` in the proportional hazard function `h(t) = h0(t) * HR`). • `reg:squarederror`: Regression with squared loss. • `reg:squaredlogerror`: Regression with squared log loss. All input labels must be greater than -1. The default `objective` value for regression is `reg:squarederror`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
`xgboost_aft_loss_distribution`	[normal, logistic, extreme]	Specifies the distribution of the Z term in the AFT model. It specifies the Probabilty Density Function used by `survival:aft` objective and `aft-nloglik` evaluation metric. The default value is `normal`.

Note

Available only in Oracle AI Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
`xgboost_aft_loss_distribution_scale`	A positive number	Specifies the scaling factor σ, which scales the size of Z term in the AFT model. The default value is 1.

Note

Available only in Oracle AI Database 26ai.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 10-19 (Cont.) XGBoost Model Settings

Setting Name	Setting Value	Description
`xgboost_aft_right_bound_column_name`	column_name	Specifies the column containing the right bounds of the labels for an AFT model. You cannot select this parameter for a non-AFT model.

Note

Available only in Oracle AI Database 26ai.

Note

Oracle Machine Learning does not support `BOOLEAN` values for this setting.

For more information on the booster settings, see XGBoost parameters

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 10-21 Using the oml.xgb Class

This example creates an XGB model and uses some of the methods of the `oml.xgb` class.

```
#Load the iris data from sklearn and combine the target and predictors into a
single DataFrame, which matches the form of a database table.
Use the oml.create function to load this Pandas DataFrame into the databae,
which creates a persistent table and returns a proxy object that you assign
to z.#

import oml
from sklearn import datasets
import pandas as pd

iris = datasets.load_iris()
x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
'Petal_Length', 'Petal_Width'])
y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
2:'virginica'}[x], iris.target)), columns = ['Species'])

#For on-premises database follow the below command to connect to the database#
oml.connect("<username>","<password>", dsn="<dsn>")
z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training data and test data.#

dat = oml.sync(table = "IRIS").split()
train_x = dat[0].drop('Species')
train_y = dat[0]['Species']
test_dat = dat[1]

#Classification Example:#

#Create an XGBoost model object.#

setting = {'xgboost_max_depth': '3',
...            'xgboost_eta': '1',
...            'xgboost_num_round': '10'}

xgb_mod = oml.xgb('classification', **setting)

#Fit the XGBoost model to the training data.#

xgb_mod.fit(train_x, train_y)
#Use the model to make predictions on the test data and return the prediction
probabilities for each category in Species.#
xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Species']], proba = True).sort_values(by =
['Sepal_Length', 'Sepal_Width'])
     Sepal_Length  Sepal_Width     Species       TOP_1  TOP_1_VAL
 0            4.4          3.0      setosa      setosa   0.993619
 1            4.4          3.2      setosa      setosa   0.993619
 2            4.5          2.3      setosa      setosa   0.942128
 3            4.8          3.4      setosa      setosa   0.993619
...           ...          ...         ...         ...        ...
 42           6.7          3.3   virginica   virginica   0.996170
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
 43           6.9          3.1  versicolor  versicolor   0.925217
 44           6.9          3.1   virginica   virginica   0.996170
 45           7.0          3.2  versicolor  versicolor   0.990586

#Create training data and test data.#

dat = oml.sync(table = "IRIS").split()

train_x = dat[0].drop('Sepal_Length')
train_y = dat[0]['Sepal_Length']
test_dat = dat[1]

#Create an XGBoost model object.#

setting = {'xgboost_booster': 'gblinear'}
xgb_mod = oml.xgb('regression', **setting)

#Fit the XGBoost Model according to the training data and parameter settings.#

xgb_mod.fit(train_x, train_y)
xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']]) # doctest:
+NORMALIZE_WHITESPACE, +ELLIPSIS
#Create an XGBoost model object.#

setting = {'xgboost_objective': 'rank:pairwise',
...             'xgboost_max_depth': '3',
...             'xgboost_eta': '0.1',
...             'xgboost_gamma': '1.0',
...             'xgboost_num_round': '4'}

xgb_mod = oml.xgb('regression', **setting)

#Fit the XGBoost Model according to the training data and parameter settings.#

xgb_mod.fit(train_x, train_y)
#Use the model to make predictions on the test data, returning the
Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.#

xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])
```

Listing for This Example

```
#Load the iris data from sklearn and combine the target and predictors into a
single DataFrame, which matches the form of a database table.
Use the oml.create function to load this Pandas DataFrame into the databae,
which creates a persistent table and returns a proxy object that you assign
to z.#

>>> import oml
>>> from sklearn import datasets
>>> import pandas as pd
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data, columns = ['Sepal_Length', 'Sepal_Width',
'Petal_Length', 'Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x: {0: 'setosa', 1: 'versicolor',
2:'virginica'}[x], iris.target)), columns = ['Species'])

>>> #For on-premises database follow the below command to connect to the
database#
>>> oml.connect("<username>","<password>", dsn="<dsn>")
>>> z = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

#Create training data and test data.#

>>> dat = oml.sync(table = "IRIS").split()
>>> train_x = dat[0].drop('Species')
>>> train_y = dat[0]['Species']
>>> test_dat = dat[1]

#Classification Example:#

#Create an XGBoost model object.#

>>> setting = {'xgboost_max_depth': '3',
...            'xgboost_eta': '1',
...            'xgboost_num_round': '10'}

>>> xgb_mod = oml.xgb('classification', **setting)

#Fit the XGBoost model to the training data.#

>>> xgb_mod.fit(train_x, train_y)

Algorithm Name: XGBOOST
Mining Function: CLASSIFICATION
Target: Species

Settings:
                    setting name            setting value
0                      ALGO_NAME             ALGO_XGBOOST
1          CLAS_WEIGHTS_BALANCED                      OFF
2                   ODMS_DETAILS             ODMS_ENABLE
3   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
4                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
5                      PREP_AUTO                      ON
6                        booster                  gbtree
7                            eta                       1
8                      max_depth                       3
9                    ntree_limit                       0
10                   num_round                      10
11                   objective          multi:softprob

Global Statistics:
  attribute name attribute value
0       NUM_ROWS             104
1       mlogloss        0.024858
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Attributes:
Petal_Length
Petal_Width
Sepal_Length
Sepal_Width

Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE      GAIN     COVER
\
0  None   Petal_Length              None            None  0.743941
0.560554
1  None    Petal_Width              None            None  0.162191
0.245400
2  None   Sepal_Length              None            None  0.003738
0.044741
3  None    Sepal_Width              None            None  0.090129
0.149306

   FREQUENCY
0   0.447761
1   0.268657
2   0.119403
3   0.164179

#Use the model to make predictions on the test data and return the prediction
probabilities for each category in Species.#

>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Species']], proba = True).sort_values(by =
['Sepal_Length', 'Sepal_Width'])
     Sepal_Length  Sepal_Width     Species       TOP_1  TOP_1_VAL
 0            4.4          3.0      setosa      setosa   0.993619
 1            4.4          3.2      setosa      setosa   0.993619
 2            4.5          2.3      setosa      setosa   0.942128
 3            4.8          3.4      setosa      setosa   0.993619
...           ...          ...         ...         ...        ...
 42           6.7          3.3   virginica   virginica   0.996170
 43           6.9          3.1  versicolor  versicolor   0.925217
 44           6.9          3.1   virginica   virginica   0.996170
 45           7.0          3.2  versicolor  versicolor   0.990586


#Regression Example:#

#Create training data and test data.#

>>> dat = oml.sync(table = "IRIS").split()

>>> train_x = dat[0].drop('Sepal_Length')
>>> train_y = dat[0]['Sepal_Length']
>>> test_dat = dat[1]

#Create an XGBoost model object.#
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> setting = {'xgboost_booster': 'gblinear'}
>>> xgb_mod = oml.xgb('regression', **setting)

#Fit the XGBoost Model according to the training data and parameter settings.#

>>> xgb_mod.fit(train_x, train_y)

Algorithm Name: XGBOOST
Mining Function: REGRESSION
Target: Sepal_Length

Settings:
                   setting name            setting value
0                     ALGO_NAME             ALGO_XGBOOST
1                   ODMS_DETAILS              ODMS_ENABLE
2  ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3                 ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
4                     PREP_AUTO                       ON
5                       booster                 gblinear
6                   ntree_limit                        0
7                     num_round                       10

Computed Settings:
              setting name setting value
0  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name attribute value
0       NUM_ROWS             104
1           rmse        0.364149

Attributes:
Petal_Length
Petal_Width
Sepal_Width
Species

Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE    WEIGHT  CLASS
0  None   Petal_Length              None            None  0.335183      0
1  None    Petal_Width              None            None  0.368738      0
2  None    Sepal_Width              None            None  0.249208      0
3  None        Species              None      versicolor -0.197582      0
4  None        Species              None       virginica -0.170522      0

>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']]) # doctest:
+NORMALIZE_WHITESPACE, +ELLIPSIS
     Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
 0            4.9          3.0           1.4      setosa    4.797075
 1            4.9          3.1           1.5      setosa    4.818641
 2            4.8          3.4           1.6      setosa    4.963796
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
 3           5.8          4.0           1.2      setosa    4.979247
...          ...          ...           ...         ...         ...
 42          6.7          3.3           5.7   virginica    6.990700
 43          6.7          3.0           5.2   virginica    6.674599
 44          6.5          3.0           5.2   virginica    6.563977
 45          5.9          3.0           5.1   virginica    6.456711


#Ranking Example:#

#Create an XGBoost model object.#

>>> setting = {'xgboost_objective': 'rank:pairwise',
...            'xgboost_max_depth': '3',
...            'xgboost_eta': '0.1',
...            'xgboost_gamma': '1.0',
...            'xgboost_num_round': '4'}

>>> xgb_mod = oml.xgb('regression', **setting)

#Fit the XGBoost Model according to the training data and parameter settings.#

>>> xgb_mod.fit(train_x, train_y)
Algorithm Name: XGBOOST
Mining Function: REGRESSION
Target: Sepal_Length

Settings:
                    setting name            setting value
0                      ALGO_NAME             ALGO_XGBOOST
1                   ODMS_DETAILS              ODMS_ENABLE
2   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
3                  ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
4                      PREP_AUTO                       ON
5                        booster                   gbtree
6                            eta                      0.1
7                          gamma                      1.0
8                      max_depth                        3
9                    ntree_limit                        0
10                     num_round                        4
11                     objective           rank:pairwise

Computed Settings:
              setting name setting value
0  ODMS_EXPLOSION_MIN_SUPP             1

Global Statistics:
  attribute name  attribute value
0       NUM_ROWS              104
1            map                1

Attributes:
Petal_Length
Petal_Width
Sepal_Width
Species
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Partition: NO

ATTRIBUTE IMPORTANCE:

  PNAME ATTRIBUTE_NAME ATTRIBUTE_SUBNAME ATTRIBUTE_VALUE      GAIN     COVER
\
0  None   Petal_Length              None            None  0.873855
0.677624
1  None    Petal_Width              None            None  0.083504
0.184802
2  None    Sepal_Width              None            None  0.042641
0.137574

   FREQUENCY
0   0.500000
1   0.285714
2   0.214286

#Use the model to make predictions on the test data, returning the
Sepal_Length, Sepal_Width, Petal_Length, and Species columns in the result.#

>>> xgb_mod.predict(test_dat.drop('Species'), supplemental_cols = test_dat[:,
['Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Species']])
     Sepal_Length  Sepal_Width  Petal_Length     Species  PREDICTION
 0            4.9          3.0           1.4      setosa    0.243485
 1            4.9          3.1           1.5      setosa    0.243485
 2            4.8          3.4           1.6      setosa    0.243485
 3            5.8          4.0           1.2      setosa    0.310980
...           ...          ...           ...         ...         ...
 42           6.7          3.3           5.7   virginica    0.771761
 43           6.7          3.0           5.2   virginica    0.728637
 44           6.5          3.0           5.2   virginica    0.728637
 45           5.9          3.0           5.1   virginica    0.674835
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

11

Automated Machine Learning

Use the automated algorithm selection, feature selection, and hyperparameter tuning of Automated Machine Learning to accelerate the machine learning modeling process.

Automated Machine Learning in OML4Py is described in the following topics.

Topics:

- About Automated Machine Learning
  Automated Machine Learning (AutoML) provides built-in data science expertise about data analytics and modeling that you can employ to build machine learning models.
- Algorithm Selection
  The `oml.automl.AlgorithmSelection` class uses the characteristics of the data set and the task to rank algorithms from the set of supported Oracle Machine Learning algorithms.
- Feature Selection
  The `oml.automl.FeatureSelection` class identifies the most relevant feature subsets for a training data set and an Oracle Machine Learning algorithm.
- Model Tuning
  The `oml.automl.ModelTuning` class tunes the hyperparameters for the specified classification or regression algorithm and training data.
- Model Selection
  The `oml.automl.ModelSelection` class automatically selects an Oracle Machine Learning algorithm according to the selected score metric and then tunes that algorithm.
- AutoML Pipeline
  The Automated Machine Learning Pipeline uses `oml.automl.Pipeline` class to automatically identify the most relevant algorithms, features, and hyperparameters based on a given training dataset.

11.1 About Automated Machine Learning

Automated Machine Learning (AutoML) provides built-in data science expertise about data analytics and modeling that you can employ to build machine learning models.

Any modeling problem for a specified data set and prediction task involves a sequence of data cleansing and preprocessing, algorithm selection, and model tuning tasks. Each of these steps require data science expertise to help guide the process to an efficient final model. Automated Machine Learning (AutoML) automates this process with its built-in data science expertise.

OML4Py has the following AutoML capabilities:

- Automated algorithm selection that selects the appropriate algorithm from the supported machine learning algorithms
- Automated feature selection that reduces the size of the original feature set to speed up model training and tuning, while possibly also increasing model quality
- Automated tuning of model hyperparameters, which selects the model with the highest score metric from among several metrics as selected by the user

Copyright © 2024, 2026, Oracle and/or its affiliates.

AutoML performs those common modeling tasks automatically, with less effort and potentially better results. It also leverages in-database algorithm parallel processing and scalability to minimize runtime and produce high-quality results.

> **Note**
>
> As the `fit` method of the machine learning classes does, the AutoML functions `reduce`, `select`, and `tune` provide a `case_id` parameter that you can use to achieve repeatable data sampling and data shuffling during model building.

The AutoML functionality is also available in a no-code user interface alongside OML Notebooks on Oracle Autonomous AI Database. For more information, see Oracle Machine Learning AutoML User Interface .

Automated Machine Learning Classes and Algorithms

The Automated Machine Learning classes are the following.

Class	Description
`oml.automl.AlgorithmSelection`	Using only the characteristics of the data set and the task, automatically selects the best algorithms from the set of supported Oracle Machine Learning algorithms. Supports classification and regression functions.
`oml.automl.FeatureSelection`	Uses meta-learning to quickly identify the most relevant feature subsets given a training data set and an Oracle Machine Learning algorithm. Supports classification and regression functions.
`oml.automl.ModelTuning`	Uses a highly parallel, asynchronous gradient-based hyperparameter optimization algorithm to tune the algorithm hyperparameters. Supports classification and regression functions.
`oml.automl.ModelSelection`	Selects the best Oracle Machine Learning algorithm and then tunes that algorithm. Supports classification and regression functions.

The Oracle Machine Learning algorithms supported by AutoML are the following:

Table 11-1 Machine Learning Algorithms Supported by AutoML

Algorithm Abbreviation	Algorithm Name
dt	Decision Tree
glm	Generalized Linear Model
glm_ridge	Generalized Linear Model with ridge regression
nb	Naive Bayes
nn	Neural Network
rf	Random Forest
svm_gaussian	Support Vector Machine with Gaussian kernel
svm_linear	Support Vector Machine with linear kernel

Classification and Regression Metrics

The following tables list the scoring metrics supported by AutoML.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 11-2 Binary and Multiclass Classification Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
accuracy	Calculates the rate of correct classification of the target. sklearn.metrics.accuracy_score(*y_true*, *y_pred*, *normalize=True*, *sample_weight=None*) Formula: (tp + tn)/samples
f1_macro	Calculates the f-score or f-measure, which is a weighted average of the precision and recall. The f1_macro takes the unweighted average of per-class scores. sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None) Formula: 2 * (precision * recall) / (precision + recall)
f1_micro	Calculates the f-score or f-measure with micro-averaging in which true positives, false positives, and false negatives are counted globally. sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None) Formula: 2 * (precision * recall) / (precision + recall)
f1_weighted	Calculates the f-score or f-measure with weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes. sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None) Formula: 2 * (precision * recall) / (precision + recall)
precision_macro	Calculates the ability of the classifier to not label a sample incorrectly. The precision_macro takes the unweighted average of per-class scores. sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None) Formula: tp / (tp + fp)
precision_micro	Calculates the ability of the classifier to not label a sample incorrectly. Uses micro-averaging in which true positives, false positives, and false negatives are counted globally. sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None) Formula: tp / (tp + fp)

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 11-2 (Cont.) Binary and Multiclass Classification Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
precision_weighted	Calculates the ability of the classifier to not label a sample incorrectly. Uses weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes. `sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)` Formula: `tp / (tp + fp)`
recall_macro	Calculates the ability of the classifier to correctly label each class. The recall_macro takes the unweighted average of per-class scores. `sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='macro', sample_weight=None)` Formula: `tp / (tp + fn)`
recall_micro	Calculates the ability of the classifier to correctly label each class with micro-averaging in which the true positives, false positives, and false negatives are counted globally. `sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='micro', sample_weight=None)` Formula: `tp / (tp + fn)`
recall_weighted	Calculates the ability of the classifier to correctly label each class with weighted averaging of per-class scores based on support (the fraction of true samples per class). Accounts for imbalanced classes. `sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='weighted', sample_weight=None)` Formula: `tp / (tp + fn)`

See Also: Scikit-learn classification metrics

Table 11-3 Binary Classification Metrics Only

Metric	Description, Scikit-learn Equivalent, and Formula
f1	Calculates the f-score or f-measure, which is a weighted average of the precision and recall. This metric by default requires a positive target to be encoded as 1 to function as expected. `sklearn.metrics.f1_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)` Formula: `2 * (precision * recall) / (precision + recall)`

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 11-3 (Cont.) Binary Classification Metrics Only

Metric	Description, Scikit-learn Equivalent, and Formula
`precision`	Calculates the ability of the classifier to not label a sample positive (1) that is actually negative (0). `sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)` Formula: `tp / (tp + fp)`
`recall`	Calculates the ability of the classifier to label all positive (1) samples correctly. `sklearn.metrics.recall_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)` Formula: `tp / (tp + fn)`
`roc_auc`	Calculates the Area Under the Receiver Operating Characteristic Curve (roc_auc) from prediction scores. `sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)` See also the definition of receiver operation characteristic.

Table 11-4 Regression Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
`r2`	Calculates the coefficient of determination (R squared). `sklearn.metrics.r2_score(y_true, y_pred, sample_weight=None, multioutput='uniform_average')` See also the definition of coefficient of determination.
`neg_mean_absolute_error`	Calculates the mean of the absolute difference of predicted and true targets (MAE). `sklearn.metrics.mean_absolute_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')` Formula: $-\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)$

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 11-4 (Cont.) Regression Metrics

Metric	Description, Scikit-learn Equivalent, and Formula
`neg_mean_squared_error`	Calculates the mean of the squared difference of predicted and true targets. `-1.0 * sklearn.metrics.mean_squared_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')` Formula: $-\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$
`neg_mean_squared_log_error`	Calculates the mean of the difference in the natural log of predicted and true targets. `sklearn.metrics.mean_squared_log_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')` Formula: $-\frac{1}{n}\sum_{i=1}^{n}(\log(Y_i) - \log(\hat{Y}_i))^2$
`neg_median_absolute_error`	Calculates the median of the absolute difference between predicted and true targets. `sklearn.metrics.median_absolute_error(y_true, y_pred)` Formula: $-Med(\{Y_i - \hat{Y}_i, 0 \leq i < n\})$

See Also: Scikit-learn regression metrics

11.2 Algorithm Selection

The `oml.automl.AlgorithmSelection` class uses the characteristics of the data set and the task to rank algorithms from the set of supported Oracle Machine Learning algorithms.

Selecting the best Oracle Machine Learning algorithm for a data set and a prediction task is non-trivial. No single algorithm works best for all modeling problems. The `oml.automl.AlgorithmSelection` class ranks the candidate algorithms according to how likely each is to produce a quality model. This is achieved by using Oracle advanced meta-learning intelligence learned from a repertoire of data sets with the goal of avoiding exhaustive searches, thereby reducing overall compute time and costs.

The `oml.automl.AlgorithmSelection` class supports classification and regression algorithms. To use the class, you specify a data set and the number of algorithms you want to evaluate.

The `select` method of the class returns a sorted list of the top algorithms and their predicted rank (from best to worst).

Copyright © 2024, 2026, Oracle and/or its affiliates.

For information on the parameters and methods of the class, invoke `help(oml.automl.AlgorithmSelection)` or see Oracle Machine Learning for Python API Reference.

Example 11-1 Using the `oml.automl.AlgorithmSelection` Class

This example creates an `oml.automl.AlgorithmSelection` object and then displays the algorithm rankings with their corresponding score metric. You may select the top entry or choose a different model depending on the needs of your particular business problem.

```
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bc = datasets.load_breast_cancer()
bc_data = bc.data.astype(float)
X = pd.DataFrame(bc_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])

# Create the database table BreastCancer.
oml_df = oml.create(pd.concat([X, y], axis=1),
                                table = 'BreastCancer')

# Split the data set into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Create an automated algorithm selection object with f1_macro as
# the score_metric argument.
asel = automl.AlgorithmSelection(mining_function='classification',
                               score_metric='f1_macro', parallel=4)

# Run algorithm selection to get the top k predicted algorithms and
# their ranking without tuning.
algo_ranking = asel.select(X, y, k=3)

# Show the selected and tuned model.
[(m, "{:.2f}".format(s)) for m,s in algo_ranking]

# Drop the database table.
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>>
>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                                table = 'BreastCancer')
>>>
>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> # Create an automated algorithm selection object with f1_macro as
... # the score_metric argument.
... asel = automl.AlgorithmSelection(mining_function='classification',
...                                  score_metric='f1_macro', parallel=4)
>>>
>>> # Run algorithm selection to get the top k predicted algorithms and
... # their ranking without tuning.
... algo_ranking = asel.select(X, y, k=3)
>>>
>>> # Show the selected and tuned model.
>>> [(m, "{:.2f}".format(s)) for m,s in algo_ranking]
[('svm_gaussian', '0.97'), ('glm_ridge', '0.96'), ('nn', '0.96')]
>>>
>>> # Drop the database table.
... oml.drop('BreastCancer')
```

11.3 Feature Selection

The `oml.automl.FeatureSelection` class identifies the most relevant feature subsets for a training data set and an Oracle Machine Learning algorithm.

In a data analytics application, feature selection is a critical data preprocessing step that has a high impact on both runtime and model performance. The `oml.automl.FeatureSelection` class automatically selects the most relevant features for a data set and model. It internally uses several feature-ranking algorithms to identify the best feature subset that reduces model training time without compromising model performance. Oracle advanced meta-learning techniques quickly prune the search space of this feature selection optimization.

The `oml.automl.FeatureSelection` class supports classification and regression algorithms. To use the `oml.automl.FeatureSelection` class, you specify a data set and the Oracle Machine Learning algorithm on which to perform the feature reduction.

For information on the parameters and methods of the class, invoke `help(oml.automl.FeatureSelection)` or see Oracle Machine Learning for Python API Reference.

Example 11-2 Using the `oml.automl.FeatureSelection` Class

This example uses the `oml.automl.FeatureSelection` class. The example builds a model on the full data set and computes predictive accuracy. It performs automated feature selection, filters the columns according to the determined set, and rebuilds the model. It then recomputes predictive accuracy.

```
import oml
from oml import automl
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
import pandas as pd
import numpy as np
from sklearn import datasets

# Load the digits data set into the database.
digits = datasets.load_digits()
X = pd.DataFrame(digits.data,
                 columns = ['pixel{}'.format(i) for i
                            in range(digits.data.shape[1])])
y = pd.DataFrame(digits.target, columns = ['digit'])
oml_df = oml.create(pd.concat([X, y], axis=1), table = 'DIGITS')

# Split the data set into train and test.
train, test = oml_df.split(ratio=(0.8, 0.2),
                           seed = 1234, strata_cols='digit')
X_train, y_train = train.drop('digit'), train['digit']
X_test, y_test = test.drop('digit'), test['digit']

# Default model performance before feature selection.
mod = oml.svm(mining_function='classification').fit(X_train,
                                                    y_train)
"{:.2}".format(mod.score(X_test, y_test))

# Create an automated feature selection object with accuracy
# as the score_metric.
fs = automl.FeatureSelection(mining_function='classification',
                             score_metric='accuracy', parallel=4)

# Get the reduced feature subset on the train data set.
subset = fs.reduce('svm_linear', X_train, y_train)
"{} features reduced to {}".format(len(X_train.columns),
                                   len(subset))

# Use the subset to select the features and create a model on the
# new reduced data set.
X_new  = X_train[:,subset]
X_test_new = X_test[:,subset]
mod = oml.svm(mining_function='classification').fit(X_new, y_train)
"{:.2} with {:.1f}x feature reduction".format(
  mod.score(X_test_new, y_test),
  len(X_train.columns)/len(X_new.columns))

# Drop the DIGITS table.
oml.drop('DIGITS')

# For reproducible results, add a case_id column with unique row
# identifiers.
row_id = pd.DataFrame(np.arange(digits.data.shape[0]),
                                columns = ['CASE_ID'])
oml_df_cid = oml.create(pd.concat([row_id, X, y], axis=1),
                        table = 'DIGITS_CID')

train, test = oml_df_cid.split(ratio=(0.8, 0.2), seed = 1234,
                               hash_cols='CASE_ID',
                               strata_cols='digit')
X_train, y_train = train.drop('digit'), train['digit']
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
X_test, y_test = test.drop('digit'), test['digit']

# Provide the case_id column name to the feature selection
# reduce function.
subset = fs.reduce('svm_linear', X_train,
                   y_train, case_id='CASE_ID')
"{} features reduced to {} with case_id".format(
                                          len(X_train.columns)-1,
                                          len(subset))

# Drop the tables created in the example.
oml.drop('DIGITS')
oml.drop('DIGITS_CID')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets
>>>
>>> # Load the digits data set into the database.
... digits = datasets.load_digits()
>>> X = pd.DataFrame(digits.data,
...                  columns = ['pixel{}'.format(i) for i
...                             in range(digits.data.shape[1])])
>>> y = pd.DataFrame(digits.target, columns = ['digit'])
>>> oml_df = oml.create(pd.concat([X, y], axis=1), table = 'DIGITS')
>>>
>>> # Split the data set into train and test.
... train, test = oml_df.split(ratio=(0.8, 0.2),
...                            seed = 1234, strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']
>>>
>>> # Default model performance before feature selection.
... mod = oml.svm(mining_function='classification').fit(X_train,
...                                                     y_train)
>>> "{:.2}".format(mod.score(X_test, y_test))
'0.92'
>>>
>>> # Create an automated feature selection object with accuracy
... # as the score_metric.
... fs = automl.FeatureSelection(mining_function='classification',
...                              score_metric='accuracy', parallel=4)
>>> # Get the reduced feature subset on the train data set.
... subset = fs.reduce('svm_linear', X_train, y_train)
>>> "{} features reduced to {}".format(len(X_train.columns),
...                                    len(subset))
'64 features reduced to 41'
>>>
>>> # Use the subset to select the features and create a model on the
... # new reduced data set.
... X_new  = X_train[:,subset]
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> X_test_new = X_test[:,subset]
>>> mod = oml.svm(mining_function='classification').fit(X_new, y_train)
>>> "{:.2} with {:.1f}x feature reduction".format(
...   mod.score(X_test_new, y_test),
...   len(X_train.columns)/len(X_new.columns))
'0.92 with 1.6x feature reduction'
>>>
>>> # Drop the DIGITS table.
... oml.drop('DIGITS')
>>>
>>> # For reproducible results, add a case_id column with unique row
... # identifiers.
>>> row_id = pd.DataFrame(np.arange(digits.data.shape[0]),
...                                columns = ['CASE_ID'])
>>> oml_df_cid = oml.create(pd.concat([row_id, X, y], axis=1),
...                         table = 'DIGITS_CID')

>>> train, test = oml_df_cid.split(ratio=(0.8, 0.2), seed = 1234,
...                                hash_cols='CASE_ID',
...                                strata_cols='digit')
>>> X_train, y_train = train.drop('digit'), train['digit']
>>> X_test, y_test = test.drop('digit'), test['digit']
>>>
>>> # Provide the case_id column name to the feature selection
... # reduce function.
>>> subset = fs.reduce('svm_linear', X_train,
...                    y_train, case_id='CASE_ID')
... "{} features reduced to {} with case_id".format(
...                                            len(X_train.columns)-1,
...                                            len(subset))
'64 features reduced to 45 with case_id'
>>>
>>> # Drop the tables created in the example.
... oml.drop('DIGITS')
>>> oml.drop('DIGITS_CID')
```

11.4 Model Tuning

The `oml.automl.ModelTuning` class tunes the hyperparameters for the specified classification or regression algorithm and training data.

Model tuning is a laborious machine learning task that relies heavily on data scientist expertise. With limited user input, the `oml.automl.ModelTuning` class automates this process using a highly-parallel, asynchronous gradient-based hyperparameter optimization algorithm to tune the hyperparameters of an Oracle Machine Learning algorithm.

The `oml.automl.ModelTuning` class supports classification and regression algorithms. To use the `oml.automl.ModelTuning` class, you specify a data set and an algorithm to obtain a tuned model and its corresponding hyperparameters. An advanced user can provide a customized hyperparameter search space and a non-default scoring metric to this black box optimizer.

For a partitioned model, if you pass in the column to partition on in the `param_space` argument of the `tune` method, `oml.automl.ModelTuning` tunes the partitioned model's hyperparameters.

For information on the parameters and methods of the class, invoke `help(oml.automl.ModelTuning)` or see Oracle Machine Learning for Python API Reference.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 11-3 Using the `oml.automl.ModelTuning` Class

This example creates an `oml.automl.ModelTuning` object.

```
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bc = datasets.load_breast_cancer()
bc_data = bc.data.astype(float)
X = pd.DataFrame(bc_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])

# Create the database table BreastCancer.
oml_df = oml.create(pd.concat([X, y], axis=1),
                    table = 'BreastCancer')

# Split the data set into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Start an automated model tuning run with a Decision Tree model.
at = automl.ModelTuning(mining_function='classification',
score_metric='accuracy',
      parallel=4)
results = at.tune('dt', X, y)

# Show the tuned model details.
tuned_model = results['best_model']
tuned_model

# Show the best tuned model train score and the
# corresponding hyperparameters.
score, params = results['all_evals'][0]
"{:.2}".format(score), ["{}:{}".format(k, params[k])
  for k in sorted(params)]

# Use the tuned model to get the score on the test set.
"{:.2}".format(tuned_model.score(X_test, y_test))

# An example invocation of model tuning with user-defined
# search ranges for selected hyperparameters on a new tuning
# metric (f1_macro).
search_space = {
  'RFOR_SAMPLING_RATIO': {'type': 'continuous',
                          'range': [0.01, 0.5]},
  'RFOR_NUM_TREES': {'type': 'discrete',
                     'range': [50, 100]},
  'TREE_IMPURITY_METRIC': {'type': 'categorical',
                           'range': ['TREE_IMPURITY_ENTROPY',
                           'TREE_IMPURITY_GINI']},}
results = at.tune('rf', X, y, param_space=search_space)
score, params = results['all_evals'][0]
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
("{:.2}".format(score), ["{}:{}".format(k, params[k])
  for k in sorted(params)])

# Some hyperparameter search ranges need to be defined based on the
# training data set sizes (for example, the number of samples and
# features). You can use placeholders specific to the data set,
# such as $nr_features and $nr_samples, as the search ranges.
search_space = {'RFOR_MTRY': {'type': 'discrete',
                              'range': [1, '$nr_features/2']}}
results = at.tune('rf', X, y, param_space=search_space)
score, params = results['all_evals'][0]
("{:.2}".format(score), ["{}:{}".format(k, params[k])
  for k in sorted(params)])

# Drop the database table.
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>>
>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                     table = 'BreastCancer')
>>>
>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> # Start an automated model tuning run with a Decision Tree model.
... at = automl.ModelTuning(mining_function='classification',
score_metric='accuracy',
...     parallel=4)
>>> results = at.tune('dt', X, y)
>>>
>>> # Show the tuned model details.
... tuned_model = results['best_model']
>>> tuned_model

Algorithm Name: Decision Tree

Mining Function: CLASSIFICATION

Target: TARGET
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
Settings:
                     setting name            setting value
0                       ALGO_NAME       ALGO_DECISION_TREE
1               CLAS_MAX_SUP_BINS                       32
2           CLAS_WEIGHTS_BALANCED                      OFF
3                     ODMS_DETAILS             ODMS_DISABLE
4   ODMS_MISSING_VALUE_TREATMENT  ODMS_MISSING_VALUE_AUTO
5                   ODMS_SAMPLING    ODMS_SAMPLING_DISABLE
6                       PREP_AUTO                       ON
7            TREE_IMPURITY_METRIC       TREE_IMPURITY_GINI
8             TREE_TERM_MAX_DEPTH                        8
9           TREE_TERM_MINPCT_NODE                     3.34
10        TREE_TERM_MINPCT_SPLIT                      0.1
11         TREE_TERM_MINREC_NODE                       10
12        TREE_TERM_MINREC_SPLIT                       20

Attributes:
mean radius
mean texture
mean perimeter
mean area
mean smoothness
mean compactness
mean concavity
mean concave points
mean symmetry
mean fractal dimension
radius error
texture error
perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension

Partition: NO

>>>
>>> # Show the best tuned model train score and the
... # corresponding hyperparameters.
... score, params = results['all_evals'][0]
>>> "{:.2}".format(score), ["{}:{}".format(k, params[k])
...   for k in sorted(params)]
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
('0.92', ['CLAS_MAX_SUP_BINS:32', 'TREE_IMPURITY_METRIC:TREE_IMPURITY_GINI',
'TREE_TERM_MAX_DEPTH:7', 'TREE_TERM_MINPCT_NODE:0.05',
'TREE_TERM_MINPCT_SPLIT:0.1'])
>>>
>>> # Use the tuned model to get the score on the test set.
... "{:.2}".format(tuned_model.score(X_test, y_test))
'0.92
>>>
>>> # An example invocation of model tuning with user-defined
... # search ranges for selected hyperparameters on a new tuning
... # metric (f1_macro).
...  search_space = {
...   'RFOR_SAMPLING_RATIO': {'type': 'continuous',
...                           'range': [0.01, 0.5]},
...   'RFOR_NUM_TREES': {'type': 'discrete',
...                      'range': [50, 100]},
...   'TREE_IMPURITY_METRIC': {'type': 'categorical',
...                            'range': ['TREE_IMPURITY_ENTROPY',
...                            'TREE_IMPURITY_GINI']},}
>>> results = at.tune('rf', X, y, param_space=search_space)
>>> score, params = results['all_evals'][0]
>>> ("{:.2}".format(score), ["{}:{}".format(k, params[k])
...   for k in sorted(params)])
('0.92', ['RFOR_NUM_TREES:53', 'RFOR_SAMPLING_RATIO:0.4999951',
'TREE_IMPURITY_METRIC:TREE_IMPURITY_ENTROPY'])
>>>
>>> # Some hyperparameter search ranges need to be defined based on the
... # training data set sizes (for example, the number of samples and
... # features). You can use placeholders specific to the data set,
... # such as $nr_features and $nr_samples, as the search ranges.
... search_space = {'RFOR_MTRY': {'type': 'discrete',
...                               'range': [1, '$nr_features/2']}}
>>> results = at.tune('rf', X, y, param_space=search_space)
>>> score, params = results['all_evals'][0]
>>> ("{:.2}".format(score), ["{}:{}".format(k, params[k])
...   for k in sorted(params)])
('0.93', ['RFOR_MTRY:10'])
>>>
>>> # Drop the database table.
... oml.drop('BreastCancer')
```

11.5 Model Selection

The `oml.automl.ModelSelection` class automatically selects an Oracle Machine Learning algorithm according to the selected score metric and then tunes that algorithm.

The `oml.automl.ModelSelection` class supports classification and regression algorithms. To use the `oml.automl.ModelSelection` class, you specify a data set and the number of algorithms you want to tune.

The `select` method of the class returns the best model out of the models considered.

For information on the parameters and methods of the class, invoke `help(oml.automl.ModelSelection)` or see Oracle Machine Learning for Python API Reference.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 11-4 Using the `oml.automl.ModelSelection` Class

This example creates an `oml.automl.ModelSelection` object and then uses the object to select and tune the best model.

```
import oml
from oml import automl
import pandas as pd
from sklearn import datasets

# Load the breast cancer data set.
bc = datasets.load_breast_cancer()
bc_data = bc.data.astype(float)
X = pd.DataFrame(bc_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])

# Create the database table BreastCancer.
oml_df = oml.create(pd.concat([X, y], axis=1),
                    table = 'BreastCancer')

# Split the data set into training and test data.
train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Create an automated model selection object with f1_macro as the
# score_metric argument.
ms = automl.ModelSelection(mining_function='classification',
                           score_metric='f1_macro', parallel=4)

# Run model selection to get the top (k=1) predicted algorithm
# (defaults to the tuned model).
select_model = ms.select(X, y, k=1)

# Show the selected and tuned model.
select_model

# Score on the selected and tuned model.
"{:.2}".format(select_model[0].score(X_test, y_test))

# Drop the database table.
oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> from sklearn import datasets
>>>
>>> # Load the breast cancer data set.
... bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>> # Create the database table BreastCancer.
>>> oml_df = oml.create(pd.concat([X, y], axis=1),
...                     table = 'BreastCancer')
>>>
>>> # Split the data set into training and test data.
... train, test = oml_df.split(ratio=(0.8, 0.2), seed = 1234)
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
>>>
>>> # Create an automated model selection object with f1_macro as the
... # score_metric argument.
... ms = automl.ModelSelection(mining_function='classification',
...                            score_metric='f1_macro', parallel=4)
>>>
>>> # Run the model selection to get the top (k=1) predicted algorithm
... # (defaults to the tuned model).
... select_model = ms.select(X, y, k=1)
>>>
>>> # Show the selected and tuned model.
... select_model

Algorithm Name: Support Vector Machine

Mining Function: CLASSIFICATION

Target: TARGET

Settings:
                    setting name                  setting value
0                      ALGO_NAME  ALGO_SUPPORT_VECTOR_MACHINES
1          CLAS_WEIGHTS_BALANCED                            OFF
2                   ODMS_DETAILS                   ODMS_DISABLE
3   ODMS_MISSING_VALUE_TREATMENT       ODMS_MISSING_VALUE_AUTO
4                  ODMS_SAMPLING         ODMS_SAMPLING_DISABLE
5                      PREP_AUTO                            ON
6         SVMS_COMPLEXITY_FACTOR                            10
7            SVMS_CONV_TOLERANCE                         .0001
8           SVMS_KERNEL_FUNCTION                 SVMS_GAUSSIAN
9                SVMS_NUM_PIVOTS                           ...
10                  SVMS_STD_DEV            5.3999999999999995

Attributes:
area error
compactness error
concave points error
concavity error
fractal dimension error
mean area
mean compactness
mean concave points
mean concavity
mean fractal dimension
mean perimeter
mean radius
mean smoothness
mean symmetry
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
mean texture
perimeter error
radius error
smoothness error
symmetry error
texture error
worst area
worst compactness
worst concave points
worst concavity
worst fractal dimension
worst perimeter
worst radius
worst smoothness
worst symmetry
worst texture
Partition: NO

>>>
>>> # Score on the selected and tuned model.
... "{:.2}".format(select_model[0].score(X_test, y_test))
'0.99'
>>>
>>> # Drop the database table.
... oml.drop('BreastCancer')
```

11.6 AutoML Pipeline

The Automated Machine Learning Pipeline uses `oml.automl.Pipeline` class to automatically identify the most relevant algorithms, features, and hyperparameters based on a given training dataset.

The `oml.automl.Pipeline` class automates three major stages of the machine learning pipeline: *algorithm selection*, *adaptive data reduction* (feature and sample size selection), and *hyperparameter optimization*. These stages are combined into an AutoML pipeline which automatically optimizes the whole pipeline with limited user input/interaction.

The `oml.automl.Pipeline` class supports classification and regression algorithms. To use the `oml.automl.Pipeline` class, you specify the mining function, score metric and the degree of parallelism for the AutoML module.

For information on the parameters and methods of the class, invoke `help(oml.automl.Pipeline)` or see Oracle Machine Learning for Python API Reference.

Example 11-5 Using the `oml.automl.Pipeline` Class

This example uses `automl.Pipeline` object to create an automated machine learning pipeline object and then uses the object to fit, predict and inspect the best tuned and fitted model produced by the AutoML pipeline.

```
import oml
from oml import automl
import pandas as pd
import numpy as np
from sklearn import datasets
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Load the breast cancer dataset into the database
bc = datasets.load_breast_cancer()
bc_data = bc.data.astype(float)
X = pd.DataFrame(bc_data, columns = bc.feature_names)
y = pd.DataFrame(bc.target, columns = ['TARGET'])
row_id = pd.DataFrame(np.arange(bc_data.shape[0]), columns = ['CASE_ID'])
df = oml.create(pd.concat([row_id, X, y], axis=1), table = 'BreastCancer')

# Split dataset into train and test
train, test = df.split(ratio=(0.8, 0.2), seed = 1234, hash_cols=['CASE_ID'])
X, y = train.drop('TARGET'), train['TARGET']
X_test, y_test = test.drop('TARGET'), test['TARGET']

# Create an automated machine learning pipeline object with f1_macro
score_metric
pipeline = automl.Pipeline(mining_function='classification',
score_metric='f1_macro', parallel=4)

# Fit the pipeline to perform automated algorithm selection, feature
selection, and model tuning on the dataset
pipeline = pipeline.fit(X, y, case_id='CASE_ID')

# Use the pipeline for prediction
pipeline.predict(X_test, supplemental_cols=y_test)

# For classification tasks, the pipeline can also predict class probabilities
pipeline.predict_proba(X_test, supplemental_cols=y_test)

# Inspect the best tuned and fitted model produced by the AutoML pipeline
pipeline.top_k_tuned_models[0]['fitted_model']

oml.drop('BreastCancer')
```

Listing for This Example

```
>>> import oml
>>> from oml import automl
>>> import pandas as pd
>>> import numpy as np
>>> from sklearn import datasets

# Load the breast cancer dataset into the database
>>> bc = datasets.load_breast_cancer()
>>> bc_data = bc.data.astype(float)
>>> X = pd.DataFrame(bc_data, columns = bc.feature_names)
>>> y = pd.DataFrame(bc.target, columns = ['TARGET'])
>>> row_id = pd.DataFrame(np.arange(bc_data.shape[0]), columns = ['CASE_ID'])
>>> df = oml.create(pd.concat([row_id, X, y], axis=1), table = 'BreastCancer')

# Split dataset into train and test
>>> train, test = df.split(ratio=(0.8, 0.2), seed = 1234,
hash_cols=['CASE_ID'])
>>> X, y = train.drop('TARGET'), train['TARGET']
>>> X_test, y_test = test.drop('TARGET'), test['TARGET']
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Create an automated machine learning pipeline object with f1_macro
score_metric
>>> pipeline = automl.Pipeline(mining_function='classification',
score_metric='f1_macro', parallel=4)

# Fit the pipeline to perform automated algorithm selection, feature
selection, and model tuning on the dataset
>>> pipeline = pipeline.fit(X, y, case_id='CASE_ID')

# Use the pipeline for prediction
>>> pipeline.predict(X_test, supplemental_cols=y_test)
TARGET PREDICTION
0 0 0
1 0 0
2 0 0
3 0 0
4 1 1
.. ... ...
109 1 1
110 1 1
111 0 0
112 1 1
113 0 0
[114 rows x 2 columns]

# For classification tasks, the pipeline can also predict class probabilities
>>> pipeline.predict_proba(X_test, supplemental_cols=y_test)
TARGET PROBABILITY_OF_0 PROBABILITY_OF_1
0 1 0.295266 0.704734
1 1 0.000947 0.999053
2 0 0.999089 0.000911
3 1 0.576097 0.423903
4 1 0.000350 0.999650
.. ... ... ...
109 0 0.991110 0.008890
110 0 0.999776 0.000224
111 0 0.966321 0.033679
112 1 0.000134 0.999866
113 1 0.000744 0.999256
[114 rows x 3 columns]

# Inspect the best tuned and fitted model produced by the AutoML pipeline
>>> pipeline.top_k_tuned_models[0]['fitted_model']
Algorithm Name: Support Vector Machine
Mining Function: CLASSIFICATION
Target: TARGET
Settings:
setting name setting value
0 ALGO_NAME ALGO_SUPPORT_VECTOR_MACHINES
1 CLAS_WEIGHTS_BALANCED OFF
2 ODMS_DETAILS ODMS_ENABLE
3 ODMS_MISSING_VALUE_TREATMENT ODMS_MISSING_VALUE_AUTO
4 ODMS_SAMPLING ODMS_SAMPLING_DISABLE
5 PREP_AUTO ON
6 SVMS_COMPLEXITY_FACTOR 10
7 SVMS_CONV_TOLERANCE .0001
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
8 SVMS_KERNEL_FUNCTION SVMS_GAUSSIAN
9 SVMS_NUM_PIVOTS 200
10 SVMS_STD_DEV 3.6742346141747673
Computed Settings:
setting name setting value
0 SVMS_NUM_ITERATIONS 30
1 SVMS_SOLVER SVMS_SOLVER_IPM
Global Statistics:
attribute name attribute value
0 CONVERGED YES
1 ITERATIONS 13
2 NUM_ROWS 455
Attributes:
area error
compactness error
concave points error
concavity error
fractal dimension error
mean area
mean compactness
mean concave points
mean concavity
mean fractal dimension
mean perimeter
mean radius
mean smoothness
mean symmetry
mean texture
perimeter error
radius error
worst area
worst compactness
worst concave points
worst concavity
worst fractal dimension
worst perimeter
worst radius
worst smoothness
worst symmetry
worst texture
Partition: NO

>>> oml.drop('BreastCancer')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

12

Import Pretrained Models in ONNX Format

Oracle AI Database 26ai includes an ONNX runtime engine for running embedding models directly inside the database. This section covers the process of importing existing pretrained embedding models into Oracle AI Database, including converting those models into the ONNX format if they are not already converted.

Open Neural Network Exchange or ONNX is an open standard format of machine learning models. Consider the following use cases:

- Using complex media input such as text or image for similarity search
- Perform text classification
- Perform reranking

You need vector embedding of the media input to perform all the tasks mentioned above. ONNX pipeline models allows you to convert text and/or image models into ONNX format if they are not already in ONNX format, import the ONNX format models into Oracle AI Database, and generate embeddings from your data within the database. The ONNX format models imported to the database could be used for tasks such as search and classification.

> **Note**
>
> You can use the ONNX pipeline utilities directly within the OML4Py Client Container. See Install OML4Py Client Container for more information. The container includes OML4Py client 2.1.1 and the `oml.utils` utility package (including `ONNXPipeline`, `ONNXPipelineConfig`, and `MiningFunction`), so you can list preconfigured models and export a selected model to a file and/or into the database from a Python session in the container.

Pretrained models are models that are already trained on a media data (text, image, etc.) and saved to a storage format for future use. Hugging Face is the most popular platform that hosts pretrained models typically created with PyTorch.

This table lists machine learning models, along with a brief description, model size, and a PAR download link for each.

Model Name	Description	Dimensions	Size (MB)	Last Updated	Download
all_MiniLM_L12_v2	This is a SentenceTransformers model suitable for semantic similarity search and clustering use cases.	384	116.92	7/15/2024	Link

Copyright © 2024, 2026, Oracle and/or its affiliates.

Model Name	Description	Dimensions	Size (MB)	Last Updated	Download
multilingual_e5_base	This is a multilingual sentence-transformer model that supports 100 languages. Supports text embedding, translation, and multilingual understanding. Features 12 layers with 768-dimensional embeddings for multilingual tasks.	768	1040	2/17/2024	Link
multilingual_e5_small	This is a multilingual sentence-transformer model that supports 100. Supports text embedding, translation, or multilingual understanding. Optimized for efficiency, it can handle multiple languages while maintaining a smaller size for faster inference and reduced computational requirements.	384	76.8	9/9/2024	Link
clip_vit_base_patch32_txt	This is a text encoder that produces embeddings compatible with the CLIP image encoder. Enables text-to-image search and similarity matching between text descriptions and images.	512	243.57	1/29/2021	Link

Copyright © 2024, 2026, Oracle and/or its affiliates.

Model Name	Description	Dimensions	Size (MB)	Last Updated	Download
clip_vit_base_patch32_img	This is an image encoder that produces embeddings compatible with the CLIP text encoder. Enables image-to-text search and similarity matching between images and text descriptions.	512	335.38	1/29/2021	Link

The Python package `oml.utils` contains three classes: `ONNXPipeline`, `ONNXPipelineConfig`, and `MiningFunction`. The package handles ONNX pipeline generation and export, while also incorporating the necessary pre- and post-processing steps.

- **ONNXPipeline** : Allows you to import a pretrained model (Your own pretrained model or one of the supported models from Hugging Face)
- **ONNXPipelineConfig** : Allows you to configure the attributes of pretrained model (Your own pretrained model or one of the supported models from Hugging Face)
- **MiningFunction**: Allows you to choose from one of the following mining options:
  - `EMBEDDING` : Corresponds to text and image embedding
  - `CLASSIFICATION` : Corresponds to text classification
  - `REGRESSION` : Corresponds to re-ranking

> **Warning**
>
> EmbeddingModel and EmbeddingModelConfig are deprecated. Instead, please use ONNXPipeline and ONNXPipelineConfig respectively. The details of the deprecated classes can be found in Python Classes to Convert Pretrained Models to ONNX Models (Deprecated). If a you choose to use a deprecated class, a warning message will be shown indicating that the classes will be removed in the future and advising the user to switch to the new class.

The ONNX pipeline models are available for text embedding, image embedding, multi-modal embedding, text classification and re-ranking.

- Convert Pretrained Models to ONNX Model: End-to-End Instructions for Text Embedding
  This section provides end-to-end instructions from installing the OML4Py client to downloading a pretrained embedding model in ONNX-format using the Python utility package offered by Oracle.
- Python Classes to Convert Pretrained Models to ONNX Models
- ONNX Pipeline Models: Text Embedding
  ONNX pipeline models provides text embedding models that accepts text as input and produces embeddings as output. The pipeline models also provide the necessary pre-processing and post-processing needed for the text.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- ONNX Pipeline Models: Image Embedding
  ONNX pipeline models provides image embedding models that accepts image as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing.
- ONNX Pipeline Models: CLIP Multi-Modal Embedding
  ONNX pipeline models provides multi-modal embedding models that accepts both image and text as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing needed.
- ONNX Pipeline Models: Text Classification
  ONNX pipeline models provides text classification models that accepts text strings as input and produces the probablity of the input string belonging to a specific label. The pipeline models also provide the necessary pre-processing and post-processing.
- ONNX Pipeline Models: Reranking Pipeline
  ONNX pipeline models provide a reranking pipeline that calculates similarity score for a given pair of texts.
- Load Custom Models from The Local Filesystem
  Custom models that have been fine-tuned from pre-trained Hugging Face models and reside on the local filesystem can be converted to the ONNX format using the new `local_model_path` setting. This setting can also be useful in cases where networking is unavailable and a pre-trained model has previously been downloaded.
- Support For Large ONNX Format Model
  OML4Py supports large ONNX format models outside of the database. A large model is divided into a single ONNX file and one or more external data files. The maximum size limit for an ONNX file is 2GB, so for models larger than 2GB, the ONNX file must be split into smaller parts.
- Model Summary
  A model summary provides a concise overview of the model. To obtain the summary after exporting the model, call the `summary()` function.

12.1 Convert Pretrained Models to ONNX Model: End-to-End Instructions for Text Embedding

This section provides end-to-end instructions from installing the OML4Py client to downloading a pretrained embedding model in ONNX-format using the Python utility package offered by Oracle.

> **Note**
>
> This example provides end to end instructions for converting pretrained text models to ONNX models. Steps 1 - 9 are identical for image models and multi-modals. You can use appropriate code/syntax mentioned in the corresponding topics to convert image models and multi models to ONNX pipeline models.

These instructions assume you have configured your Oracle Linux 8 repo in `/etc/yum.repos.d`, configured a Wallet if using an Autonomous AI Database, and set up a proxy if needed.

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. Install Python:

```
sudo yum install libffi-devel openssl openssl-devel tk-devel xz-devel \
zlib-devel bzip2-devel readline-devel libuuid-devel \
ncurses-devel libaio tcl-devel openblas


wget https://www.python.org/ftp/python/3.13.5/Python-3.13.5.tgz
mkdir -p $HOME/python
tar -xvzf Python-3.13.5.tgz --strip-components=1 -C $HOME/python
cd $HOME/python
./configure --enable-shared --prefix=$HOME/python
make clean; make
make altinstall
```

2. Set variables `PYTHONHOME`, `PATH`, and `LD_LIBRARY_PATH`:

```
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

3. Create symlink for python3 and pip3:

```
cd $HOME/python/bin
ln -s python3.13 python3
ln -s pip3.13 pip3
```

4. Install Oracle Instant Client if you plan to load embedded models from Python directly into a database. If you are exporting models to a file, you can skip this step—see the note about environment variables in Step 5 for further details.

```
sudo yum install -y oracle-instantclient-release-el8 oracle-instantclient-
basic
```

5. Create an environment file, for example, `env.sh`, that defines the Python and Oracle Instant client environment variables and source these environment variables before each OML4Py client session. Alternatively, add the environment variable definitions to `.bashrc` so they are defined when the user logs into their Linux machine.

```
# Environment variables for Python
export PYTHONHOME=$HOME/python
export PATH=$PYTHONHOME/bin:$PATH
export LD_LIBRARY_PATH=$PYTHONHOME/lib:$LD_LIBRARY_PATH
```

> **Note**
>
> Environment variable for Oracle Instant Client - only if the Oracle Instant Client is installed for exporting models to the database.
> `export LD_LIBRARY_PATH=$HOME/instantclient_23_26:$LD_LIBRARY_PATH`
>
> .

Copyright © 2024, 2026, Oracle and/or its affiliates.

6. Create a file named requirements.txt that contains the required third-party packages listed below.

```
--extra-index-url https://download.pytorch.org/whl/cpu
pandas==2.2.3
setuptools==80.8.0
scipy==1.14.1
matplotlib==3.10.0
oracledb==3.3.0
scikit-learn==1.6.1
numpy==2.1.0
pyarrow==19.0.0
onnxruntime==1.20.0
onnxruntime-extensions==0.14.0
onnx==1.18.0
torch==2.9.0
transformers==4.56.1
sentencepiece==0.2.1
```

> **Note**
>
> Alternatively, you can install the required supporting packages by downloading and using the supporting packages bundle: oml4py-supporting-linux-x86_64-2.1.1.zip. See Install the Required Supporting Packages for Linux for On-Premises Databases for more information.

7. Upgrade pip3 and install the packages listed in requirements.txt.

```
pip3 install --upgrade pip
pip3 install -r requirements.txt
```

8. To install the OML4Py client, download the OML4Py 2.1.1 client from the OML4Py download page. Accept the Oracle License Agreement, then click on oml4py-client-linux-x86_64-2.1.1.zip to download the ZIP file. After downloading, upload the ZIP file to your Linux machine.

```
unzip oml4py-client-linux-x86_64-2.1.1.zip
pip3 install client/oml-2.1.1-cp313-cp313-linux_x86_64.whl
```

> **Note**
>
> Install the OML4Py Slim Client Modules if you only need access to the ONNX model conversion feature. This lightweight client is ideal for scenarios where you require ONNX model conversion without the complete set of OML4Py functionalities. See Install OML4Py Slim Client for more information.

Copyright © 2024, 2026, Oracle and/or its affiliates.

9. To get a list of all preconfigured models, start Python and import ONNXPipelineConfig from `oml.utils`.

```
python3
from oml.utils import ONNXPipelineConfig
ONNXPipelineConfig.show_preconfigured()
```

```
['sentence-transformers/all-mpnet-base-v2',
'sentence-transformers/all-MiniLM-L6-v2',
'sentence-transformers/multi-qa-MiniLM-L6-cos-v1',
'sentence-transformers/distiluse-base-multilingual-cased-v2',
'sentence-transformers/all-MiniLM-L12-v2',
'BAAI/bge-small-en-v1.5',
'BAAI/bge-base-en-v1.5',
'taylorAI/bge-micro-v2',
'intfloat/e5-small-v2',
'intfloat/e5-base-v2',
'thenlper/gte-base',
'thenlper/gte-small',
'TaylorAI/gte-tiny',
'sentence-transformers/paraphrase-multilingual-mpnet-base-v2',
'intfloat/multilingual-e5-base',
'intfloat/multilingual-e5-small',
'sentence-transformers/stsb-xlm-r-multilingual',
'Snowflake/snowflake-arctic-embed-xs',
'Snowflake/snowflake-arctic-embed-s',
'Snowflake/snowflake-arctic-embed-m',
'mixedbread-ai/mxbai-embed-large-v1',
'openai/clip-vit-large-patch14',
'google/vit-base-patch16-224',
'microsoft/resnet-18',
'microsoft/resnet-50',
'WinKawaks/vit-tiny-patch16-224',
'Falconsai/nsfw_image_detection',
'WinKawaks/vit-small-patch16-224',
'nateraw/vit-age-classifier',
'rizvandwiki/gender-classification',
'AdamCodd/vit-base-nsfw-detector',
'trpakov/vit-face-expression',
'BAAI/bge-reranker-base',
'BAAI/bge-large-en-v1.5',
'ibm-granite/granite-embedding-278m-multilingual',
'thenlper/gte-large',
'Snowflake/snowflake-arctic-embed-l',
'WhereIsAI/UAE-Large-V1']
```

10. Choose from:

    - To generate an ONNX file that you can manually upload to the database using the `DBMS_VECTOR.LOAD_ONNX_MODEL`, refer to step 3 of SQL Quick Start and skip step 10.
    - To upload the model directly into the database using the full version of the OML4Py client, skip this step and proceed to step 11.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Export a preconfigured embedding model to a local file. Import ONNXPipeline and ONNXPipelineConfig from `oml.utils`. This exports the ONNX-format model to your local file system.

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig

# Export to file
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
pipeline.export2file("your_preconfig_file_name",output_dir=".")
```

Move the ONNX file to a directory on the database server, and create a directory on the file system and in the database for the import.

```
mkdir -p /tmp/models
sqlplus / as sysdba
alter session set container=<name of pluggable database>;
```

Apply the necessary permissions and grants.

```
-- directory to store ONNX files for import
CREATE DIRECTORY ONNX_IMPORT AS '/tmp/models';
-- grant your OML user read and write permissions on the directory
GRANT READ, WRITE ON DIRECTORY ONNX_IMPORT to OMLUSER;
-- grant to allow user to import the model
GRANT CREATE MINING MODEL TO OMLUSER;
```

Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure to load the model in your OML user schema. In this example, the procedure loads the ONNX model file named `all-MiniLM-L6.onnx` from the ONNX_IMPORT directory into the database as a model named ALL_MINILM_L6.

```
BEGIN
   DBMS_VECTOR.LOAD_ONNX_MODEL(
     directory => 'ONNX_IMPORT',
     file_name => 'all-MiniLM-L6-v2.onnx',
     model_name => 'ALL_MINILM_L6',
     metadata => JSON('{"function" : "embedding", "embeddingOutput" :
"embedding", "input": {"input": ["DATA"]}}'));
END;
```

11. Export a preconfigured embedding model to the database. If you are using a database connection, make sure to update the connection details to match your credentials and database environment.

> **Note**
>
> To ensure step 11 works properly, complete steps 4 first.

```
# Import oml library and EmbeddingModel from oml.utils
import oml
from oml.utils import ONNXPipeline, ONNXPipelineConfig
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Set embedded mode to false for Oracle AI Database on premises. This is
not supported or required for Oracle Autonomous AI Database.
oml.core.methods.__embed__ = False

# Create a database connection.

# Oracle AI Database on-premises
oml.connect("<user>", "<password>", port=<port number> host="<hostname>",
service_name="<service name>")

# Oracle Autonomous AI Database
oml.connect(user="<user>", password="<password>", dsn="myadb_low")
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
pipeline.export2db("ALL_MINILM_L6")
```

Query the model and its views, and you can generate embeddings from Python or SQL.

```
import oracledb
cr = oml.cursor()
data = cr.execute("select vector_embedding(ALL_MINILM_L6 using 'RES' as
DATA)AS embedding from dual")
data.fetchall()


[(array('f', [-0.11642304062843323, 0.015433191321790218,
-0.04692629724740982, 0.007167273201048374,
0.035023435950279236, -0.04029884934425354, 0.018412720412015915,
0.0648672804236412, 0.014992560259997845,
0.012053844518959522, -0.03542192652821541, -0.09510122984647751,
-0.020823240280151367, 0.049922555685043335,
-0.01863112859427929, -0.026279663667082787, -0.03260187804698944,
0.05227319151163101, -0.09848847985267639,
0.0009467907948419452, 0.04395370930433273, 0.01060416642576456,
0.029333725571632385, -0.04252052679657936,
-0.025767622515559196, -0.03291202709078789, -0.053170595318078995,
0.07415973395109177, 0.0082617262378335,
...
...


SELECT VECTOR_EMBEDDING(ALL_MINILM_L6 USING 'RES' as DATA) AS embedding;


EMBEDDING
--------------------------------------------------------------------------
-----
[-1.16423041E-001,1.54331913E-002,-4.69262972E-002,7.1672732E-003,3.5023436
E-002
,-4.02988493E-002,1.84127204E-002,6.48672804E-002,1.49925603E-002,1.2053844
5E-00
2,-3.54219265E-002,-9.51012298E-002,-2.08232403E-002,4.99225557E-002,-1.863
11286
E-002,-2.62796637E-002,-3.2601878E-002,5.22731915E-002,-9.84884799E-002,9.4
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
67907
95E-004,4.39537093E-002,1.06041664E-002,2.93337256E-002,-4.25205268E-002,-2
.5767
...
...
```

> **Note**
>
> To suppress Torch warnings, use the following code:
>
> ```
> import warnings
> import torch warnings.filterwarnings("ignore",
> category=DeprecationWarning)
> warnings.filterwarnings("ignore", category=torch.jit.TracerWarning)
> ```

12. Verify the model exists using SQL:

```
sqlplus $USER/pass@PDBNAME;


select model_name, algorithm, mining_function from user_mining_models
where  model_name='ALL_MINILM_L6';


------------------------------------------------------------------------------
MODEL_NAME                ALGORITHM                      MINING_FUNCTION
------------------------- ------------------------------ ------------------------------
ALL_MINILM_L6             ONNX                           EMBEDDING
```

12.2 Python Classes to Convert Pretrained Models to ONNX Models

This topic provides the functions, attributes and example usage of the python classes `ONNXPipelineConfig` and `ONNXPipeline`.

Functions and Attributes of ONNXPipelineConfig

The `ONNXPipelineConfig` class contains the properties required for the package to perform downloading, exporting, augmenting, validation, and storing of an ONNX model. The class provides access to configuration properties using the dot operator. As a convenience, well-known configurations are provided as templates.

Parameters

This table describes the functions and properties of the `ONNXPipelineConfig` class.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Functions	Parameter Type	Returns	Description
`from_template(name, **kwargs)`	• `name` (`String`): The name of the template • `**kwargs`: template properties to override or add	Instance of `ONNXPipelineConfig`	A static function that creates an `ONNXPipelineConfig` object based on a predefined template given by the name parameter. You can use named arguments to override the template properties.
`show_templates()`	NA	List of existing templates	A static function that returns a list of existing templates by name.
`show_preconfigured()`	• `include_properties` (`bool, optional`): A flag indicating whether properties should be included in the results. Defaults to `False` so only names will be included by default. • `model_name` (`str, optional`): A model name to filter by when including properties. This argument will be ignored if `include_properties` is `False`. Otherwise only the properties of this model will be included in the results.	A list of preconfigured model names or properties.	Shows a list of preconfigured model names, or properties. By default, this function returns a list of names only. If the properties are required, pass the `include_properties` parameter as `True`. The returned list will contain a single dict where each key of the dict is the name of a preconfigured model and the value is the property set for that model. Finally, if only a single set of properties for a specific model is required, pass the name of the model in the `model_name` parameter (the `include_properties` parameter should also be `True`). This will return a list of a single dict with the properties for the specified model.

Template Properties

The text template has configuration properties shown below:

```
"do_lower_case": true,
"post_processors":[{"name":"Pooling","type":"mean"},{"name":"Normalize"}]
```

> **Note**
>
> All other properties in the Properties table will take the default values. Any property without a default value must be provided when creating the `ONNXPipelineConfig` instance.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Properties

This table shows all properties that can be configured. preconfigured models already have these properties set to specific values. Templates will use the default values unless a user overrides it when using the `from_template` function on `ONNXPipelineConfig`.

Property	Description
`post_processors`	An array of post_processors that will be loaded after the model is loaded or initialized. The list of known and supported post_processors is provided later in this section. Templates may define a list of post_processors for the types of models they support. Otherwise, an empty array is the default.
`max_seq_length`	This property is applicable for text-based models only. The maximum length of input to the model as number of tokens. There is no default value. Specify this value for models that are not preconfigured.
`do_lower_case`	Specifies whether or not to lowercase the input when tokenizing. The default value is `True`.
`quantize_model`	Perform quantization on the model. This could greatly reduce the size of the model as well as speed up the process. It may however result in different results for the embedding vector (against the original model) and possibly small reduction in accuracy. The default value is `False`.
`distance_metrics`	An array of names of suitable distance metrics for the model. The names must be name of distance metrics used for Oracle vector distance operator. Only used when exporting the model to the database. Supported list is ["EUCLIDEAN","COSINE","MANHATTAN","HAMMING","DOT","EUCLIDEAN_SQUARED", "JACCARD"]. The default value is an empty array.
`languages`	An array of language (Abbreviation) supported in the Database. Only used when exporting the model to the database. For a supported list of languages, see Languages. The default value is an empty array.
`use_float16`	Specifies whether or not to convert the exported onnx model to float16. The default value is `False`.

Properties of post_processors

This table describes the built-in post_processors and their configuration parameters.

post_processor	Parameters	Description
`Pooling`	• `name`: `Pooling`. • `type`: Valid values should be `mean`(Default), `max`, `cls`	The Pooling post_processor summarizes the output of the transformer model into a fixed-length vector.
`Normalize`	• `name`: Specify `Normalize`	The Normalize post_processor bounds the vector values to a range using L2 normalization.

Copyright © 2024, 2026, Oracle and/or its affiliates.

post_processor	Parameters	Description
`Dense`	• `name`: Dense • `in_features`: Input feature size • `out_features`: Output feature size • `bias`: Whether to learn an additive bias. The default value is `True`. • `activation_function`: Activation function of the dense layer. Currently only supports `Tanh` as the activation function.	Applies transformation to the incoming data.

Example: Configure post_processors

In this example, you override the post_processors in the sentence-transformers template with a `Max` Pooling post_processor followed by Normalization.

```
config = ONNXPipelineConfig.from_template("text")
config.post_processors = [{"name":"Pooling","type":"max"},
{"name":"Normalize"}]
```

Functions and Attributes of ONNXPipeline

Use the `ONNXPipeline` class to convert transformer models to the ONNX format with post_processing steps embedded into the final model.

Parameters

This table describes the signature and properties of the `ONNXPipeline` class.

Functions	Parameters	Description
`ONNXPipeline(model_name,configuration=None,settings={})`	• `model_name(str)`: Name of the model to be used. For example, `medicalai/ClinicalBERT`. • `configuration (ONNXPipelineConfig, optional)`: An initialized `ONNXPipelineConfig` object. This parameter must be specified when using a template. If not specified, the model is assumed to be a preconfigured model. • `settings (dict, optional)`: A dictionary of global settings that control operations such as logging levels, download behavior, and file locations.	Creates a new instance of the `ONNXPipeline` class.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Settings

The settings object is a dictionary passed to the `ONNXPipeline` class. It provides global properties used for non-model-specific operations, such as logging and model download behavior.

For example:

```
pipe = ONNXPipeline(model_name, config, settings={"ignore_safetensors_error":
True})
```

where: `ignore_safetensors_error=True` allows execution to continue even if the requested model does not provide a `safetensors` version.

Property	Type	Default Value	Description
`cache_dir`	`str`	`$HOME/.cache/OML`	Directory where Hugging Face models are downloaded and cached. Model files will be downloaded to directories relative to the `cache_dir`. If the directory does not exist at execution time, it will be created automatically.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Property	Type	Default Value	Description
logging_level	str	ERROR	The level of severity for log messages. Valid values are ['DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL'].

Note

This log level is also applied globally to all Python

Copyright © 2024, 2026, Oracle and/or its affiliates.

Property	Type	Default Value	Description
			packages and is also mapped to the ONNX Runtime libraries.
`force_download`	`bool`	`False`	Forces downloading model files from Hugging Face even if they already exist in the local cache.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Property	Type	Default Value	Description
`local_model_path`	`str`	`None`	Path to a local copy of a model to be exported instead of downloading the model from Hugging Face. Must be a valid directory path.
`cleanup`	`bool`	`True`	Controls whether intermediate files generated during model export are removed after execution. If set to `False`, intermediate export files are retained.
`ignore_checksum_error`	`bool`	`False`	Ignores any errors caused by mismatch in checksums when using preconfigured models.
`ignore_safetensors_error`	`bool`	`False`	Ignores errors when the requested model does not have a `safetensors` version available.

Functions

This table describes the function and properties of the `ONNXPipeline` class.

Function	Parameters	Description
`export2file(export_name,output_dir=None)`	• `export_name(string)`: Name of the exported file. The file is saved with the `.onnx` extension. • `output_dir(string)`: Output directory where the file will be saved. If not specified, the file is saved in the current working directory.	Exports the model to a local `.onnx` file.
`export2db(export_name)`	• `export_name(string)`: Name used for the mining model object in the database. The name must comply with database object naming rules.	Exports the model to the database.

Example: Preconfigured Model

This example illustrates the preconfigured embedding model that comes with the Python package. You can use this model without any additional configurations.

```
"sentence-transformers/distiluse-base-multilingual-cased-v2": {
        "max_seq_length": 128,
        "do_lower_case": false,
        "post_processors":[{"name":"Pooling","type":"mean"},
{"name":"Dense","in_features":768, "out_features":512, "bias":true,
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
"activation_function":"Tanh"}],
        "quantize_model":true,
        "distance_metrics": ["COSINE"],
        "languages": ["ar", "bg", "ca", "cs", "dk", "d", "us", "el", "et",
"fa", "sf", "f", "frc", "gu", "iw", "hi", "hr", "hu", "hy", "in", "i", "ja",
"ko", "lt",
                      "lv", "mk", "mr", "ms", "n", "nl", "pl", "pt", "ptb",
"ro", "ru", "sk", "sl", "sq", "lsr", "s", "th", "tr", "uk", "ur", "vn",
"zhs", "zht"]
    }
```

12.3 ONNX Pipeline Models: Text Embedding

ONNX pipeline models provides text embedding models that accepts text as input and produces embeddings as output. The pipeline models also provide the necessary pre-processing and post-processing needed for the text.

> **Note**
>
> - This API is updated for 23.7. The version used for 23.6 and below used python packages EmbeddingModel and EmbeddingModelConfig. These packages are replaced with ONNXPipeline and ONNXPipelineConfig respectively. Oracle recommends that you use the latest version. You can find the details of the deprecated python classes in Python Classes to Convert Pretrained Models to ONNX Models (Deprecated).
> - This feature will **only** work on OML4Py 2.1 client. It is not supported on the OML4Py server.
> - In-database embedding models must include tokenization and post-processing. Providing only the core ONNX model to `DBMS_VECTOR.LOAD_ONNX_MODEL` is insufficient because you need to handle tokenization externally, pass tensors into the SQL operator, and convert output tensors into vectors.
> - Oracle is providing a Hugging Face all-MiniLM-L12-v2 model in ONNX format, available to download directly to the database using `DBMS_VECTOR.LOAD_ONNX_MODEL`. For more information, see the blog post Now Available! Pre-built Embedding Generation model for Oracle Database 23ai.

If you do not have a pretrained embedding model in ONNX-format to generate embeddings for your data, Oracle offers a Python package that downloads pretrained models from an external source, converts the model to ONNX format augmented with pre-processing and post-processing steps, and imports the resulting ONNX-format model into Oracle AI Database. Use the `DBMS_VECTOR.LOAD_ONNX_MODEL` procedure or `export2db()` function to import the file as a mining model. (`export2db()` is a method on the ONNXPipeline object). Then leverage the in-database ONNX Runtime with the ONNX model to produce vector embeddings.

At a high level, the Python package performs the following tasks:

- Downloads the pretrained model from external source (example: from Hugging Face repository) to your system
- Augments the model with pre-processing and post-processing steps and creates a new ONNX model
- Validates the augmented ONNX model

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Loads into the database as a data mining model or optionally exports to a file

The Python package can take any of the models in the preconfigured list as input. Alternatively, you can use the built-in template that contains common configurations for certain groups of models such as text or image models. To understand what a preconfigured list, what a built-in template is, and how to use them, read further.

Limitations

This table describes the limitations of the Python package.

> **Note**
>
> This feature is available with the OML4Py 2.1 client only.

Parameter	Description
`Transformer Model Type`	Supports transformer models that supports text embedding.
`Model Size`	Quantization can help reduce the size.
`Tokenizers`	Must be either `BERT`, `GPT2`, `SENTENCEPIECE`, `CLIP`, or `ROBERTA`.

Preconfigured List of Models

Preconfigured list of models are common models from external resource repositories that are provided with the Python package. The preconfigured models have an existing specification. Users can create their own specification using the text template as a starting point. To get a list of all model names in the preconfigured list, you can use the `show_preconfigured` function.

Templates

The Python package provides built-in text template for you to configure the pretrained models with pre-processing and post-processing operations. The template has a default specification for the pretrained models. This specification can be changed or augmented to create custom configurations. The text template uses Mean Pooling and Normalization as post-processing operations by default.

Converting Text Models to ONNX Format and Storing them as a File or in a Database:

To use the Python package `oml.utils`, ensure that you have the following:

- OML4Py 2.1 Client running on Linux X64 for On-Premises Databases
- Python 3.12 (the earlier versions are not compatible)
- For the full end-to-end example, see Convert Pretrained Models to ONNX Model: End-to-End Instructions.

1. Start Python in your work directory.

```
python3


Python 3.12.6 | (main, Feb 27 2024, 17:35:02) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

2. On the OML4Py client, load the Python classes:

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig
```

3. You can get a list of all preconfigured models by running the following:

```
ONNXPipelineConfig.show_preconfigured()
```

4. To get a list of available templates:

```
ONNXPipelineConfig.show_templates()
```

5. Choose from:

- Generating a text pipeline using a preconfigured model "sentence-transformers/all-MiniLM-L6-v2" and save it in a local directory:

```
#generate from preconfigured model "sentence-transformers/all-MiniLM-L6-
v2"
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
pipeline.export2file("your_preconfig_file_name",output_dir=".")
```

- Generating a text pipeline using a preconfigured model "sentence-transformers/all-MiniLM-L6-v2" and save it in the database:

```
#generate from preconfigured model "sentence-transformers/all-MiniLM-L6-
v2"
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2")
pipeline.export2db("your_preconfig_file_name")
```

> **Note**
>
> In order for export2db( ) to succeed, a connection is required to the database. It is assumed that you have provided your credentials in the following code and the connection is successful.
>
> ```
> import oml
> oml.connect(user="username",password="password",dsn="dsn");
> ```

- Generating a text pipeline with a template and save it in a local directory

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig
config   = ONNXPipelineConfig.from_template("text", max_seq_length=256,
           distance_metrics=["COSINE"], quantize_model=True)
pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-
v2",
           config=config)
pipeline.export2file("your_preconfig_model_name", output_dir=".")
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Let's understand the code:

`from oml.utils import ONNXPipeline, ONNXPipelineConfig`

This line imports two classes, `ONNXPipeline` and `ONNXPipelineConfig`.

In the preconfigured models first example, where the ONNX format model is saved in a particular directory:

- `pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2")` creates an instance of the `ONNXPipeline` class, loading a pretrained model specified by the `model_name` parameter. `pipeline` is the text embedding model object. `sentence-transformers/all-MiniLM-L6-v2` is the model name for computing sentence embeddings. This is the model name under Hugging Face. Oracle supports models from Hugging Face.
- The `export2file` command creates an ONNX format model with a user-specified model name in the file. `your_preconfig_file_name` is a user defined ONNX model file name.
- `output_dir="."` specifies the output directory where the file will be saved. The `"."` denotes the current directory (that is, the directory from which the script is running).

In the preconfigured models second example, where the ONNX model is saved in the database:

- `pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2")` creates an instance of the `ONNXPipleline` class, loading a pretrained model specified by the `model_name` parameter. `pipeline` is the text embedding model object. `sentence-transformers/all-MiniLM-L6-v2` is the model name for computing sentence embeddings. This is the model name under Hugging Face. Oracle supports models from Hugging Face.
- The `export2db` command creates an ONNX format model with a user defined model name in the database. `your_preconfig_model_name` is a user defined ONNX model name.

In the template example:

- `ONNXPipelineConfig.from_template("text", max_seq_length=256 distance_metrics=["COSINE"], quantize_model=True)`: This line creates a configuration object for an embedding model using a method called `from_template`. The `"text"` argument indicates the name of the template. The `max_seq_length=512` parameter specifies the maximum length of input to the model as number of tokens. There is no default value. Specify this value for models that are not preconfigured.
- `pipeline = ONNXPipeline(model_name="sentence-transformers/all-MiniLM-L6-v2", config=config)` initializes an `ONNXPipeliene` instance with a specific model and the previously defined configuration. The `model_name="all-MiniLM-L6-v2"` argument specifies the name or identifier of the pretrained model to be loaded.
- The `export2file` command creates an ONNX format model with a user defined model name in the file. `your_preconfig_model_name` is a user defined ONNX model name.
- `output_dir="."` specifies the output directory where the file will be saved. The `"."` denotes the current directory (that is, the directory from which the script is running).

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> - For models larger than 400MB, Oracle recommends quantization.
>
> **Quantization** reduces the model size by converting model weights from high-precision representation to low-precision format. The quantization option converts the weights to INT8. The smaller model size enables you to cache the model in shared memory further improving the performance.
> - The `.onnx` file is created with opset version 17 and ir version 8. For more information about these version numbers, see https://onnxruntime.ai/docs/reference/compatibility.html#onnx-opset-support.

6. Exit Python.

   ```
   exit()
   ```

7. Inspect if the converted models are present in your directory.

   > **Note**
   >
   > ONNX files are only created when `export2file` is used. If `export2db` is used, local ONNX files will be generated, and the model will be saved to the database.

   List all files in the current directory that have the extension `onnx`.

   ```
   ls -ltr *.onnx
   ```

   The Python utility package validates the embedding text model before you can run them using ONNX Runtime. Oracle supports ONNX embedding models that conform to `string` as input and `float32 [<batch size>,<vector dimension>]` as output.

   If the input or output of the model doesn't conform to the description, you receive an error during the import.

8. Choose from:

   - Load the ONNX model to your database using PL/SQL:

     a. You can load the ONNX model using this syntax:

     ```
     BEGIN
        DBMS_VECTOR.LOAD_ONNX_MODEL(
          directory => 'ONNX_IMPORT',
          file_name => 'all-MiniLM-L6.onnx',
          model_name => 'ALL_MINILM_L6');
     END;
     ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

b. Move the ONNX file named `your_template_file_name` to a directory on the database server, and create a directory on the file system and in the database for the import.

```
mkdir -p /tmp/models
sqlplus / as sysdba
alter session set container=ORCLPDB;
```

Apply the necessary permissions and grants. In this example, we are using a pluggable database named ORCLPDB.

```
-- directory to store ONNX files for import
CREATE DIRECTORY ONNX_IMPORT AS '/tmp/models';
-- grant your OML user read and write permissions on the directory
GRANT READ, WRITE ON DIRECTORY ONNX_IMPORT to OMLUSER;
-- grant to allow user to import the model
GRANT CREATE MINING MODEL TO OMLUSER;
```

- Load the ONNX model to your database using Python:

9. Verify the model is in the database:

At your operating system prompt, start SQL*Plus, connect to it :

```
sqlplus OML_USER/password@pdbname_medium;



SQL*Plus: Release 23.0.0.0.0 - Production on Wed May 1 15:33:29 2024
Version 23.4.0.24.05

Copyright (c) 1982, 2024, Oracle. All rights reserved.

Last Successful login time: Wed May 01 2024 15:27:06 -07:00

Connected to:
Oracle AI
                                  Database 26ai Enterprise Edition Release
23.0.0.0.0 - Production
Version 23.4.0.24.05
```

List the ONNX model in the database to make sure it was loaded:

```
SQL> select MODEL_NAME, ALGORITHM from user_mining_models WHERE
MODEL_NAME='YOUR_PRECONFIG_MODEL_NAME';


MODEL_NAME ALGORITHM
-----------------------------------
YOUR_PRECONFIG_MODEL_NAME ONNX
```

`DBMS_VECTOR.LOAD_ONNX_MODEL` is only needed if `export2file` was used to save the ONNX model file to the local system instead of using `export2db` to save the model in the database. The `DBMS_VECTOR.LOAD_ONNX_MODEL` imports the ONNX format model into the Oracle AI

Copyright © 2024, 2026, Oracle and/or its affiliates.

Database to leverage the in-database ONNX Runtime to produce vector embeddings using the `VECTOR_EMBEDDING` SQL operator.

See Also

- *Oracle AI Database SQL Language Reference* for information about the `VECTOR_EMBEDDING` SQL function
- *Oracle AI Database PL/SQL Packages and Types Reference* for information about the `IMPORT_ONNX_MODEL` procedure
- *Oracle AI Database PL/SQL Packages and Types Reference* for information about the `LOAD_ONNX_MODEL` procedure
- *Oracle Machine Learning for SQL Concepts* for more information about importing pretrained embedding models in ONNX format and generating vector embeddings
- https://onnx.ai/onnx/intro/ for ONNX documentation

12.4 ONNX Pipeline Models: Image Embedding

ONNX pipeline models provides image embedding models that accepts image as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing.

Image Embedding Pipeline

1. **Input** : Input to the pipeline is an image. Each image is represented as an array of bytes. This can be a BLOB when using SQL or a similar in-memory byte representation like `io.ByetIO` in Python.
2. **Pre-Processing** : The image embedding pipeline utilizes an Image Processor in the pre-processing step. The Image Processor is dependent on the model configuration in the Hugging Face repository. When you provide a pretrained image model (e.g. "google/vit-base-patch16-224"), the pipeline builder will look up the Image Processor class from the model's configuration and determine if it's supported by looking up the Image Processor name in pipeline builder templates. If a match is found then the specific Image Processor is loaded and configured according to the template. The following table shows the relationship between pipeline builder templates and their corresponding Image Processor classes. Image Processor classes are provided by the transformer library.

Note

You do not need to reference these templates when using preconfigured image models. However any non-preconfigured model is template driven and these templates can be used for such models.

Template	Image Processor Class
Image_ViT	transformers.ViTImageProcessor
Image_ConvNext	transformers.ConvNextImageProcessor

Copyright © 2024, 2026, Oracle and/or its affiliates.

Each Image Processor has several possible operations that will modify the input image in a specific way. Each of these operations is represented by a configuration in the Image Processor template. The modification of these operations and their respective configuration is not supported. You are able to view the configurations and should be aware of them. The following table shows the image operations and their respective configurations:

Table 12-1 Image Processor Configurations

Image Processor Operation	Description
Decode	Converts the compressed image to 3-D raster format.
Resize	Resizes the given image to the new shape.
Rescale	Rescales (element wise multiplication) an image with a numeric value.
Normalize	Normalizes (adjusts the intensity values of pixels to a desired range, often between 0 to 1) an image using the given mean and standard deviation (std) using the following formulation: *(image - mean) / std*
CenterCrop	Crops the image with a fixed size from the center.

> **Note**
>
> The Image pre-processors are considered to be fixed and modifications are not allowed. We are exposing these details for your information and understanding.

3. **Original Model**: Pre-trained pytorch model from Hugging Face repository. The repository also contain preconfigured models which have an existing specification. To get a list of all model names in the preconfigured list, you can use the `show_preconfigured` function. You can also create your own specification using the above mentioned templates as a starting point.
4. **Output** : The pipeline generates embeddings for each input image.

Image Embedding Examples

1. Exporting a pre-configured image model:

```
from oml.utils import ONNXPipeline
pipeline =ONNXPipeline("google/vit-base-patch16-224")
pipeline.export2file("your_preconfig_model_name")
```

This example uses the `google/vit-base-patch16-224` model, which is a pre-configured model shipped in OML4Py 2.1. The pipeline is exported to a file which will result in a file **your_preconfig_model_name.onnx** in the current directory. This model can optionally be exported directly to the database using the pipeline.export2db function. Although the exported pipeline will just produce embeddings, not a classification output, it is possible to train an OML classification model to take these embeddings and produce the desired classification.

2. Exporting a non pre-configured model with a template:

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> Refer to this section for understanding the templates

```
from oml.utils import ONNXPipeline,ONNXPipelineConfig
import oml
config = ONNXPipelineConfig.from_template("Image_ViT")
pipeline = ONNXPipeline("nateraw/food",config=config)
oml.connect("pyquser","pyquser",dsn="pydsn")
oml.export2db("your_preconfig_model_name")
```

In this example, since the `nateraw/food` model is not included as a preconfigured model in OML4Py 2.1, a template approach was chosen. Since this is a ViT based model, the ViT template is selected.

The process of loading a image model converted to a ONNX format is very similar to the ONNX format model generated from text. Refer to steps 6-9 from ONNX Pipeline Models : Text Embedding for understanding how to load a ONNX model and use it further.

12.5 ONNX Pipeline Models: CLIP Multi-Modal Embedding

ONNX pipeline models provides multi-modal embedding models that accepts both image and text as an input and produces embeddings as output. The pipeline models also provide the necessary pre-processing needed.

CLIP Multi-Modal Embedding Pipeline

CLIP models are multi-modal which allows you to perform embedding on both text and image input data. The main advantage of this model is to do image-text similarity. You can compare vectors related to text snippets and a given image, to understand which text can better describe given image. The pipeline generator will generate two ONNX pipeline models for a pretrained CLIP model, distinguished by their suffixes. The pipeline model for images is suffixed with _**img** and the model for text is suffixed with _**txt**. The same models with their suffixes will be loaded to the database when using *export2db*. For performing CLIP related tasks such as image-text similarity, both models will need to be used at inference time.

1. **Input**: CLIP models consist of two pipelines: an image embedding pipeline and a text embedding pipeline. The Image pipeline takes images as described in the Image Embedding Pipeline section of ONNX Pipeline Models: Image Embedding, and the text Pipeline takes text as described in ONNX Pipeline Models: Text Embedding (Text Embedding support was introduced in OML4Py 2.0).
2. **Pre-processing**: The image pipeline for CLIP models utilizes the same pre-processing strategy as described in the Image Embedding Pipeline section of ONNX Pipeline Models: Image Embedding. That is, an image processor that matches the model's configuration is utilized to prepare the images. The text pipeline utilizes a specific tokenizer: the **CLIPTokenizer (transformers.models.clip.CLIPTokenizer)**.
3. **Post-processing**: As a post-processing step, Normalization is added to both the image and text pipelines.
4. **Output**: Both models will produce vectors of the same shape that can then be compared using some similarity measure.

Copyright © 2024, 2026, Oracle and/or its affiliates.

CLIP Multi-Modal Embedding Examples

1. Exporting a pre-configured image model to a file:
   The following example will produce two pipelines called **clip_img.onnx** and **clip_txt.onnx**, which can be used for image and text embeddings respectively.

   ```
   from oml.utils import ONNXPipeline
   pipeline = ONNXPipeline("openai/clip-vit-large-patch14")
   pipeline.export2file("clip")
   ```

2. Exporting a pre-configured image model to the database:
   This example will produce two in-database models called **clip_img** and **clip_txt**, which can be used for image and text embeddings respectively.

   ```
   fromoml.utils import ONNXPipeline
   import oml
   pipeline = ONNXPipeline("openai/clip-vit-large-patch14")
   oml.connect("pyquser","pyquser",dsn="pydsn")
   pipeline.export2db("clip")
   ```

3. Exporting a non pre-configured with a template to a file:
   This example will work for clip models that are not preconfigured. It will create two files called **clip_14_img.onnx** and **clip_14_txt.onnx**.

   ```
   from oml.utils import ONNXPipeline, ONNXPipelineConfig
   config = ONNXPipelineConfig.from_template("multimodal_clip")
   pipeline = ONNXPipeline("openai/clip-vit-base-patch16",config=config)
   pipeline.export2file("clip_16")
   ```

For an end to end example of using CLIP to generate multi-modal embeddings, see Generate Multi-modal Embeddings Using CLIP.

12.6 ONNX Pipeline Models: Text Classification

ONNX pipeline models provides text classification models that accepts text strings as input and produces the probablity of the input string belonging to a specific label. The pipeline models also provide the necessary pre-processing and post-processing.

In addition to text embedding models, Hugging Face repository also hosts transformer models that can be used for text classification. These models are typically fine-tuned embedding models on which a classification "head" was appended. The head changes the output of the model from a vector of embeddings to a list of labels and probabilities. For text based classification tasks such as sentiment analysis, you can generate a Text Classification Pipeline using OML4Py 2.1.

Text Classification Pipeline

1. **Input**: Input to the text classification pipeline is provided in the form of a batch, or array, of 1 ore more text strings. Each text string provided in the input will correspond to an output list containing the labels and probabilities.
2. **Pre-Processing**: Similar to text embedding pipeline, the text classification pipeline also configures a tokenizer for tokenizing the text inputs. The following tokenizer classes are supported in OML4Py 2.1:

Copyright © 2024, 2026, Oracle and/or its affiliates.

Table 12-2 Tokenizer Classes available for Text Classifiaction Pipeline

Tokenizer Class	Tokenizer Type
`transformers.models.bert.BertTokenizer`	`BERT`
`transformers.models.clip.CLIPTokenizer`	`CLIP`
`transformers.models.distilbert.DistilBertTokenizer`	`BERT`
`transformers.models.gpt2.GPT2Tokenizer`	`GPT2`
`transformers.models.mpnet.MPNetTokenizer`	`BERT`
`transformers.models.roberta.tokenization_roberta.RobertaTokenizer`	`ROBERTA`
`transformers.models.xlm_roberta.XLMRobertaTokenizer`	`SENTENCEPIECE`

> **Note**
>
> A tokenizer is automatically configured based on the tokenizer class configured for the model on Hugging Face. Tokenizer classes are provided by the transformer library. If the tokenizer class configured for the model in Hugging Face is not supported by OML4Py 2.1, an error will be raised.

3. **Original Model**: The original model must be a pre-trained PyTorch model in Hugging Face repository or from the local file system. Models on the local file system must match the Hugging Face format.
4. **Post-Processing**: The text classification pipeline provides a softmax function for post-processing by default. The softmax function will normalize the set of scores (logits) produced by the model into a probability distribution where each value is normalized to a range of [0,1]and all values sum to 1. You can choose to not include the softmax post-processing by providing an empty list of post-processors when generating the pipeline in OML4Py 2.1.
5. **Output**: The output of the classification pipeline is a list of probabilities for each input string. The length of the list is equal to the total number of classification targets or labels.

The pipeline generator assigns output labels based on the output indices. Currently, the mapping between model outputs and labels defaults to the order in which the outputs are produced.

> **Note**
>
> Label metadata does not become part of the ONNX model. It is only applied when exporting the model to the db with export2db.

Text Classification Pipeline Examples

In order to generate ONNX pipeline models for image classification, `MiningFunction` class is used from the python utilities. The `MiningFunction` class can take one of the three values:

Copyright © 2024, 2026, Oracle and/or its affiliates.

EMBEDDING, CLASSIFICATION, and REGRESSION. You can choose one depending on the task which you would like to perform. The MiningFunction enum is defined as follows :

```
class MiningFunction(Enum) :
    EMBEDDING = 1
    CLASSIFCATION = 2
    REGRESSION = 3
```

Since you are working on text classification pipeline, you would choose CLASSIFICATION or 2 (Enum) for the class MiningFunction

> **Warning**
>
> EmbeddingModel and EmbeddingModelConfig are deprecated. Instead, please use ONNXPipeline and ONNXPipelineConfig respectively. The details of the deprecated classes can be found in Python Classes to Convert Pretrained Models to ONNX Models (Deprecated). If a you choose to use a deprecated class, a warning message will be shown indicating that the classes will be removed in the future and advising the user to switch to the new class.

1. Example for generating a text pipeline with a template:

```
from oml.utils import ONNXPipeline,ONNXPipelineConfig,MiningFunction
config = ONNXPipelineConfig.from_template("text",max_seq_length=512)
pipeline = ONNXPipeline("SamLowe/roberta-base-
go_emotions",config=config,function=MiningFunction.CLASSIFICATION)
pipeline.export2file("emotions","testouput")
```

2. Importing the text classification pipeline generated in the above example in the DB (SQL)

> **Note**
>
> The following code assumes that you have created a directory in PL/SQL named 'ONNX_IMPORT'

```
BEGIN
    DBMS_VECTOR.LOAD_ONNX_MODEL(
    'ONNX_IMPORT',
    'emotions.onnx',
    'emotions',

JSON('{"function":"classification","classificationProbOutput":"logits","inp
ut":{"input":["DATA"]},
        "labels":
["admiration","amusement","anger","annoyance","approval","caring","confusio
n","curiosity",

"desire","disappointment","disapproval","disgust","embarrassment","exciteme
nt","fear","gratitude",

"grief","joy","love","nervousness","optimism","pride","realization","relief
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
","remorse","sadness","surprise","neut"]}')
    );
END;
```

In this example the label metadata is being applied when invoking the `DBMS_VECTOR.LOAD_ONNX_MODEL` function to import the ONNX pipeline into the database. Please recall that the "labels" property is part of the JSON argument to the function and not a part of the ONNX file. The labels must be provided separately (if not provided only the numerical indices of the output will be used). When exporting directly to the database from OML4Py using the `export2db` function, you can either rely on the default label metadata provided in the model's configuration on Hugging Face, or provide your own label metadata via a template argument.

3. Example for exporting a text classification pipeline with default label metadata:

```
from oml.utils importONNXPipeline,ONNXPipelineConfig,MiningFunction
import oml
config =
ONNXPipelineConfig.from_template("text",max_seq_length=512,labels=["admirat
ion","amusement","anger","annoyance","approval","caring",

"confusion","curiosity","desire","disappointment","disapproval","disgust","
embarrassment","excitement",

"fear","gratitude","grief","joy","love","nervousness","optimism","pride","r
ealization","relief",

"remorse","sadness","surprise","neut"])
pipeline = ONNXPipeline("SamLowe/roberta-base-
go_emotions",config=config,function=MiningFunction.CLASSIFICATION)
oml.connect("pyquser","pyquser",dsn="pydsn")
pipeline.export2db("emotions")
```

In this example the pipeline is exported with label metadata that you provided in the template. This example is a combination of the examples provided in the previous two steps.

4. Example for scoring a text classification pipeline:

```
select prediction(emotions using 'Today is a good day' as data),
        prediction_probability(emotions using 'Today is a good day'as
data) from dual;
```

12.7 ONNX Pipeline Models: Reranking Pipeline

ONNX pipeline models provide a reranking pipeline that calculates similarity score for a given pair of texts.

Reranking Pipeline

Reranking models (also known as cross-encoders or rerankers) calculate a similarity score for given pairs of texts. Instead of encoding the input text into a fixed-length vector as text embedding models do, rerankers encode two input texts simultaneously and produces a similarity score for the pair. By default it outputs a logit which can be converted to a number between 0 and 1 by applying a sigmoid activation function on it. Although you can compute

Copyright © 2024, 2026, Oracle and/or its affiliates.

similarity score between two texts through embedding models by first computing the embeddings of the texts and then applying cosine similarity, the re-ranker models usually provide superior performance and they can be used to re-rank the top-k results from an embedding model.

1. **Input**: There are two inputs to the reranking pipeline named `first_input` and `second_input`. Each contains an array of one or more text strings. The two inputs should have equal number of text strings. An array will be produced for each pair of texts from `first_input` and`second_input`. For example, for the following input pairs two arrays are produced, one for the pair`'hi'` and `'hi'` and the other for the pair `'halloween'` and `'hello'`:
   `{'first_input':['hi','halloween'],'second_input':['hi','hello']}`
2. **Pre-Processing**:
   Pre-processing for reranking pipeline includes tokenization. The tokenizer class: `transformers.models.xlm_roberta.tokenization_xlm_roberta.XLMRobertaTokenizer` is supported in OML4Py 2.1 for reranking models.
3. **Output**: The output of a reranking pipeline is an array of similarity scores, one for each input text pair. For the above example in the Input section, the output similarity score is 9.290878 for the pair `'hi'` and `'hi'` and 4.3913193 for the pair `'halloween'` and `'hello'`.

Reranking Pipeline Examples

1. Generating a Reranking Pipeline:

```
from oml.utils import ONNXPipeline, ONNXPipelineConfig
from oml import MiningFunction, ONNXPipeline
mn = 'BAAI/bge-reranker-base'
em = ONNXPipeline(mn, function=MiningFunction.REGRESSION)
mf = 'bge-reranker-base'
em.export2file(mf)
```

2. Importing the reranker model to the database:

```
BEGIN
DBMS_VECTOR.LOAD_ONNX_MODEL('DM_DUMP','bge-reranker-
base.onnx','doc_model', JSON('{"function" : "regression",
                               "regressionOutput" : "output", "input":
{"first_input": ["DATA1"],"second_input": ["DATA2"]}}'));
END;
```

3. Obtain the similarity score in the database:

```
SELECT prediction(doc_model USING 'what is panda?' as DATA1, 'hi' as
DATA2) from dual;
```

   The above example produces a similarity score between two text strings using the re-ranker ONNX pipeline imported in step 2. The score range from -infinity to +infinity. Score close to +infinity indicates a high similarity between the two strings, and score closer to -infinity indicate low similarity between the strings. A sigmoid function can map the similarity

Copyright © 2024, 2026, Oracle and/or its affiliates.

score to a float value in the range [0,1]. A sample output score corresponding to the above example is shown here:

```
PREDICTION(DOC_MODELUSING'WHATISPANDA?'ASDATA1,'HI'ASDATA2)
-----------------------------------------------------------
                      -7.73E+000
```

12.8 Load Custom Models from The Local Filesystem

Custom models that have been fine-tuned from pre-trained Hugging Face models and reside on the local filesystem can be converted to the ONNX format using the new `local_model_path` setting. This setting can also be useful in cases where networking is unavailable and a pre-trained model has previously been downloaded.

The model directory for a custom model must follow the following format:

```
<basedir>/<model_file_subfolders>
```

- basedir: This is the base directory for the model sub folders and is passed as the `local_model_path` settings parameter. This parameter is described in the Generate the Augmented ONNX File section. It can be an absolute or relative path. If relative, it will be relative to where Python was started.
- model_file_subfolders: These nested folders are created based on the model's name. For each '/' character in the model's name, you should create a new subdirectory.

Example 12-1 Local Directory for Custom Model Files

For example, if the model name is `mymodel`, only one directory named `mymodel` is expected. However, if the model name is `user/mymodel`, you should create a parent directory called `user` with a subdirectory named `mymodel`. All model files should be placed in the final subdirectory (`mymodel` in this case).

```
user
    mymodel
```

Generate the Augmented ONNX File

When initializing the EmbeddingModel object, use the settings parameter to pass a new property `local_model_path`, which identifies the `basedir` for the model. For example, if the `basedir` for the model on the local filesystem is `$HOME/sample-model`, and there is a custom model named `my-model`, you can achieve this with the following Python code:

```
from oml.utils import EmbeddingModel
import os

em = EmbeddingModel('my-
model',settings={'local_model_path':f'{os.environ["HOME"]}/sample-model'})
em.export2file('my-onnx-model','.')
```

At the end of this process, you should find a file named `my-onnx-model.onnx` in the current directory.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> The above code is the same for exporting to the database except that export2db is used instead of export2file. For more information, see ONNX Pipeline Models: Text Embedding. The code is compatible with all downloaded pretrained models, preconfigured models, and templates.

Error and Warning Conditions

Table 12-3 Error and Warning Conditions

Error (or Warning) Conditions	Error (or Warning) Message	Notes
force_download specified	force_download cannot be true when specifying 'local_model_path'	force_download is mutually exclusive with this feature so it cannot be specified when local_model_path has a value.
local_model_path is not a directory	'{local_path}' specified by 'local_model_path' must be an existing directory.	`{local_path}` is the value specified in `local_model_path` . This message is returned when the value of local_model_path is not an existing directory.
Missing model config file	Expected local model folder {model_path} to contain the file config.json but it was not found.	`{model_path}` is the full path to the local model folder. This message is returned when the local model folder does not contain a file named config.json.
Missing model files	Expected local model folder {model_path} to at least contain one of model.safetensors or pytorch_model.bin files. Neither were found.	`{model_path}` is the full path to the local model folder. This message is returned when the local model folder does not contain either a pytorch_mode.bin file OR at least one file that does not end in .safetensors.
preconfigured model used (Warning)	Warning: checksum validation is disabled when loading preconfigured models from a local path.	This warning is shown when a preconfigured model is loaded from a local path.

12.9 Support For Large ONNX Format Model

OML4Py supports large ONNX format models outside of the database. A large model is divided into a single ONNX file and one or more external data files. The maximum size limit for an ONNX file is 2GB, so for models larger than 2GB, the ONNX file must be split into smaller parts.

You can leverage ONNXPipelines to utilize external data. External data can be applied in two ways:

- If model size is less than 1 GB: If the model size is less than 1 GB, manually activate the use of external data, set the parameter `use_external_data` to True. It can be used with both preconfigured models and non-configured models.

Copyright © 2024, 2026, Oracle and/or its affiliates.

  - By using a preconfigured model.

    ```
    from oml.utils import ONNXPipeline,ONNXPipelineConfig
    config = ONNXPipelineConfig.from_template("text", max_seq_length=512,
    use_external_data=True)
    pipeline = ONNXPipeline("Snowflake/snowflake-arctic-embed-s",config)
    ```

  - If the model is not configured, you can use templates.

    ```
    from oml.utils import ONNXPipeline, ONNXPipelineConfig
    config = ONNXPipelineConfig.from_template("text", max_seq_length=512,
    use_external_data=True)
    pipeline = ONNXPipeline("Snowflake/snowflake-arctic-embed-s", config)
    ```

- If model size exceeds 1 GB: Loading a model of size greter than 1 GB without quantization enabled will default to using external data. For models greater than 2 GB with quantization enabled (`quantize_model=True`), will result in an error, instead of activating the external data. The following example, which uses a large multilingual-e5 model, will default to using the external data:

  ```
  from oml.utils import ONNXPipeline,ONNXPipelineConfig
  config = ONNXPipelineConfig.from_template("text", max_seq_length=512)
  pipeline = ONNXPipeline("intfloat/multilingual-e5-large", config)
  pipeline.export2db("multilingual")
  ```

> **Note**
>
> You can always manually activate the use of external data by setting the parameter `use_external_data=True` for models of any size.

The `export2file` function will create a zip file instead of a single ONNX file. This zip file will contain:

- An ONNX file: This file corresponds to the model graph.
- One or more .data files: These files will contain the tensors and any other external data associated with the model.
- A JSON file: This file details which tensor belongs to a specific location on each .data file.

An example of such a zip file would be as follows:

- large_model.onnx
- large_model_data_0.data, large_model_data_1.data
- large_model_external_data.json

Model size support in ONNX pipeline API:

- Less than 1 GB: Supported with optimal performance.
- 1 GB to 2 GB: If quantization is enabled, full support is provided; otherwise, the model is exported using external initializers.
- 0.4 GB to 2 GB (without quantization): A warning is displayed, recommending quantization to improve performance.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- Greater than 2 GB: Models over 2 GB are supported only with quantization disabled, using external data. With quantization enabled, the limit is 2 GB.

12.10 Model Summary

A model summary provides a concise overview of the model. To obtain the summary after exporting the model, call the `summary()` function.

The model summary can be retrieved from ONNX pipeline by calling the `summary()` function after executing either `export2db()` or `export2file()`. The summary includes details such as:

- Whether the model was quantized
- Whether external data was used
- The tokenizer type, pre-processors and post-processors applied
- Other relevant metadata

```
from oml.utils import ONNXPipeline,ONNXPipelineConfig
model_name="sentence-transformers/all-MiniLM-L6-v2"
em = ONNXPipeline(model_name=model_name,settings={"logging_level":"ERROR"})
em.export2db("allminil6v2")
em.summary()


{'description': "Original model sentence-transformers/all-MiniLM-L6-v2
augmented with pre/post processors. Generated by OML4Py ver. 2.1.
Generated model is not quantized. Generated model doesn't use external data.
Model tokenizer is <class
'transformers.models.bert.tokenization_bert.BertTokenizer'>.
Model post processors include: mean_pooling,mean_pooling.", 'producer':
'OML4Py 2.1'}
```

If the model was exported to the database, the summary is also included in the view storing ONNX metadata information:

```
SQL> select * from DM$VJALLMINIL6V2;

METADATA
--------------------------------------------------------------------------------
--
{"function":"embedding","embeddingOutput":"embedding","input":{"input":
["DATA"]}
,"pooling":"mean","normalization":true,"maxSequenceLength":256,"suitableDistan
ce
Metrics":["COSINE","DOT","EUCLIDEAN"],"languages":["us"],"modelDescription":
{"de
scription":"Original model sentence-transformers/all-MiniLM-L6-v2 augmented
with
 pre/post processors. Generated by OML4Py ver. 2.1. Generated model is not
quant
ized. Generated model doesn't use external data. Model tokenizer is <class
'tran
sformers.models.bert.tokenization_bert.BertTokenizer'>. Model post processors
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
in
clude: mean_pooling,mean_pooling.","producer":"OML4Py 2.1"}}
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

13
OML4Py Metrics

OML4Py provides a metric module that contains functions to compute metrics for model evaluation in the database.

For information on the `oml.metrics.metric_function` class attributes and methods, call `help(oml.metrics.metric_function)` or see Oracle Machine Learning for Python API Reference. For example, to learn more about `confusion_matrix`, call `help(oml.metrics.confusion_matrix)`.

The following table lists the functions supported by the `oml.metrics` in the metrics module on the prediction data frame.

All functions in the table require `pred_df(prediction data frame)`, `y_true(true values)`, and `y_pred(predicted values)` as input arguments, where:

- `pred_df(prediction data frame)`: Data frame that contains the ground truth or actual values, the predicted values, and optionally the sample weights. If the data frame does not contain the required columns the function will throw an exception. The data frame can contain any number of columns; however, only the columns specified in the parameters will be used for computation. There are no restrictions on the column names.

  The target and prediction columns can be of any type, except `oml.Bytes`, when performing classification or regression metrics. Additionally, for regression metrics, both the prediction and true value columns must not be of the type `oml.String`. If a weighted metric is calculated, both the data frame and the metric must include the weight column. The weight column must be of type `oml.float` or `oml.integer`, and its sum must be greater than zero.
- `y_true(true values)`: The name of the column that contains the target (actual) values.
- `y_pred(predicted values)`: The name of the column that contains the predicted values.

Table 13-1 Metrics Module

Metric Function	Description
confusion_matrix	Computes the confusion matrix.
precision_score	Computes the precision.
recall_score	Computes the recall.
f1_score	Computes the f1 score.
accuracy_score	Computes the accuracy.
balanced_accuracy_score	Computes the balanced accuracy.
roc_auc_score	Computes the Area Under the Receiver Operating Characteristic Curve (ROC AUC).
mean_squared_error	Computes the mean squared error.
mean_squared_log_error	Computes the mean squared log error.
mean_absolute_error	Computes the mean absolute error.
median_absolute_error	Computes the median absolute error.
r2_score	Computes the coefficient of determination.

Copyright © 2024, 2026, Oracle and/or its affiliates.

To learn more about the metric functions, see Oracle Machine Learning for Python API Reference.

Example 13-1 Using the *confusion matrix* Function.

This example uses the function `confusion_matrix` to compute the confusion matrix for the provided DataFarme.

```
import oml
import pandas as pd
from oml.metrics import confusion_matrix

y_true = [1, 0, 1, 0, 1, 1, 0, 1, 1, 0]
y_pred = [1, 1, 1, 0, 0, 1, 1, 0, 0, 0]
sample_weight = [1, 2, 1, 1, 1, 1, 1, 3, 1, 1]

pdf = pd.DataFrame({'y_true': y_true, 'y_pred': y_pred, 'sample_weight':
sample_weight})
df = oml.create(pdf, 'test_class')
confusion_matrix(df, 'y_true', 'y_pred', sample_weight='sample_weight')


array([[2, 3], [5, 3]])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

14

Embedded Python Execution

Embedded Python Execution is a feature of Oracle Machine Learning for Python that allows you to invoke user-defined Python functions directly in an Oracle AI Database instance.

Embedded Python Execution is available on:

- Oracle Autonomous AI Database, where pre-installed Python packages can be used, via Python, REST and SQL APIs.
- Oracle AI Database on premises, ExaCS, ExaC@C, DBCS, and Oracle AI Database deployed in a compute instance, where the user can custom install third-party packages to use with EPE, via Python and SQL APIs.

Embedded Python Execution is described in the following topics:

Topics:

- About Embedded Python Execution
  With Embedded Python Execution, you can invoke user-defined Python functions in Python engines spawned and managed by the Oracle AI Database instance.
- Parallelism with OML4Py Embedded Python Execution
  OML4Py embedded Python execution allows users to invoke user-defined functions from Python, SQL, and REST interfaces using Python engines spawned and controlled by the Oracle Autonomous AI Database environment.
- Datastores Supporting Embedded Python Execution
  OML4Py includes the datastore views supporting Embedded Python Execution. You can use these datastore views with the Embedded Python Execution APIs to work with the datastores and their contents.
- Script repository for user-defined Python functions supporting EPE
  OML4Py includes the script repository views supporting Embedded Python Execution. You can use these script repository views with the Embedded Python Execution APIs to work with the script repository and their contents.
- Python API for Embedded Python Execution
  You can invoke user-defined Python functions directly in an Oracle AI Database instance by using Embedded Python Execution functions.
- SQL API for Embedded Python Execution with On-premises AI Database
  SQL API for Embedded Python Execution with On-premises AI Database has SQL interfaces for Embedded Python Execution and for datastore and script repository management.
- SQL API for Embedded Python Execution with Autonomous AI Database
  The SQL API for Embedded Python Execution with Autonomous AI Database provides SQL interfaces for setting authorization tokens, managing access control list (ACL) privileges, executing Python scripts, and synchronously and asynchronously running jobs.

Copyright © 2024, 2026, Oracle and/or its affiliates.

14.1 About Embedded Python Execution

With Embedded Python Execution, you can invoke user-defined Python functions in Python engines spawned and managed by the Oracle AI Database instance.

Embedded Python Execution is available in Oracle Autonomous AI Database.

In Oracle Autonomous AI Database, you can use:

- – An OML Notebooks Python interpreter session (see Use the Python Interpreter in a Notebook Paragraph)
  - – REST API for Embedded Python Execution
  - – SQL API for Embedded Python Execution with Autonomous AI Database

In an on-premises Oracle AI Database, you can use:

- In an on-premises Oracle AI Database, you can use:
  - – Python API for Embedded Python Execution
  - – SQL API for Embedded Python Execution with On-premises AI Database

The following topic compares the four Embedded Python Execution APIs.

Topics:

14.2 Parallelism with OML4Py Embedded Python Execution

OML4Py embedded Python execution allows users to invoke user-defined functions from Python, SQL, and REST interfaces using Python engines spawned and controlled by the Oracle Autonomous AI Database environment.

The user-defined functions can be invoked in a data-parallel and task-parallel manner with multiple Python engines, with output formats including structured data, XML, JSON, and PNG images.

Oracle Autonomous AI Database provides different service levels to manage the load on the system by controlling the degree of parallelism jobs can use:

- LOW - the default, with maximum 2 degrees of parallelism
- MEDIUM - maximum of 4 degrees of parallelism, and allows greater concurrency for job processing
- HIGH - maximum of 8 degrees of parallelism but significantly limits the number of concurrent jobs

Parallelism applies to:

- `oml.row_apply`, `oml.group_apply`, and `oml.index_apply` using the Python API for embedded Python execution
- `pyqRowEval`, `pyqGroupEval`, and `*pyqIndexEval` using the SQL API for embedded Python execution
- `row-apply`, `group-apply`, `index-apply` using the REST API for embedded Python execution

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> `pyqIndexEval` is available on Oracle Autonomous AI Database only.

Setting Parallelism Using Embedded Python Execution

For the ADB Python API for Embedded Python Execution:

The parallel parameter specifies the preferred degree of parallelism to use in the embedded Python execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or `0` for no parallelism
- `True` for the default data parallelism

Setting the argument `parallel=True` corresponds to service level defined in the notebook interpreter. The argument `parallel=x` is limited by the service level. For instance, the maximum number of parallel engines allowed by the MEDIUM service level is 4, therefore selecting `parallel=6` effectively results in `parallel=4`.

For the ADB SQL API for Embedded Python Execution:

The argument `oml_parallel_flag` and `oml_service_level` are used together to enable data-parallelism and task-parallelism. For more information see Special Control Arguments (Autonomous AI Database).

For the ADB REST API for Embedded Python Execution:

When executing a REST API Embedded Python Execution function, the `service` argument allows you to select the Autonomous Database service level to be used. For example, the `parallelFlag` is set to `true` in order to use database parallelism along with the MEDIUM service.

```
-d '{"parallelFlag":true,"service":"MEDIUM"}'
```

For more information see Specify a Service Level.

14.3 Datastores Supporting Embedded Python Execution

OML4Py includes the datastore views supporting Embedded Python Execution. You can use these datastore views with the Embedded Python Execution APIs to work with the datastores and their contents.

View	Description
ALL_PYQ_DATASTORES View	Contains information about the datastores available to the current user.
ALL_PYQ_DATASTORE_CONTENTS View	Contains information about the objects in the datastores available to the current user.
USER_PYQ_DATASTORES View	Contains information about the datastores owned by the current user.

Datastore views are described in the following topics.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Topics:

- ALL_PYQ_DATASTORE_CONTENTS View
  The ALL_PYQ_DATASTORE_CONTENTS view contains information about the contents of datastores that are available to the current user.
- ALL_PYQ_DATASTORES View
  The ALL_PYQ_DATASTORES view contains information about the datastores that are available to the current user.
- USER_PYQ_DATASTORES View
  The USER_PYQ_DATASTORES view contains information about the datastores that are owned by the current user.

14.3.1 ALL_PYQ_DATASTORE_CONTENTS View

The ALL_PYQ_DATASTORE_CONTENTS view contains information about the contents of datastores that are available to the current user.

Column	Datatype	Null	Description
DSOWNER	VARCHAR2(128)	NULL permitted	The owner of the datastore.
DSNAME	VARCHAR2(128)	NULL permitted	The name of the datastore.
OBJNAME	VARCHAR2(128)	NULL permitted	The name of an object in the datastore.
CLASS	VARCHAR2(128)	NULL permitted	The class of a Python object in the datastore.
OBJSIZE	NUMBER	NULL permitted	The size of an object in the datastore.
LENGTH	NUMBER	NULL permitted	The length of an object in the datastore. The length is 1 for all objects unless the object is a `list`, `dict`, `pandas.DataFrame`, or `oml.DataFrame`, in which case it is equal to `len(obj)`.
NROW	NUMBER	NULL permitted	The number of rows of an object in the datastore. The number is 1 for all objects except for `pandas.DataFrame` and `oml.DataFrame` objects, in which case it is equal to `len(df)`.
NCOL	NUMBER	NULL permitted	The number of columns of an object in the datastore. The number is `len(obj)` if the object is a `list` or `dict`, `len(obj.columns)` if the object is a `pandas.DataFrame` or `oml.DataFrame`, and 1 otherwise.

Example 14-1 Selecting from the ALL_PYQ_DATASTORE_CONTENTS View

This example selects all columns from the ALL_PYQ_DATASTORE_CONTENTS view. For the creation of the datastores in this example, see Example 8-14.

```
SELECT * FROM ALL_PYQ_DATASTORE_CONTENTS


DSOWNER   DSNAME        OBJNAME       CLASS              OBJSIZE  LENGTH
NROW  NCOL
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--------  ------------  ------------  -----------------  -------  ------
----  ----
OML_USER  ds_pydata     oml_boston    oml.DataFrame         1073     506
506    14
OML_USER  ds_pydata     oml_diabetes  oml.DataFrame          964     442
442    11
OML_USER  ds_pydata     wine          Bunch                24177       5
1     5
OML_USER  ds_pymodel    regr1         LinearRegression       706       1
1     1
OML_USER  ds_pymodel    regr2         oml.glm               5664       1
1     1
OML_USER  ds_wine_data  oml_wine      oml.DataFrame      1410     178
178    14
```

14.3.2 ALL_PYQ_DATASTORES View

The ALL_PYQ_DATASTORES view contains information about the datastores that are available to the current user.

Column	Datatype	Null	Description
DSOWNER	VARCHAR2(256)	NULL permitted	The owner of the datastore.
DSNAME	VARCHAR2(128)	NULL permitted	The name of the datastore.
NOBJ	NUMBER	NULL permitted	The number of objects in the datastore.
DSSIZE	NUMBER	NULL permitted	The size of the datastore.
CDATE	DATE	NULL permitted	The date on which the datastore was created.
DESCRIPTION	VARCHAR2(2000)	NULL permitted	A description of the datastore.
GRANTABLE	VARCHAR2(1)	NULL permitted	Whether or not the read privilege to the datastore may be granted. The value in this column is either T for True or F for False.

Example 14-2 Selecting from the ALL_PYQ_DATASTORES View

This example selects all columns from the ALL_PYQ_DATASTORES view. It then selects only the DSNAME and GRANTABLE columns from the view. For the creation of the datastores in these examples, see Example 8-14.

```
SELECT * FROM ALL_PYQ_DATASTORES;
```

```
DSOWNER    DSNAME          NOBJ    DSSIZE   CDATE       DESCRIPTION       G
--------   ------------   -----   -------   ---------   ---------------   -
OML_USER   ds_pydata          3     26214   18-MAY-19   python datasets   F
OML_USER   ds_pymodel         2      6370   18-MAY-19                     T
OML_USER   ds_wine_data       1      1410   18-MAY-19   wine dataset      F
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

This example selects only the DSNAME and GRANTABLE columns from the view.

```
SELECT DSNAME, GRANTABLE FROM ALL_PYQ_DATASTORES;
```

```
DSNAME       G
----------   -
ds_pydata    F
ds_pymodel   T
ds_wine_data F
```

14.3.3 USER_PYQ_DATASTORES View

The USER_PYQ_DATASTORES view contains information about the datastores that are owned by the current user.

Column	Datatype	Null	Description
DSNAME	VARCHAR2(128)	NULL permitted	The name of the datastore.
NOBJ	NUMBER	NULL permitted	The number of objects in the datastore.
DSSIZE	NUMBER	NULL permitted	The size of the datastore.
CDATE	DATE	NULL permitted	The date on which the datastore was created.
DESCRIPTION	VARCHAR2(2000)	NULL permitted	A description of the datastore.
GRANTABLE	VARCHAR2(1)	NULL permitted	Whether or not the read privilege to the datastore may be granted. The value in this column is either T for True or F for False.

Example 14-3 Selecting from the USER_PYQ_DATASTORES View

This example selects all columns from the USER_PYQ_DATASTORES view. For the creation of the datastores in these examples, see Example 8-14.

```
SELECT * FROM USER_PYQ_DATASTORES;
```

```
DSNAME       NOBJ   DSSIZE   CDATE       DESCRIPTION       G
----------   ----   ------   ---------   ---------------   -
ds_wine_data    1     1410   18-MAY-19   wine dataset      F
ds_pydata       3    26214   18-MAY-19   python datasets   F
ds_pymodel      2     6370   18-MAY-19                     T
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

This example selects only the DSNAME and GRANTABLE columns from the view.

```
SELECT DSNAME, GRANTABLE FROM USER_PYQ_DATASTORES;
```

```
DSNAME        G
----------    -
ds_wine_data  F
ds_pydata     F
ds_pymodel    T
```

14.4 Script repository for user-defined Python functions supporting EPE

OML4Py includes the script repository views supporting Embedded Python Execution. You can use these script repository views with the Embedded Python Execution APIs to work with the script repository and their contents.

View	Description
ALL_PYQ_SCRIPTS View	Describes the scripts that are available to the current user.
USER_PYQ_SCRIPTS View	Describes the user-defined Python functions in the script repository that are owned by the current user.

Script repository views are described in the following topics.

Topics:

- ALL_PYQ_SCRIPTS View
  The `ALL_PYQ_SCRIPTS` view contains information about the user-defined Python functions in the OML4Py script repository that are available to the current user.
- USER_PYQ_SCRIPTS View
  This view contains information about the user-defined Python functions in the OML4Py script repository that are owned by the current user.

14.4.1 ALL_PYQ_SCRIPTS View

The `ALL_PYQ_SCRIPTS` view contains information about the user-defined Python functions in the OML4Py script repository that are available to the current user.

Column	Datatype	Null	Description
OWNER	VARCHAR2(256)	NULL permitted	The owner of the user-defined Python function.
NAME	VARCHAR2(128)	NULL permitted	The name of the user-defined Python function.
SCRIPT	CLOB	NULL permitted	The user-defined Python function.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 14-4 Selecting from the ALL_PYQ_SCRIPTS View

This example selects the owner and the name of the user-defined Python function from the ALL_PYQ_SCRIPTS view.

```
SELECT owner, name FROM ALL_PYQ_SCRIPTS;



OWNER      NAME
--------   -----------------
OML_USER   create_iris_table
OML_USER   tmpqfun2
PYQSYS     tmpqfun2
```

This example selects the name of the user-defined Python function and the function definition from the view.

```
SELECT name, script FROM ALL_PYQ_SCRIPTS WHERE name = 'create_iris_table';



NAME                SCRIPT
-----------------
-----------------------------------------------------------------------
create_iris_table   "def create_iris_table():  from sklearn.datasets import
load_iris ...
```

14.4.2 USER_PYQ_SCRIPTS View

This view contains information about the user-defined Python functions in the OML4Py script repository that are owned by the current user.

Column	Datatype	Null	Description
NAME	VARCHAR2(128)	NOT NULL	The name of the user-defined Python function.
SCRIPT	CLOB	NULL permitted	The user-defined Python function.

Example 14-5 Selecting from the USER_PYQ_SCRIPTS View

This example selects all columns from USER_PYQ_SCRIPTS.

```
SELECT * FROM USER_PYQ_SCRIPTS;



NAME                SCRIPT
-----------------
-----------------------------------------------------------------------
create_iris_table   "def create_iris_table():    from sklearn.datasets import
load_iris ...
tmpqfun2            "def return_frame():     import numpy as np       import
pickle     ...
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

14.5 Python API for Embedded Python Execution

You can invoke user-defined Python functions directly in an Oracle AI Database instance by using Embedded Python Execution functions.

Python API for Embedded Python Execution is described in the following topics.

Topics:

- About Python API for Embedded Python Execution
- Run a User-Defined Python Function
  Use the `oml.do_eval` function to run a user-defined input function that explicitly retrieves data or for which external data is not required.
- Run a User-Defined Python Function on the Specified Data
  Use the `oml.table_apply` function to run a Python function on data that you specify with the `data` parameter.
- Run a Python Function on Data Grouped By Column Values
  Use the `oml.group_apply` function to group the values in a database table by one or more columns and then run a user-defined Python function on each group.
- Run a User-Defined Python Function on Sets of Rows
  Use the `oml.row_apply` function to chunk data into sets of rows and then run a user-defined Python function on each chunk.
- Run a User-Defined Python Function Multiple Times
  Use the `oml.index_apply` function to run a Python function multiple times in Python engines spawned by the database environment.
- Save and Manage User-Defined Python Functions in the Script Repository
  The OML4Py script repository stores user-defined Python functions for use with Embedded Python Execution functions.

14.5.1 About Python API for Embedded Python Execution

You may choose to run your functions in a data-parallel or task-parallel manner in one or more of these Python engines. In data-parallel processing, the data is partitioned and the same user-defined Python function of each data subset is invoked using one or more Python engines. In task-parallel processing, a user-defined function is invoked multiple times in one or more Python engines with a unique index passed in as an argument; for example, you may use task parallelism for Monte Carlo simulations in which you use the index to set a random seed.

The following table lists the Python functions for Embedded Python Execution.

Function	Description
`oml.do_eval`	Runs a user-defined Python function in a Python engine spawned and managed by the database environment.
`oml.group_apply`	Partitions a database table by the values in one or more columns and runs the provided user-defined Python function on each partition.
`oml.index_apply`	Runs a Python function multiple times, passing in a unique index of the invocation to the user-defined function.
`oml.row_apply`	Partitions a database table into sets of rows and runs the provided user-defined Python function on the data in each set.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Function	Description
`oml.table_apply`	Runs a Python function on data in the database as a single `pandas.DataFrame` in a single Python engine.

About Special Control Arguments

Special control arguments control what happens before or after the running of the function that you pass to an Embedded Python Execution function. You specify a special control argument with the `**kwargs` parameter of a function such as `oml.do_eval`. The control arguments are not passed to the function specified by the `func` argument of that function.

Table 14-1 Special Control Arguments

Argument	Description
`oml_input_type`	Identifies the type of input data object that you are supplying to the `func` argument. The input types are the following: • `pandas.DataFrame` • `numpy.recarray` • `'default'` (the default value) If all columns are numeric, then default type is a 2-dimensional `numpy.ndarray` of type `numpy.float64`. Otherwise, the default type is a `pandas.DataFrame`.
`oml_na_omit`	Controls the handling of missing values in the input data. If you specify `oml_na_omit = True`, then rows that contain missing values are removed from the input data. If all of the rows contain missing values, then the input data is an empty `oml.DataFrame`. The default value is `False`.

About Output

When a user-defined Python function runs in OML4Py, by default it returns the Python objects returned by the function. Also, OML4Py captures all `matplotlib.figure.Figure` objects created by the user-defined Python function and converts them into PNG format.

If `graphics = True`, the Embedded Python Execution functions return `oml.embed.data_image._DataImage` objects. The `oml.embed.data_image._DataImage` class contains Python objects and PNG images. Calling the method `__repr__()` displays the PNG images and prints out the Python object. By default, `.dat` returns the Python object that the user-defined Python function returned; `.img` returns a list containing PNG image data for each figure.

About the Script Repository

Embedded Python Execution includes the ability to create and store user-defined Python functions in the OML4Py script repository, grant or revoke the read privilege to a user-defined Python function, list the available user-defined Python functions, load user-defined Python functions into the Python environment, or drop a user-defined Python function from the script repository.

Along with whatever other actions a user-defined Python function performs, it can also create, retrieve, and modify Python objects that are stored in OML4Py datastores.

Copyright © 2024, 2026, Oracle and/or its affiliates.

In Embedded Python Execution, a user-defined Python function runs in one or more Python engines spawned and managed by the database environment. The engines are dynamically started and managed by the database. From the same user-defined Python function you can get structured data and PNG images.

You can make the user-defined Python function either private or global. A global function is available to any user. A private function is available only to the owner or to users to whom the owner of the function has granted the read privilege.

14.5.2 Run a User-Defined Python Function

Use the `oml.do_eval` function to run a user-defined input function that explicitly retrieves data or for which external data is not required.

The `oml.do_eval` function runs a user-defined Python function in a Python engine spawned and managed by the database environment.

The syntax of the `oml.do_eval` function is the following:

```
oml.do_eval(func, func_owner=None, graphics=False, **kwargs)
```

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

The `oml.do_eval` function returns a Python object or an `oml.embed.data_image._DataImage`. If no image is rendered in the user-defined Python function, `oml.do_eval` returns whatever Python object is returned by the function. Otherwise, it returns an `oml.embed.data_image._DataImage` object.

Example 14-6 Using the oml.do_eval Function

This example defines a Python function that returns a Pandas DataFrame with the columns ID and RES. It then passes that function to the `oml.do_eval` function.

```
import pandas as pd
import oml

def return_df(num, scale):
    import pandas as pd
    id = list(range(0, int(num)))
    res = [i/scale for i in id]
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
    return pd.DataFrame({"ID":id, "RES":res})

res = oml.do_eval(func=return_df, scale = 100, num = 10)
type(res)

res
```

Listing for This Example

```
>>> import pandas as pd
>>> import oml
>>>
>>> def return_df(num, scale):
...     import pandas as pd
...     id = list(range(0, int(num)))
...     res = [i/scale for i in id]
...     return pd.DataFrame({"ID":id, "RES":res})
...
>>>
>>> res = oml.do_eval(func=return_df, scale = 100, num = 10)
>>> type(res)
<class 'pandas.core.frame.DataFrame'>
>>>
>>> res
   ID  RES
0 0.0 0.00
1 1.0 0.01
2 2.0 0.02
3 3.0 0.03
4 4.0 0.04
5 5.0 0.05
6 6.0 0.06
7 7.0 0.07
8 8.0 0.08
9 9.0 0.09
```

14.5.3 Run a User-Defined Python Function on the Specified Data

Use the `oml.table_apply` function to run a Python function on data that you specify with the `data` parameter.

The `oml.table_apply` function runs a user-defined Python function in a Python engine spawned and managed by the database environment. With the `func` parameter, you can supply a Python function or you can specify the name of a user-defined Python function in the OML4Py script repository.

The syntax of the function is the following:

```
oml.table_apply(data, func, func_owner=None, graphics=False, **kwargs)
```

The `data` argument is an `oml.DataFrame` that contains the data that the `func` function operates on.

The `func` argument is the function to run. It may be one of the following:

Copyright © 2024, 2026, Oracle and/or its affiliates.

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the execution of the function.

The `oml.table_apply` function returns a Python object or an `oml.embed.data_image._DataImage`. If no image is rendered in the user-defined Python function, `oml.table_apply` returns whatever Python object is returned by the function. Otherwise, it returns an `oml.embed.data_image._DataImage` object.

Example 14-7 Using the oml.table_apply Function

This example builds a regression model using in-memory data, and then uses the `oml.table_apply` function to predict using the model on the first 10 rows of the IRIS table.

```
import oml
import pandas as pd
from sklearn import datasets
from sklearn import linear_model

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Build a regression model using in-memory data.
iris = oml_iris.pull()
regr = linear_model.LinearRegression()
regr.fit(iris[['Sepal_Width', 'Petal_Length', 'Petal_Width']],
         iris[['Sepal_Length']])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
regr.coef_

# Use oml.table_apply to predict using the model on the first 10
# rows of the IRIS table.
def predict(dat, regr):
    import pandas as pd
    pred = regr.predict(dat[['Sepal_Width', 'Petal_Length',
                             'Petal_Width']])
    return pd.concat([dat,pd.DataFrame(pred)], axis=1)

res = oml.table_apply(data=oml_iris.head(n=10),
                      func=predict, regr=regr)
res
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>> from sklearn import linear_model
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>>
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Build a regression model using in-memory data.
... iris = oml_iris.pull()
>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[['Sepal_Width', 'Petal_Length', 'Petal_Width']],
...          iris[['Sepal_Length']])
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
>>> regr.coef_
array([[ 0.65083716,  0.70913196, -0.55648266]])
>>>
>>> # Use oml.table_apply to predict using the model on the first 10
... # rows of the IRIS table.
... def predict(dat, regr):
... import pandas as pd
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
... pred = regr.predict(dat[['Sepal_Width', 'Petal_Length',
...                          'Petal_Width']])
... return pd.concat([dat,pd.DataFrame(pred)], axis=1)
...
>>> res = oml.table_apply(data=oml_iris.head(n=10),
...                       func=predict, regr=regr)
>>> res   Sepal_Length  Sepal_Width  Petal_Length  Petal_Width
0           4.6          3.6             1          0.2
1           5.1          2.5             3          1.1
2           6.0          2.2             4          1.0
3           5.8          2.6             4          1.2
4           5.5          2.3             4          1.3
5           5.5          2.5             4          1.3
6           6.1          2.8             4          1.3
7           5.7          2.5             5          2.0
8           6.0          2.2             5          1.5
9           6.3          2.5             5          1.9

      Species         0
0      setosa  4.796847
1  versicolor  4.998355
2  versicolor  5.567884
3  versicolor  5.716923
4  versicolor  5.466023
5  versicolor  5.596191
6   virginica  5.791442
7   virginica  5.915785
8   virginica  5.998775
9   virginica  5.971433
```

14.5.4 Run a Python Function on Data Grouped By Column Values

Use the `oml.group_apply` function to group the values in a database table by one or more columns and then run a user-defined Python function on each group.

The `oml.group_apply` function runs a user-defined Python function in a Python engine spawned and managed by the database environment. The `oml.group_apply` function passes the `oml.DataFrame` specified by the `data` argument to the user-defined `func` function as its first argument. The `index` argument to `oml.group_apply` specifies the columns of the `oml.DataFrame` by which the database groups the data for processing by the user-defined Python function. The `oml.group_apply` function can use data-parallel execution, in which one or more Python engines perform the same Python function on different groups of data.

The syntax of the function is the following.

```
oml.group_apply(data, index, func, func_owner=None, parallel=None,
orderby=None, graphics=False, **kwargs)
```

The `data` argument is an `oml.DataFrame` that contains the in-database data that the `func` function operates on.

The `index` argument is an `oml.DataFrame` object, the columns of which are used to group the data before sending it to the `func` function.

The `func` argument is the function to run. It may be one of the following:

Copyright © 2024, 2026, Oracle and/or its affiliates.

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or `0` for no parallelism
- `True` for the default data parallelism

The optional `orderby` argument is an `oml.DataFrame`, `oml.Float`, or `oml.String` that specifies the ordering of the group partitions.

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `False`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

The `oml.group_apply` function returns a `dict` of Python objects or a `dict` of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined Python function, `oml.group_apply` returns a `dict` of Python object returned by the function. Otherwise, it returns a `dict` of `oml.embed.data_image._DataImage` objects.

Example 14-8 Using the oml.group_apply Function

This example defines some functions and calls `oml.group_apply` for each function.

```
import pandas as pd
from sklearn import datasets
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
# Create the IRIS database table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Define a function that counts the number of rows and returns a
# dataframe with the species and the count.
def group_count(dat):
    import pandas as pd
    return pd.DataFrame([(dat["Species"][0], dat.shape[0])],\
                        columns = ["Species", "COUNT"])

# Select the Species column to use as the index argument.
index = oml.DataFrame(oml_iris['Species'])

# Group the data by the Species column and run the user-defined
# function for each species.
res = oml.group_apply(oml_iris, index, func=group_count,
                      oml_input_type="pandas.DataFrame")
res

# Define a function that builds a linear regression model, with
# Petal_Width as the feature and Petal_Length as the target value,
# and that returns the model after fitting the values.
def build_lm(dat):
    from sklearn import linear_model
    lm = linear_model.LinearRegression()
    X = dat[["Petal_Width"]]
    y = dat[["Petal_Length"]]
    lm.fit(X, y)
    return lm

# Run the model for each species and return an objectList in
# dict format with a model for each species.
mod = oml.group_apply(oml_iris[:,["Petal_Length", "Petal_Width",
                                  "Species"]], index, func=build_lm)

# The output is a dict of key-value pairs for each species and model.
type(mod)

# Sort dict by the key species.
{k: mod[k] for k in sorted(mod.keys())}
```

Listing for This Example

```
>>> import pandas as pd
>>> from sklearn import datasets
>>> import oml
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>>
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...                             {0: 'setosa', 1: 'versicolor',
...                              2:'virginica'}[x], iris.target)),
...                   columns = ['Species'])
>>>
>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table.
... oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Define a function that counts the number of rows and returns a
... # dataframe with the species and the count.
... def group_count(dat):
...     import pandas as pd
...     return pd.DataFrame([(dat["Species"][0], dat.shape[0])],\
...                          columns = ["Species", "COUNT"])
...
>>> # Select the Species column to use as the index argument.
... index = oml.DataFrame(oml_iris['Species'])
>>>
>>> # Group the data by the Species column and run the user-defined
... # function for each species.
... res = oml.group_apply(oml_iris, index, func=group_count,
...                       oml_input_type="pandas.DataFrame")
>>> res
{'setosa':   Species  COUNT
0  setosa     50, 'versicolor':       Species  COUNT
0  versicolor     50, 'virginica':      Species  COUNT
0  virginica     50}
>>>
>>> # Define a function that builds a linear regression model, with
... # Petal_Width  as the feature and Petal_Length as the target value,
... # and that returns the model after fitting the values.
... def build_lm(dat):
...     from sklearn import linear_model
...     lm = linear_model.LinearRegression()
...     X = dat[["Petal_Width"]]
...     y = dat[["Petal_Length"]]
...     lm.fit(X, y)
...     return lm
...
>>> # Run the model for each species and return an objectList in
... # dict format with a model for each species.
... mod = oml.group_apply(oml_iris[:,["Petal_Length", "Petal_Width",
...                                   "Species"]], index, func=build_lm)
>>>
>>> # The output is a dict of key-value pairs for each species and model.
... type(mod)
<class 'dict'>
>>>
>>> # Sort dict by the key species.
... {k: mod[k] for k in sorted(mod.keys())}
{'setosa': LinearRegression(copy_X=True, fit_intercept=True,
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
n_jobs=None,normalize=False), 'versicolor': LinearRegression(copy_X=True,
fit_intercept=True, n_jobs=None, normalize=False), 'virginica':
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)}
```

14.5.5 Run a User-Defined Python Function on Sets of Rows

Use the `oml.row_apply` function to chunk data into sets of rows and then run a user-defined Python function on each chunk.

The `oml.row_apply` function passes the `oml.DataFrame` specified by the `data` argument as the first argument to the user-defined `func` Python function. The `rows` argument specifies the maximum number of rows of the `oml.DataFrame` to assign to each chunk. The last chunk of rows may have fewer rows than the number specified.

The `oml.row_apply` function runs the Python function in a database-spawned Python engine. The function can use data-parallel execution, in which one or more Python engines perform the same Python function on different chunks of the data.

The syntax of the function is the following.

```
oml.row_apply(data, func, func_owner=None, rows=1, parallel=None,
graphics=False, **kwargs)
```

The `data` argument is an `oml.DataFrame` that contains the data that the `func` function operates on.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `rows` argument is an `int` that specifies the maximum number of rows to include in each chunk.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or `0` for no parallelism
- `True` for the default data parallelism

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `True`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The `oml.row_apply` function returns a `pandas.DataFrame` or a list of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined Python function, `oml.row_apply` returns a `pandas.DataFrame`. Otherwise, it returns a list of `oml.embed.data_image._DataImage` objects.

Example 14-9 Using the oml.row_apply Function

This example creates the `x` and `y` variables using the iris data set. It then creates the persistent database table IRIS and the `oml.DataFrame` object `oml_iris` as a proxy for the table.

The example builds a regression model based on iris data. It defines a function that predicts the Petal_Width values based on the Sepal_Length, Sepal_Width, and Petal_Length columns of the input data. It then concatenates the Species column, the Petal_Width column, and the predicted Petal_Width as the object to return. Finally, the example calls the `oml.row_apply` function to apply the `make_pred()` function on each 4-row chunk of the input data.

```
import oml
import pandas as pd
from sklearn import datasets
from sklearn import linear_model

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()
x = pd.DataFrame(iris.data,
                 columns = ['Sepal_Length','Sepal_Width',
                            'Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor',
                           2:'virginica'}[x], iris.target)),
                 columns = ['Species'])

# Drop the IRIS database table if it exists.
try:
    oml.drop('IRIS')
except:
    pass

# Create the IRIS database table and the proxy object for the table.
oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')

# Build a regression model to predict Petal_Width using in-memory
# data.
iris = oml_iris.pull()
regr = linear_model.LinearRegression()
regr.fit(iris[['Sepal_Length', 'Sepal_Width', 'Petal_Length']],
         iris[['Petal_Width']])
regr.coef_

# Define a Python function.
def make_pred(dat, regr):
    import pandas as pd
    import numpy as np
    pred = regr.predict(dat[['Sepal_Length',
                             'Sepal_Width',
                             'Petal_Length']])
    return pd.concat([dat[['Species', 'Petal_Width']],
                      pd.DataFrame(pred,
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                                    columns=['Pred_Petal_Width'])],
                                    axis=1)

input_data = oml_iris.split(ratio=(0.9, 0.1), strata_cols='Species')[1]
input_data.crosstab(index = 'Species').sort_values('Species')

res = oml.row_apply(input_data, rows=4, func=make_pred,
                    regr=regr, parallel=2)
type(res)
res
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>> from sklearn import datasets
>>> from sklearn import linear_model
>>>
>>> # Load the iris data set and create a pandas.DataFrame for it.
... iris = datasets.load_iris()
>>> x = pd.DataFrame(iris.data,
...                  columns = ['Sepal_Length','Sepal_Width',
...                             'Petal_Length','Petal_Width'])
>>> y = pd.DataFrame(list(map(lambda x:
...                           {0: 'setosa', 1: 'versicolor',
...                            2:'virginica'}[x], iris.target)),
...                  columns = ['Species'])
>>>
>>> # Drop the IRIS database table if it exists.
... try:
...     oml.drop('IRIS')
... except:
...     pass
>>>
>>> # Create the IRIS database table and the proxy object for the table.
>>> oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
>>>
>>> # Build a regression model to predict Petal_Width using in-memory
... # data.
... iris = oml_iris.pull()
>>> regr = linear_model.LinearRegression()
>>> regr.fit(iris[['Sepal_Length', 'Sepal_Width', 'Petal_Length']],
...          iris[['Petal_Width']])
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
>>> regr.coef_
array([[-0.20726607,  0.22282854,  0.52408311]])
>>>
>>> # Define a Python function.
... def make_pred(dat, regr):
...     import pandas as pd
...     import numpy as np
...     pred = regr.predict(dat[['Sepal_Length',
...                              'Sepal_Width',
...                              'Petal_Length']])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...     return pd.concat([dat[['Species', 'Petal_Width']],
...                       pd.DataFrame(pred,
...                                    columns=['Pred_Petal_Width'])],
...                                    axis=1)
>>>
>>> input_data = oml_iris.split(ratio=(0.9, 0.1), strata_cols='Species')[1]
>>> input_data.crosstab(index = 'Species').sort_values('Species')
      SPECIES  count
0      setosa      7
1  versicolor      8
2   virginica      4
>>>  res = oml.row_apply(input_data, rows=4, func=make_pred, regr=regr,
...                      columns=['Species',
...                               'Petal_Width',
...                               'Pred_Petal_Width']))
>>> res = oml.row_apply(input_data, rows=4, func=make_pred,
...                      regr=regr, parallel=2)
>>> type(res)
<class 'pandas.core.frame.DataFrame'>
>>> res
       Species  Petal_Width  Pred_Petal_Width
0       setosa          0.4          0.344846
1       setosa          0.3          0.335509
2       setosa          0.2          0.294117
3       setosa          0.2          0.220982
4       setosa          0.2          0.080937
5   versicolor          1.5          1.504615
6   versicolor          1.3          1.560570
7   versicolor          1.0          1.008352
8   versicolor          1.3          1.131905
9   versicolor          1.3          1.215622
10  versicolor          1.3          1.272388
11   virginica          1.8          1.623561
12   virginica          1.8          1.878132
```

14.5.6 Run a User-Defined Python Function Multiple Times

Use the `oml.index_apply` function to run a Python function multiple times in Python engines spawned by the database environment.

The syntax of the function is the following:

```
oml.index_apply(times, func, func_owner=None, parallel=None, graphics=False,
**kwargs)
```

The `times` argument is an `int` that specifies the number of times to run the `func` function.

The `func` argument is the function to run. It may be one of the following:

- A Python function
- A string that is the name of a user-defined Python function in the OML4Py script repository
- A string that defines a Python function
- An `oml.script.script.Callable` object returned by the `oml.script.load` function

Copyright © 2024, 2026, Oracle and/or its affiliates.

The optional `func_owner` argument is a string or `None` (the default) that specifies the owner of the registered user-defined Python function when argument `func` is a registered user-defined Python function name.

The `parallel` argument is a boolean, an `int`, or `None` (the default) that specifies the preferred degree of parallelism to use in the Embedded Python Execution job. The value may be one of the following:

- A positive integer greater than or equal to 1 for a specific degree of parallelism
- `False`, `None`, or `0` for no parallelism
- `True` for the default data parallelism

The `graphics` argument is a boolean that specifies whether to look for images. The default value is `True`.

With the `**kwargs` parameter, you can pass additional arguments to the `func` function. Special control arguments, which start with `oml_`, are not passed to the function specified by `func`, but instead control what happens before or after the running of the function.

The `oml.index_apply` function returns a list of Python objects or a list of `oml.embed.data_image._DataImage` objects. If no image is rendered in the user-defined Python function, `oml.index_apply` returns a list of the Python objects returned by the user-defined Python function. Otherwise, it returns a list of `oml.embed.data_image._DataImage` objects.

Example 14-10 Using the oml.index_apply Function

This example defines a function that returns the mean of a set of random numbers the specified number of times.

```
import oml
import pandas as pd

def compute_random_mean(index):
    import numpy as np
    import scipy
    from statistics import mean
    np.random.seed(index)
    res = np.random.random((100,1))*10
    return mean(res[1])
res = oml.index_apply(times=10, func=compute_random_mean)
type(res)
res
```

Listing for This Example

```
>>> import oml
>>> import pandas as pd
>>>
>>> def compute_random_mean(index):
...     import numpy as np
...     import scipy
...     from statistics import mean
...     np.random.seed(index)
...     res = np.random.random((100,1))*10
...     return mean(res[1])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
...
>>> res = oml.index_apply(times=10, func=compute_random_mean)
>>> type(res)
<class 'list'>
>>> res
[7.203244934421581, 0.25926231827891333, 7.081478226181048,
5.4723224917572235, 8.707323061773764, 3.3197980530117723,
7.7991879224011464, 9.68540662820932, 5.018745921487388,
0.207519493594015]
```

14.5.7 Save and Manage User-Defined Python Functions in the Script Repository

The OML4Py script repository stores user-defined Python functions for use with Embedded Python Execution functions.

> **Note**
>
> The user-defined Python functions can be used outside of Embedded Python Execution. You can store functions and reload them back into notebooks or other user-defined functions.

The script repository is a component of the Embedded Python Execution functionality.

The following topics describe the script repository and the Python functions for managing user-defined Python functions:

Topics:

- About the Script Repository
  Use these functions to store, manage, and use user-defined Python functions in the script repository.
- Create and Store a User-Defined Python Function
  Use the `oml.script.create` function to add a user-defined Python function to the script repository.
- List Available User-Defined Python Functions
  Use the `oml.script.dir` function to list the user-defined Python functions in the OML4Py script repository.
- Load a User-Defined Python Function
  Use the `oml.script.load` function to load a user-defined Python function from the script repository into a Python session.
- Drop a User-Defined Python Function from the Repository
  Use the `oml.script.drop` function to remove a user-defined Python function from the script repository.
- Export a User-Defined Python Function
  Use the `oml.export_script` function to export a user-defined Python functions from the OML4Py script repository to a destination file.
- Import a User-Defined Python Function
  Use the `oml.import_script` function to import a user-defined Python functions to the OML4Py script repository from a specified file.

Copyright © 2024, 2026, Oracle and/or its affiliates.

14.5.7.1 About the Script Repository

Use these functions to store, manage, and use user-defined Python functions in the script repository.

The following table lists the Python functions for the script repository.

Function	Description
`oml.script.create`	Registers a single user-defined Python function in the script repository.
`oml.script.dir`	Lists the user-defined Python functions present in the script repository.
`oml.script.drop`	Drops a user-defined Python function from the script repository.
`oml.script.load`	Loads a user-defined Python function from the script repository into a Python session.

The following table lists the Python functions for managing access to user-defined Python functions in the script repository, and to datastores and datastore objects.

Function	Description
`oml.grant`	Grants read privilege permission to another user to a datastore or user-defined Python function owned by the current user.
`oml.revoke`	Revokes the read privilege permission that was granted to another user to a datastore or user-defined Python function owned by the current user.

14.5.7.2 Create and Store a User-Defined Python Function

Use the `oml.script.create` function to add a user-defined Python function to the script repository.

With the `oml.script.create` function, you can store a single user-defined Python function in the OML4Py script repository. You can then specify the user-defined Python function as the `func` argument to the Embedded Python Execution functions `oml.do_eval`, `oml.group_apply`, `oml.index_apply`, `oml.row_apply`, and `oml.table_apply`.

You can make the user-defined Python function either private or global. A private user-defined Python function is available only to the owner, unless the owner grants the read privilege to other users. A global user-defined Python function is available to any user.

The syntax of `oml.script.create` is the following:

```
oml.script.create(name, func, is_global=False, overwrite=False)
```

The `name` argument is a string that specifies a name for the user-defined Python function in the Python script repository.

The `func` argument is the Python function to run. The argument can be a Python function or a string that contains the definition of a Python function. You must specify a string in an interactive session if `readline` cannot get the command history.

The `is_global` argument is a boolean that specifies whether to create a global user-defined Python function. The default value is `False`, which indicates that the user-defined Python function is a private function available only to the current session user. When `is_global` is

Copyright © 2024, 2026, Oracle and/or its affiliates.

`True`, it specifies that the function is global and every user has the read privilege and the execute privilege to it.

The `overwrite` argument is a boolean that specifies whether to overwrite the user-defined Python function if it already exists. The default value is `False`.

Example 14-11 Using the oml.script.create Function

This example stores two user-defined Python functions in the script repository. It then lists the contents of the script repository using different arguments to the `oml.script.dir` function.

Load the iris dataset as a pandas dataframe from the seaborn library. Use the `oml.create` function to create the IRIS database table and the proxy object for the table.

```
%python

from sklearn import datasets
import pandas as pd
import oml

# Load the iris data set and create a pandas.DataFrame for it.
iris = datasets.load_iris()

# Create objects containing data for the user-defined functions to use.
x = pd.DataFrame(iris.data,
                 columns =
['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width'])
y = pd.DataFrame(list(map(lambda x:
                          {0: 'setosa', 1: 'versicolor', 2:'virginica'}[x],
iris.target)),
                 columns = ['Species'])

# Create the IRIS database table and the proxy object for the table.

try:
  oml.drop(table="IRIS")
except:
  pass

oml_iris = oml.create(pd.concat([x, y], axis=1), table = 'IRIS')
```

Create an user-defined function `build_lm1` and use `oml.script.create` function to store it in the OML4Py script repository. The parameter `"build_lm1"` is a string that specifies the name of the user-defined function. The parameter `func=build_lm1` is the Python function to run. Run the user-defined Python function in embedded Python execution.

```
%python

# Define a function.

build_lm1 = '''def build_lm1(dat):
  from sklearn import linear_model
  regr = linear_model.LinearRegression()
  import pandas as pd
  dat = pd.get_dummies(dat, drop_first=True)
  X = dat[["Sepal_Width", "Petal_Length", "Petal_Width",
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
"Species_versicolor", "Species_virginica"]]
  y = dat[["Sepal_Length"]]
  regr.fit(X, y)
  return regr'''

# Create a private user-defined Python function.
oml.script.create("build_lm1", func=build_lm1, overwrite=True)

# Run the user-defined Python function in embedded Python execution
res = oml.table_apply(oml_iris, func="build_lm1",
oml_input_type="pandas.DataFrame")

res
res.coef_
```

The output is the following:

```
array([[ 0.49588894,  0.82924391, -0.31515517, -0.72356196, -1.02349781]])
```

Define another user-defined function `build_lm2`, store the function as a global script in the OML4Py script repository. Run the user-defined Python function in embedded Python execution.

```
%python

# Define another function

build_lm2 = '''def build_lm2(dat):
  from sklearn import linear_model
  regr = linear_model.LinearRegression()
  X = dat[["Petal_Width"]]
  y = dat[["Petal_Length"]]
  regr.fit(X, y)
  return regr'''

# Save the function as a global script to the script repository, overwriting
any existing function with the same name.
oml.script.create("build_lm2", func=build_lm2, is_global=True,
overwrite=True)

res = oml.table_apply(oml_iris, func="build_lm2",
oml_input_type="pandas.DataFrame")
res
```

The output is the following:

```
LinearRegression()
```

List the user-defined Python functions in the script repository available to the current user only.

```
%python

oml.script.dir()
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

The output is similar to the following:

```
                name  ...                date
0         build_lm1  ... 2022-12-15 19:02:44
1         build_mod  ... 2022-12-12 23:02:31
2     myFitMultiple  ... 2022-12-14 22:30:43
3  sample_iris_table  ... 2022-12-14 22:21:24

[4 rows x 4 columns]
```

List all of the user-defined Python functions available to the current user.

```
%python

oml.script.dir(sctype='all')
```

The output is similar to the following:

```
     owner  ...                date
0   PYQSYS  ... 2022-02-11 06:06:44
1   PYQSYS  ... 2022-10-19 16:59:50
2   PYQSYS  ... 2022-10-19 16:59:52
3   PYQSYS  ... 2022-10-19 16:59:53
```

List the user-defined Python functions available to all users.

```
%python

oml.script.dir(sctype='global')
```

The output is similar to the following:

```
                  name  ...                date
0                GLBLM  ... 2022-02-11 06:06:44
1         RandomRedDots  ... 2022-10-19 16:59:50
2        RandomRedDots2  ... 2022-10-19 16:59:52
3        RandomRedDots3  ... 2022-10-19 16:59:53
4                  TEST  ... 2021-08-13 17:37:02
5                 TEST4  ... 2021-08-13 17:42:49
6              TEST_FUN  ... 2021-08-13 22:38:54
```

14.5.7.3 List Available User-Defined Python Functions

Use the `oml.script.dir` function to list the user-defined Python functions in the OML4Py script repository.

The syntax of the `oml.script.dir` function is the following:

```
oml.script.dir(name=None, regex_match=False, sctype='user')
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

The `name` argument is a string that specifies the name of a user-defined Python function or a regular expression to match to the names of user-defined Python functions in the script repository. When `name` is `None`, this function returns the type of user-defined Python functions specified by argument `sctype`.

The `regex_match` argument is a boolean that indicates whether argument `name` is a regular expression to match. The default value is `False`.

The `sctype` argument is a string that specifies the type of user-defined Python function to list. The value may be one of the following.

- `user`, to specify the user-defined Python functions available to the current user only.
- `grant`, to specify the user-defined Python functions to which the read and execute privilege have been granted by the current user to other users.
- `granted`, to specify the user-defined Python functions to which the read and execute privilege have been granted by other users to the current user.
- `global`, to specify all of the global user-defined Python functions created by the current user.
- `all`, to specify all of the user-defined Python functions available to the current user.

The `oml.script.dir` function returns a `pandas.DataFrame` that contains the columns NAME and SCRIPT and, optionally, the columns OWNER and GRANTEE.

Example 14-12 Using the oml.script.dir Function

This example lists the contents of the script repository using different arguments to the `oml.script.dir` function. For the creation of the user-defined Python functions, see Example 14-11.

```
import oml

# List the user-defined Python functions in the script
# repository available to the current user only.
oml.script.dir()

# List all of the user-defined Python functions available
# to the current user.
oml.script.dir(sctype='all')

# List the user-defined Python functions available to all users.
oml.script.dir(sctype='global')

# List the user-defined Python functions that contain the letters
# BL and that are available to all users.
oml.script.dir(name="BL", regex_match=True, sctype='all')
```

Listing for This Example

```
>>> import oml
>>>
>>> # List the user-defined Python functions in the script
... # repository available to the current user only.
... oml.script.dir()
   NAME                                              SCRIPT
0  MYLM  def build_lm1(dat):\n    from sklearn import l...
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>>
>>> # List all of the user-defined Python functions available
... to the current user.
... oml.script.dir(sctype='all')
     OWNER    NAME                                                SCRIPT
0    PYQSYS  GLBLM  def build_lm2(dat):\n    from sklearn import l...
1  OML_USER   MYLM  def build_lm1(dat):\n    from sklearn import l...
>>>
>>> # List the user-defined Python functions available to all users.
>>> oml.script.dir(sctype='global')
    NAME                                              SCRIPT
0  GLBLM  def build_lm2(dat):\n    from sklearn import l...
>>>
>>> # List the user-defined Python functions that contain the letters
... # BL and that are available to all users.
... oml.script.dir(name="BL", regex_match=True, sctype='all')
    OWNER   NAME                                               SCRIPT
0  PYQSYS  GLBLM  def build_lm2(dat):\n    from sklearn import l...
```

14.5.7.4 Load a User-Defined Python Function

Use the `oml.script.load` function to load a user-defined Python function from the script repository into a Python session.

The syntax of the function is the following:

```
oml.script.load(name, owner=None)
```

The `name` argument is a string that specifies the name of the user-defined Python function to load from the OML4Py script repository.

The optional `owner` argument is a string that specifies the owner of the user-defined Python function or `None` (the default). If `owner=None`, then this function finds and loads the user-defined Python function that matches `name` in the following order:

1. A user-defined Python function that the current user created.
2. A global user-defined Python function that was created by another user.

The `oml.script.load` function returns an `oml.script.script.Callable` object that references the named user-defined Python function.

Example 14-13 Using the oml.script.load Function

This example loads user-defined Python functions from the script repository and pulls them to the local Python session. For the creation of the user-defined Python functions, see Example 14-11.

```
import oml

# Load the MYLM and GLBLM user-defined Python functions.
MYLM = oml.script.load(name="MYLM")
GMYLM = oml.script.load(name="GLBLM")

# Pull the models to the local Python session.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
MYLM(oml_iris.pull()).coef_
GMYLM(oml_iris.pull())
```

Listing for This Example

```
>>> import oml
>>>
>>> # Load the MYLM and GLBLM user-defined Python functions.
>>> MYLM = oml.script.load(name="MYLM")
>>> GMYLM = oml.script.load(name="GLBLM")
>>>
>>> # Pull the models to the local Python session.
... MYLM(oml_iris.pull()).coef_
array([[ 0.49588894, 0.82924391, -0.31515517, -0.72356196, -1.02349781]])
>>> GMYLM(oml_iris.pull())
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
         normalize=False)
```

14.5.7.5 Drop a User-Defined Python Function from the Repository

Use the `oml.script.drop` function to remove a user-defined Python function from the script repository.

The `oml.script.drop` function drops a user-defined Python function from the OML4Py script repository.

The syntax of the function is the following:

```
oml.script.drop(name, is_global=False, silent=False)
```

The `name` argument is a string that specifies the name of the user-defined Python function in the script repository.

The `is_global` argument is a boolean that specifies whether the user-defined Python function to drop is a global or a private user-defined Python function. The default value is `False`, which indicates a private user-defined Python function.

The `silent` argument is a boolean that specifies whether to display an error message when `oml.script.drop` encounters an error in dropping the specified user-defined Python function. The default value is `False`.

Example 14-14 Using the oml.script.drop Function

This example drops user-defined Python functions the MYLM private user-defined Python function and the GLBLM global user-defined Python function from the script repository. For the creation of the user-defined Python functions, see Example 14-11.

```
import oml

# List the available user-defined Python functions.
oml.script.dir(sctype="all")

# Drop the private user-defined Python function.
oml.script.drop("MYLM")

# Drop the global user-defined Python function.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
oml.script.drop("GLBLM", is_global=True)

# List the available user-defined Python functions again.
oml.script.dir(sctype="all")
```

Listing for This Example

```
>>> import oml
>>>
>>> # List the available user-defined Python functions.
... oml.script.dir(sctype="all")
     OWNER   NAME                                                SCRIPT
0   PYQSYS  GLBLM  def build_lm2(dat):\n  from sklearn import lin...
1 OML_USER   MYLM  def build_lm1(dat):\n  from sklearn import lin...
>>>
>>> # Drop the private user-defined Python function.
... oml.script.drop("MYLM")
>>>
>>> # Drop the global user-defined Python function.
... oml.script.drop("GLBLM", is_global=True)
>>>
>>> # List the available user-defined Python functions again.
... oml.script.dir(sctype="all")
Empty DataFrame
Columns: [OWNER, NAME, SCRIPT]
Index: []
```

14.5.7.6 Export a User-Defined Python Function

Use the `oml.export_script` function to export a user-defined Python functions from the OML4Py script repository to a destination file.

The syntax of the function is the following:

```
oml.export_script(file_name, name=None, regex_match=False, sctype='user',
overwrite=False)
```

Function Parameter:

- file_name (type: string): Specifies either the file name or the path of the destination file containing the exported scripts, and both must end with either `.json` or `.py`. Only two formats are supported: `.json` and `.py`. The file type is determined by the suffix. For example, consider the following two cases regarding the destination file path:
  - `/dir1/file1.json`: This is a valid path , and the output will be saved as `file1.json` at `/dir1/file1.json`.
  - `/dir1`: This is an invalid path and will result in an error.

  Similarly, for `.py` format.
- name (type: string, list of string or None (default)): Specifies the script name or a regular expression to match script names. If name is `None` or empty list, it exports all the scripts based on the specified argument `sctype`.
- regex_match (type: bool or False (default)): Indicates whether the argument `name` is a regular expression or not. If multiple user-defined functions match the regex, only one file

Copyright © 2024, 2026, Oracle and/or its affiliates.

will be generated. All matched user-defined functions will be included in this single file, which is named according to the file_name parameter.

- sctype (type: string): Specifies the type of user-defined Python function to export. It can take one of the following values:
  - `user` (default): Exports user-defined Python function created by the current session user.
  - `grant`: Exports user-defined Python functions to which the current user has granted read and execute privileges to the other users.
  - `granted`: Exports user-defined Python functions to which other users have granted read and execute privileges to the current user.
  - `global`: Exports all of the global user-defined Python functions created by the current user.
  - `all`: Exports all user-defined Python functions that are available to the current user, for which the current user has read access.
- overwrite (type: bool or False (default)): If set to `True`, it will overwrite the `file_name` if it already exists.

Example 14-15 Using the oml.export_script Function

This example uses the `oml.export_script` function to export the user-defined Python function in both `.json` and `.py` file formats. In the example `oml.export_script(file_name="script_user.json", sctype="user")`, since the argument name is None and sctype is "user", the function will match all user-defined functions created by the current session user.

For the creation of the user-defined Python functions, see Example 14-11.

```
import oml

#List the user-defined Python functions in the script
# repository available to the current user only.
oml.script.dir(name="LM", regex_match=True, sctype="all")[['owner', 'name',
'script', 'description']]

#Export user-owned and global scripts to the files named
#script_user.json and script_global.py respectively.
oml.export_script(file_name="script_user.json", sctype="user")
oml.export_script(file_name="script_global.py", sctype="global")

#Drop the scripts named MYLM and GLBLM.
oml.script.drop("MYLM")
oml.script.drop("GLBLM", is_global=True)

# List all of the user-defined Python functions available to the current user.
oml.script.dir(sctype="all")

#Verifies whether the specified set of files is present in the list of files
located in a directory.
import os
{"script_user.json", "script_global.py"}.issubset(os.listdir(path="./"))
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Listing for This Example

```
>>> oml.script.create("MYLM", func=build_lm1, description="my lm model build")
>>> oml.script.create("GLBLM", func=build_lm2, is_global=True)
>>> oml.script.dir(name="LM", regex_match=True, sctype="all")[['owner',
'name', 'script', 'description']]
    owner  name                                              script \
0  PYQSYS GLBLM def build_lm2(dat):\n from sklearn import lin...
1 PYQUSER  MYLM def build_lm1(dat):\n from sklearn import lin...
<BLANKLINE>
        description
0              None
1 my lm model build
>>> oml.export_script(file_name="script_user.json", sctype="user")
>>> oml.export_script(file_name="script_global.py", sctype="global")
>>> oml.script.drop("MYLM")
>>> oml.script.drop("GLBLM", is_global=True)
>>> oml.script.dir(sctype="all")
Empty DataFrame
Columns: [owner, name, script, description, date]
Index: []
>>> import os
>>> {"script_user.json", "script_global.py"}.issubset(os.listdir(path="./"))
True
```

14.5.7.7 Import a User-Defined Python Function

Use the `oml.import_script` function to import a user-defined Python functions to the OML4Py script repository from a specified file.

The syntax of the function is the following:

```
oml.import_script(file_name, is_global=False, overwrite=False)
```

Function Parameter:

- file_name (type: string): Specifies the file containing the scripts to import. Only two formats are supported: `.json` and `.py`. The file type is determined by the suffix.
- is_global (type: bool or False (default)): Indicates whether the user-defined Python function is globally accessible after imported. By default, it is set to `False`, meaning it is only accessible to you.
- overwrite (type: bool or False (default)): If set to `True`, overwrite the existing script if it already exists.

Example 14-16 Using the oml.import_script Function

This example uses the `oml.import_script` function to import the user-defined Python function in both `.json` and `.py` file formats from a specified file. For the creation of the user-defined Python functions, see Example 14-11.

```
import oml

# Import the file named script_user.json as a private user-defined Python
function.
oml.import_script(file_name="script_user.json")
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
oml.script.dir()[['name', 'script', 'description']]

# Import the file name script_global.py as a global user-defined Python
function.
oml.import_script(file_name="script_global.py", is_global=True)
oml.script.dir(name="LM", regex_match=True, sctype="all")[['owner', 'name',
'script','description']]

# Drop the private user-defined Python function.
oml.script.drop("MYLM")

# Drop the global user-defined Python function.
oml.script.drop("GLBLM", is_global=True)
```

Listing for This Example

```
oml.import_script(file_name="script_user.json")
>>> oml.script.dir()[['name', 'script', 'description']]
  name                                                script       description
0 MYLM def build_lm1(dat):\n from sklearn import lin... my lm model build
>>> oml.import_script(file_name="script_global.py", is_global=True)
>>> oml.script.dir(name="LM", regex_match=True, sctype="all")[['owner',
'name', 'script', 'description']]
    owner  name                                                script \
0  PYQSYS GLBLM def build_lm2(dat):\n from sklearn import lin...
1 PYQUSER  MYLM def build_lm1(dat):\n from sklearn import lin...
<BLANKLINE>
        description
0              None
1 my lm model build

>>> oml.script.drop("MYLM")
>>> oml.script.drop("GLBLM", is_global=True)
```

14.6 SQL API for Embedded Python Execution with On-premises AI Database

SQL API for Embedded Python Execution with On-premises AI Database has SQL interfaces for Embedded Python Execution and for datastore and script repository management.

The following topics describe the OML4Py SQL interfaces for Embedded Python Execution.

Topics:

- About the SQL API for Embedded Python Execution with On-Premises Database
  With the SQL API, you can run user-defined Python functions in one or more separate Python engines in an Oracle AI Database environment, manage user-defined Python functions in the OML4Py script repository, and control access to and get information about datastores and about user-defined Python functions in the script repository.
- pyqEval Function (On-Premises Database)
  This topic describes the `pyqEval` function when used in an on-premises Oracle AI Database. The `pyqEval` function runs a user-defined Python function that explicitly retrieves data or for which external data is to be automatically loaded for the function.

Copyright © 2024, 2026, Oracle and/or its affiliates.

- pyqTableEval Function (On-Premises Database)
  This topic describes the `pyqTableEval` function when used in an on-premises Oracle AI Database. The `pyqTableEval` function runs a user-defined Python function on data from an Oracle AI Database table.
- pyqRowEval Function (On-Premises Database)
  This topic describes the `pyqRowEval` function when used in an on-premises Oracle AI Database. The `pyqRowEval` function chunks data into sets of rows and then runs a user-defined Python function on each chunk.
- pyqGroupEval Function (On-Premises Database)
  This topic describes the `pyqGroupEval` function when used in an on-premises Oracle AI Database. The `pyqGroupEval` function groups data by one or more columns and runs a user-defined Python function on each group.
- pyqGrant Function (On-Premises Database)
  This topic describes the `pyqGrant` function when used in an on-premises Oracle AI Database.
- pyqRevoke Function (On-Premises Database)
  This topic describes the `pyqRevoke` function when used in an on-premises Oracle AI Database.
- pyqScriptCreate Procedure (On-Premises Database)
  This topic describes the `pyqScriptCreate` procedure in an on-premises Oracle AI Database. The `pyqScriptCreate` procedure creates a user-defined Python function and adds it to the OML4Py script repository.
- pyqScriptDrop Procedure (On-Premises Database)
  This topic describes the `pyqScriptDrop` procedure in an on-premises Oracle AI Database. The `pyqScriptDrop` procedure removes a user-defined Python function from the OML4Py script repository.

14.6.1 About the SQL API for Embedded Python Execution with On-Premises Database

With the SQL API, you can run user-defined Python functions in one or more separate Python engines in an Oracle AI Database environment, manage user-defined Python functions in the OML4Py script repository, and control access to and get information about datastores and about user-defined Python functions in the script repository.

You can use the SQL interface for Embedded Python Execution with an on-premises Oracle Database instance.

OML4Py provides the following types of SQL functions and procedures.

- SQL table functions for running user-defined Python functions in one or more database-spawned and managed Python engines; the user-defined Python functions may reference Python objects in OML4Py datastores and use third-party packages installed with the database server machine Python engines..
- PL/SQL procedures for creating and dropping user-defined Python functions in the OML4Py script repository.
- PL/SQL procedures for granting and revoking the read privilege to datastores and the datastore objects in them, and to user-defined Python functions in the OML4Py script repository.

The following table lists the SQL functions for Embedded Python Execution and the PL/SQL procedures for managing datastores and user-defined Python functions.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Function or Procedure	Description
`pyqEval` function	Runs a user-defined Python function on the data passed in.
`pyqGroupEval` function	Groups data by one or more columns and runs a user-defined Python function on each group.
`pyqTableEval` function	Runs a user-defined Python function on data in the database.
`pyqRowEval` function	Runs the specified number of rows in each invocation of the user-defined Python function in parallel processes.
`pyqGrant` procedure	Grants the read privilege to another user to a user-defined Python function owned by the current user.
`pyqRevoke` procedure	Revokes the read privilege that was granted to another user to a user-defined Python function owned by the current user.
`pyqScriptCreate` procedure	Creates a user-defined Python function in the script repository.
`pyqScriptDrop` procedure	Drops a user-defined Python function from the script repository.

14.6.2 pyqEval Function (On-Premises Database)

This topic describes the `pyqEval` function when used in an on-premises Oracle AI Database. The `pyqEval` function runs a user-defined Python function that explicitly retrieves data or for which external data is to be automatically loaded for the function.

You can pass arguments to the Python function with the `PAR_LST` parameter.

The `pyqEval` function does not automatically receive any data from the database. The Python function generates the data that it uses or it explicitly retrieves it from a data source such as Oracle AI Database, other databases, or flat files.

The Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
pyqEval(
    par_lst VARCHAR2,
    out_fmt VARCHAR2,
    scr_name VARCHAR2,
    scr_owner VARCHAR2 DEFAULT NULL)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameters

Parameter	Description
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}'
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Returns

Function pyqEval returns a table that has the structure specified by the OUT_FMT parameter value.

Example 14-17 Using the pyqEval Function

This example defines Python functions and stores them in the OML4Py script repository. It invokes the pyqEval function on the user-defined Python functions.

In a PL/SQL block, create an unnamed Python function that is stored in script repository with the name pyqFun1.

```
BEGIN
   sys.pyqScriptCreate('pyqFun1', 'func = lambda: "Hello World from a
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
lambda!"',
                          FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Invoke the `pyqEval` function, which runs the user-defined Python function and returns the results as XML.

```
SELECT name, value
  FROM table(pyqEval(
             NULL,
             'XML',
             'pyqFun1'));
```

The output is the following.

```
NAME  VALUE
----  --------------------------------------------------
      <root><str>Hello World from a lambda!</str></root>
```

Drop the user-defined Python function.

```
BEGIN
  sys.pyqScriptDrop('pyqFun1');
END;
/
```

Define a Python function that returns a `numpy.ndarray` that is stored in script repository with the name pyqFun2.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
       import numpy as np
       import pickle
       z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
                      [float(x)/10 for x in range(10)],
                      [x for x in range(10)],
                      [bool(x%2) for x in range(10)],
                      [pickle.dumps(x) for x in range(10)],
                      ["test"+str(x**2) for x in range(10)])],
                    dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"),
                           ("d", "?"), ("e", "S20"), ("f", "O")])
       return z');
END;
/
```

Invoke the `pyqEval` function, which runs the pyqFun2 user-defined Python function.

```
SELECT *
  FROM table(pyqEval(
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                NULL,
                '{"A":"varchar2(10)", "B":"number",
                  "C":"number", "D":"number",
                  "E":"raw(10)", "F": "varchar2(10)" }',
                 'pyqFun2'));
```

The output is the following.

```
A                    B          C          D E                    F
---------- ---------- ---------- ---------- -------------------- ----------
0demo                0          0          0 80034B002E           test0
1demo         1.0E-001          1          1 80034B012E           test1
2demo         2.0E-001          2          0 80034B022E           test4
3demo         3.0E-001          3          1 80034B032E           test9
4demo         4.0E-001          4          0 80034B042E           test16
5demo         5.0E-001          5          1 80034B052E           test25
6demo         6.0E-001          6          0 80034B062E           test36
7demo         7.0E-001          7          1 80034B072E           test49
8demo         8.0E-001          8          0 80034B082E           test64
9demo         9.0E-001          9          1 80034B092E           test81

10 rows selected.
```

Drop the user-defined Python function.

```
BEGIN
  sys.pyqScriptDrop('pyqFun2');
END;
/
```

14.6.3 pyqTableEval Function (On-Premises Database)

This topic describes the `pyqTableEval` function when used in an on-premises Oracle AI Database. The `pyqTableEval` function runs a user-defined Python function on data from an Oracle AI Database table.

You pass data to the Python function with the `INP_NAM` parameter. You can pass arguments to the Python function with the `PAR_LST` parameter.

The Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
pyqTableEval(
    inp_nam VARCHAR2,
    par_lst VARCHAR2,
    out_fmt VARCHAR2,
    scr_name VARCHAR2,
    scr_owner VARCHAR2 DEFAULT NULL)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}'
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Returns

Function pyqTableEval returns a table that has the structure specified by the OUT_FMT parameter value.

Example 14-18 Using the pyqTableEval Function

This example stores a user-defined Python function in the OML4Py script repository with the name create_iris_table. It uses the function to create a database table as the result of a

Copyright © 2024, 2026, Oracle and/or its affiliates.

pyqEval function invocation. It creates another user-defined Python function that fits a linear regression model to the input data and saves the model in the OML4Py datastore. The example runs a SQL SELECT statement that invokes the pyqTableEval function, which invokes the function stored in the script repository with the name myLinearRegressionModel.

In a PL/SQL block, define the Python function create_iris_table and store in the script repository with the name create_iris_table, overwriting any existing user-defined Python function stored in the script repository with the same name.

The create_iris_table function imports and loads the iris data set, creates two pandas.DataFrame objects, and then returns the concatenation of those objects.

```
BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
       from sklearn.datasets import load_iris
       import pandas as pd
       iris = load_iris()
       x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
             "Sepal_Width", "Petal_Length", "Petal_Width"])
       y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                                  2: "virginica"}[x], iris.target)),\
                        columns = ["Species"])
       return pd.concat([y, x], axis=1)',
    FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
CREATE TABLE IRIS AS
(SELECT * FROM pyqEval(
      NULL,
      '{"Species":"VARCHAR2(10)","Sepal_Length":"number",
        "Sepal_Width":"number","Petal_Length":"number",
        "Petal_Width":"number"}',
      'create_iris_table'
));
```

Define the Python function fit_model and store it with the name myLinearRegressionModel as a private function in the script repository, overwriting any existing user-defined Python function stored with that name.

The fit_model function fits a regression model to the input data dat and then saves the fitted model as an object specified by the modelName argument to the datastore specified by the datastoreName argument. The fit_model function returns the fitted model in a string format.

By default, Python objects are saved to a new datastore with the specified datastoreName. To save an object to an existing datastore, either set the overwrite or append argument to True in the oml.ds.save invocation.

```
BEGIN
   sys.pyqScriptCreate('myLinearRegressionModel',
      'def fit_model(dat, modelName, datastoreName):
         import oml
         from sklearn import linear_model
         regr = linear_model.LinearRegression()
         regr.fit(dat.loc[:, ["Sepal_Length", "Sepal_Width", \
                              "Petal_Length"]], dat.loc[:,["Petal_Width"]])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
        oml.ds.save(objs={modelName:regr}, name=datastoreName,
overwrite=True)
        return str(regr)',
      FALSE, TRUE);
END;
/
```

Run a `SELECT` statement that invokes the `pyqTableEval` function. The `INP_NAM` parameter of the `pyqTableEval` function specifies the IRIS table as the data to pass to the Python function. The `PAR_LST` parameter specifies the names of the model and datastore to pass to the Python function, and specifies the `oml_connect` control argument to establish an OML4Py connection to the database during the invocation of the user-defined Python function. The `OUT_FMT` parameter specifies returning the value in XML format and the `SCR_NAME` parameter specifies the myLinearRegressionModel function in the script repository as the Python function to invoke. The XML output is a CLOB; you can call `set long [length]` to get more output.

```
SELECT *
FROM table(pyqTableEval(
  'IRIS',
  '{"modelName":"linregr",
    "datastoreName":"pymodel",
    "oml_connect":1}',
  'XML',
  'myLinearRegressionModel'));
```

The output is the following:

```
NAME  VALUE
----- ----------------------------------------------------------------
      <root><str>LinearRegression()</str></root>
```

14.6.4 pyqRowEval Function (On-Premises Database)

This topic describes the `pyqRowEval` function when used in an on-premises Oracle AI Database. The `pyqRowEval` function chunks data into sets of rows and then runs a user-defined Python function on each chunk.

The `pyqRowEval` function passes the data specified by the `INP_NAM` parameter to the Python function specified by the `SCR_NAME` parameter. You can pass arguments to the Python function with the `PAR_LST` parameter.

The `ROW_NUM` parameter specifies the maximum number of rows to pass to each invocation of the Python function. The last set of rows may have fewer rows than the number specified.

The Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You may define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
pyqRowEval(
    inp_nam VARCHAR2,
    par_lst VARCHAR2,
    out_fmt VARCHAR2,
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
    row_num NUMBER,
    scr_name VARCHAR2,
    scr_owner VARCHAR2 DEFAULT NULL)
```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}'
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
ROW_NUM	The number of rows to include in each invocation of the Python function.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Returns

Function `pyqRowEval` returns a table that has the structure specified by the `OUT_FMT` parameter value.

Example 14-19 Using the pyqRowEval Function

This example loads the Python model linregr to predict row chunks of sample iris data. The model is created and saved in the datastore pymodel in Example 14-18.

The example defines a Python function and stores it in the OML4Py script repository. It uses the user-defined Python function to create a database table as the result of the `pyqEval` function. It defines a Python function that runs a prediction function on a model loaded from the OML4Py datastore. It then invokes the `pyqTableEval` function to invoke the function on chunks of rows from the database table.

In a PL/SQL block, define the function `sample_iris_table` and store it in the script repository. The function loads the iris data set, creates two `pandas.DataFrame` objects, and then returns a sample of the concatenation of those objects.

```
BEGIN
  sys.pyqScriptCreate('sample_iris_table',
    'def sample_iris_table(size):
       from sklearn.datasets import load_iris
       import pandas as pd
       iris = load_iris()
       x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                        "Sepal_Width","Petal_Length","Petal_Width"])
       y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                                  2: "virginica"}[x], iris.target)),\
                        columns = ["Species"])
       return pd.concat([y, x], axis=1).sample(int(size))',
     FALSE, TRUE);  -- V_GLOBAL, V_OVERWRITE
END;
/
```

Create the SAMPLE_IRIS table in the database as the result of a `SELECT` statement, which invokes the `pyqEval` function on the `sample_iris_table` user-defined Python function saved in the script repository with the same name. The `sample_iris_table` function returns an iris data sample of size `size`.

```
CREATE TABLE sample_iris AS
SELECT *
  FROM TABLE(pyqEval(
               '{"size":20}',
               '{"Species":"varchar2(10)","Sepal_Length":"number",
                "Sepal_Width":"number","Petal_Length":"number",
                "Petal_Width":"number"}',
                'sample_iris_table'));
```

Define the Python function `predict_model` and store it with the name `linregrPredict` in the script repository. The function predicts the data in `dat` with the Python model specified by the `modelName` argument, which is loaded from the datastore specified by the `datastoreName`

Copyright © 2024, 2026, Oracle and/or its affiliates.

argument. The predictions are finally concatenated and returned with `dat` as the object that the function returns.

```
BEGIN
  sys.pyqScriptCreate('linregrPredict',
    'def predict_model(dat, modelName, datastoreName):
       import oml
       import pandas as pd
       objs = oml.ds.load(name=datastoreName, to_globals=False)
       pred = objs[modelName].predict(dat[["Sepal_Length","Sepal_Width",\
                                           "Petal_Length"]])
       return pd.concat([dat, pd.DataFrame(pred, \
                         columns=["Pred_Petal_Width"])], axis=1)',
     FALSE, TRUE);
END;
/
```

Run a `SELECT` statement that invokes the `pyqRowEval` function, which runs the specified Python function on each chunk of rows in the specified data set.

The `INP_NAM` argument specifies the data in the `SAMPLE_IRIS` table to pass to the Python function.

The `PAR_LST` argument specifies connecting to the OML4Py server with the special control argument `oml_connect`, passing the input data as a `pandas.DataFrame` with the special control argument `oml_input_type`, along with values for the function arguments `modelName` and `datastoreName`.

In the `OUT_FMT` argument, the JSON string specifies the column names and data types of the table returned by `pyqRowEval`.

The `ROW_NUM` argument specifies that five rows are included in each invocation of the function specified by `SCR_NAME`.

The `SCR_NAME` parameter specifies `linregrPredict`, which is the name in the script repository of the user-defined Python function to invoke.

```
SELECT *
  FROM table(pyqRowEval(
     'SAMPLE_IRIS',
     '{"oml_connect":1,"oml_input_type":"pandas.DataFrame",
       "modelName":"linregr", "datastoreName":"pymodel"}',
     '{"Species":"varchar2(10)", "Sepal_Length":"number",
       "Sepal_Width":"number", "Petal_Length":"number",
       "Petal_Width":"number","Pred_Petal_Width":"number"}',
     5,
     'linregrPredict'));
```

The output is the following:

```
Species    Sepal_Length Sepal_Width Petal_Length Petal_Width Pred_Petal_Width
---------- ------------ ----------- ------------ -----------
------------------
versicolor          5.4           3          4.5         1.5  1.66731546068336
versicolor            6         3.4          4.5         1.6  1.63208723397328
setosa              5.5         4.2          1.4         0.2
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
0.289325450127603
virginica          6.4         3.1         5.5         1.8  2.00641535609046
versicolor         6.1         2.8         4.7         1.2  1.58248012323666
setosa             5.4         3.7         1.5         0.2
0.251046097050724
virginica          7.2           3         5.8         1.6  1.97554457713195
versicolor         6.2         2.2         4.5         1.5  1.32323976658868
setosa             4.8         3.1         1.6         0.2
0.294116926466465
virginica          6.7         3.3         5.7         2.5  2.0936178656911
virginica          7.2         3.6         6.1         2.5  2.26646663788204
setosa               5         3.6         1.4         0.2
0.259261360689759
virginica          6.3         3.4         5.6         2.4  2.14639883810232
virginica          6.1           3         4.9         1.8  1.73186245496453
versicolor         6.1         2.9         4.7         1.4  1.60476297762276
versicolor         5.7         2.8         4.5         1.3  1.56056992978395
virginica          6.4         2.7         5.3         1.9  1.8124673155904
setosa               5         3.5         1.3         0.3
0.184570194825823
versicolor         5.6         2.7         4.2         1.3  1.40178874834007
setosa             4.5         2.3         1.3         0.3
0.0208089790714202
```

14.6.5 pyqGroupEval Function (On-Premises Database)

This topic describes the `pyqGroupEval` function when used in an on-premises Oracle AI Database. The `pyqGroupEval` function groups data by one or more columns and runs a user-defined Python function on each group.

The `pyqGroupEval` function runs the user-defined Python function specified by the `SCR_NAME` parameter. Pass data to the Python function with the `INP_NAM` parameter. Pass arguments to the Python function with the `PAR_LST` parameter. Specify one or more grouping columns with the `GRP_COL` parameter.

The Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. Define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
pyqGroupEval(
    inp_nam VARCHAR2,
    par_lst VARCHAR2,
    out_fmt VARCHAR2,
    grp_col VARCHAR2,
    scr_name VARCHAR2,
    scr_owner VARCHAR2 DEFAULT NULL)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}'
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. • The name of a table or view to use as a prototype. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.
GRP_COL	The names of the grouping columns by which to partition the data. Use commas to separate multiple columns. For example, to group by GENDER and YEAR: "GENDER,YEAR"
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Returns

Function `pyqGroupEval` returns a table that has the structure specified by the `OUT_FMT` parameter value.

Example 14-20 Using the pyqGroupEval Function

This example defines the Python function `create_iris_table` and stores it with the name create_iris_table in the OML4Py script repository. It then invokes `pyqEval`, which invokes the user-definded Python function and creates the IRIS database table. The example creates the package `irisPkg` and uses that package in specifying the data cursor to pass to the `irisGroupEval` function, which is a user-defined `pyqGroupEval` function. It defines another Python function, `group_count` and stores it in the script repository with the name mygroupcount. The example then invokes the `irisGroupEval` function and passes it the Python function saved with the name mygroupcount.

In a PL/SQL block, define the Python function `create_iris_table` and store in the script repository with the name create_iris_table.

```
BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
       from sklearn.datasets import load_iris
       import pandas as pd
       iris = load_iris()
       x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                        "Sepal_Width","Petal_Length","Petal_Width"])
       y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                                  2: "virginica"}[x], iris.target)),\
                        columns = ["Species"])
       return pd.concat([y, x], axis=1)');
END;
/
```

Invoke the `pyqEval` function to create the database table `IRIS`, using the Python function stored with the name `create_iris_table` in the script repository.

```
CREATE TABLE IRIS AS
(SELECT * FROM pyqEval(
      NULL,
      '{"Species":"VARCHAR2(10)","Sepal_Length":"number",
        "Sepal_Width":"number","Petal_Length":"number",
        "Petal_Width":"number"}',
      'create_iris_table'
));
```

Define the Python function `group_count` and store it with the name `mygroupcount` in the script repository. The function returns a `pandas.DataFrame` generated on each group of data `dat`.

```
BEGIN
  sys.pyqScriptCreate('mygroupcount',
    'def group_count(dat):
    import pandas as pd
    return pd.DataFrame([(dat["Species"][0], dat.shape[0])],\
    columns = ["Species", "CNT"]) ');
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
END;
/
```

Issue a query that invokes the `pyqGroupEval` function. In the function, the `INP_NAM` argument specifies the data in the IRIS table to pass to the function.

The `PAR_LST` argument specifies the special control argument `oml_input_type`.

The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the table returned by `pyqGroupEval`.

The `GRP_COL` parameter specifies the column to group by.

The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `mygroupcount` in the script repository.

```
SELECT *
   FROM table(
    pyqGroupEval(
      'IRIS',
      '{"oml_input_type":"pandas.DataFrame"}',
      '{"Species":"varchar2(10)", "CNT":"number"}',
      'Species',
      'mygroupcount'));
```

The output is the following.

```
Species           CNT
---------- ----------
setosa             50
versicolor         50
virginica          50
```

14.6.6 pyqGrant Function (On-Premises Database)

This topic describes the `pyqGrant` function when used in an on-premises Oracle AI Database.

The `pyqGrant` function grants read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```
pyqGrant (
    V_NAME          VARCHAR2    IN
    V_TYPE          VARCHAR2    IN
    V_USER          VARCHAR2    IN     DEFAULT)
```

Parameters

Parameter	Description
V_NAME	The name of an OML4Py datastore or a script in the OML4Py script repository.
V_TYPE	For a datastore, the type is `datastore`; for script the type is `pyqScript`.
V_USER	The name of the user to whom to grant access.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Example 14-21 Granting Read Access to a script

```
-- Grant read privilege access to Scott.
BEGIN
  pyqGrant('pyqFun1', 'pyqscript', 'SCOTT');
END;
/
```

Example 14-22 Granting Read Access to a datastore

```
-- Grant read privilege access to datastore ds1 to SCOTT.
BEGIN
  pyqGrant('ds1', 'datastore', 'SCOTT');
END;
/
```

Example 14-23 Granting Read Access to a Script to all Users

```
-- Grant read privilege access to script RandomRedDots to all users.
BEGIN
  pyqGrant('pyqFun1', 'pyqscript', NULL);
END;
/
```

Example 14-24 Granting Read Access to a datastore to all Users

```
-- Grant read privilege access to datastore ds1 to all users.
BEGIN
  pyqGrant('ds1', 'datastore', NULL);
END;
/
```

14.6.7 pyqRevoke Function (On-Premises Database)

This topic describes the `pyqRevoke` function when used in an on-premises Oracle AI Database.

The `pyqRevoke` function revokes read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```
pyqRevoke (
    V_NAME          VARCHAR2    IN
    V_TYPE          VARCHAR2    IN
    V_USER          VARCHAR2    IN     DEFAULT)
```

Parameters

Parameter	Description
V_NAME	The name of an OML4Py datastore or a script in the OML4Py script repository.
V_TYPE	For a datastore, the type is `datastore`; for script the type is `pyqScript`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
V_USER	The name of the user from whom to revoke access.

Example 14-25 Revoking Read Access to a script

```
-- Revoke read privilege access to script pyqFun1 from SCOTT.
BEGIN
  pyqRevoke('pyqFun1', 'pyqscript', 'SCOTT');
END;
/
```

Example 14-26 Revoking Read Access to a datastore

```
-- Revoke read privilege access to datastore ds1 from SCOTT.
BEGIN
  pyqRevoke('ds1', 'datastore', 'SCOTT');
END;
/
```

Example 14-27 Revoking Read Access to a script from all Users

```
-- Revoke read privilege access to script pyqFun1 from all users.
BEGIN
  pyqRevoke('pyqFun1', 'pyqscript', NULL);
END;
/
```

Example 14-28 Revoking Read Access to a datastore from all Users

```
-- Revoke read privilege access to datastore ds1 from all users.
BEGIN
  pyqRevoke('ds1', 'datastore', NULL);
END;
/
```

14.6.8 pyqScriptCreate Procedure (On-Premises Database)

This topic describes the `pyqScriptCreate` procedure in an on-premises Oracle AI Database. The `pyqScriptCreate` procedure creates a user-defined Python function and adds it to the OML4Py script repository.

To create a user-defined Python function, you must have the PYQADMIN database role.

Syntax

```
sys.pyqScriptCreate (
    V_NAME          VARCHAR2    IN
    V_SCRIPT        CLOB        IN
    V_GLOBAL        BOOLEAN     IN     DEFAULT
    V_OVERWRITE     BOOLEAN     IN     DEFAULT)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
V_NAME	A name for the user-defined Python function in the OML4Py script repository.
V_SCRIPT	The definition of the Python function.
V_GLOBAL	TRUE specifies that the user-defined Python function is public; FALSE specifies that the user-defined Python function is private.
V_OVERWRITE	If the script repository already has a user-defined Python function with the same name as V_NAME, then TRUE replaces the content of that user-defined Python function with V_SCRIPT and FALSE does not replace it.

Example 14-29 Using the pyqScriptCreate Procedure

This example creates a private user-defined Python function named pyqFun2 in the OML4Py script repository.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
       import numpy as np
       import pickle
       z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
       [float(x)/10 for x in range(10)],
       [x for x in range(10)],
       [bool(x%2) for x in range(10)],
       [pickle.dumps(x) for x in range(10)],
       ["test"+str(x**2) for x in range(10)])],
       dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
       ("e", "S20"), ("f", "O")])
       return z');
END;
/
```

This example creates a global user-defined Python function named pyqFun2 in the script repository and overwrites any existing user-defined Python function of the same name.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
       import numpy as np
       import pickle
       z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
       [float(x)/10 for x in range(10)],
       [x for x in range(10)],
       [bool(x%2) for x in range(10)],
       [pickle.dumps(x) for x in range(10)],
       ["test"+str(x**2) for x in range(10)])],
       dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
       ("e", "S20"), ("f", "O")])
       return z',
       TRUE,  -- Make the user-defined Python function global.
       TRUE); -- Overwrite any global user-defined Python function
              -- with the same name.
END;
/
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

This example creates a private user-defined Python function named `create_iris_table` in the script repository.

```
BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
       from sklearn.datasets import load_iris
       import pandas as pd
       iris = load_iris()
       x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                        "Sepal_Width","Petal_Length","Petal_Width"])
       y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                                 2: "virginica"}[x], iris.target)),\
                        columns = ["Species"])
       return pd.concat([y, x], axis=1)');
END;
/
```

Display the user-defined Python functions owned by the current user.

```
SELECT * from USER_PYQ_SCRIPTS;
```

```
NAME               SCRIPT
-----------------
------------------------------------------------------------------------
create_iris_table  def create_iris_table():       from sklearn.datasets
import load_iris ...
pyqFun2            def return_frame():       import numpy as np       import
pickle      ...
```

Display the user-defined Python functions available to the current user.

```
SELECT * from ALL_PYQ_SCRIPTS;
```

```
OWNER     NAME               SCRIPT
--------   -----------------
-----------------------------------------------------------------------
OML_USER   create_iris_table  "def create_iris_table():  from
sklearn.datasets import load_iris ...
OML_USER   pyqFun2             "def return_frame():  import numpy as np
import pickle      ...
PYQSYS     pyqFun2             "def return_frame():  import numpy as np
import pickle      ...
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

14.6.9 pyqScriptDrop Procedure (On-Premises Database)

This topic describes the `pyqScriptDrop` procedure in an on-premises Oracle AI Database. The `pyqScriptDrop` procedure removes a user-defined Python function from the OML4Py script repository.

To drop a user-defined Python function, you must have the `PYQADMIN` database role.

Syntax

```
sys.pyqScriptDrop (
    V_NAME          VARCHAR2    IN
    V_GLOBAL        BOOLEAN     IN     DEFAULT
    V_SILENT        BOOLEAN     IN     DEFAULT)
```

Parameter	Description
V_NAME	A name for the user-defined Python function in the OML4Py script repository.
V_GLOBAL	A BOOLEAN that specifies whether the user-defined Python function to drop is a global or a private user-defined Python function. The default value is FALSE, which indicates a private user-defined Python function. TRUE specifies that the user-defined Python function is public.
V_SILENT	A BOOLEAN that specifies whether to display an error message when `sys.pyqScriptDrop` encounters an error in dropping the specified user-defined Python function. The default value is FALSE.

Example 14-30 Using the sys.pyqScriptDrop Procedure

For the creation of the user-defined Python functions dropped in these examples, see Example 14-29.

This example drops the private user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
  sys.pyqScriptDrop('pyqFun2');
 END;
 /
```

This example drops the global user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
  sys.pyqScriptDrop('pyqFun2', TRUE);
 END;
 /
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

14.7 SQL API for Embedded Python Execution with Autonomous AI Database

The SQL API for Embedded Python Execution with Autonomous AI Database provides SQL interfaces for setting authorization tokens, managing access control list (ACL) privileges, executing Python scripts, and synchronously and asynchronously running jobs.

The following topics describe the SQL API interfaces for Embedded Python Execution.

Topics:

- Access and Authorization Procedures and Functions
  Use the network access control lists (ACL) API to control access by users to external network services and resources from the database. Use the token store API to persist the authorization token issued by a cloud host so it can be used with subsequent SQL calls.
- Embedded Python Execution Functions (Autonomous Database)
  The SQL API for Embedded Python Execution with Autonomous AI Database functions are described in the following topics.
- Asynchronous Jobs (Autonomous AI Database)
  When a function is run asynchronously, it's run as a job which can be tracked by using the `pyqJobStatus` and `pyqJobResult` functions.
- Special Control Arguments (Autonomous AI Database)
  Use the `PAR_LST` parameter to specify special control arguments and additional arguments to be passed into the Python script.
- Output Formats (Autonomous AI Database)
  The `OUT_FMT` parameter controls the format of output returned by the table functions `pyqEval`, `pyqGroupEval`, `pyqIndexEval`, `pyqRowEval`, `pyqTableEval`, and `pyqJobResult`.

14.7.1 Access and Authorization Procedures and Functions

Use the network access control lists (ACL) API to control access by users to external network services and resources from the database. Use the token store API to persist the authorization token issued by a cloud host so it can be used with subsequent SQL calls.

Use the following to manage ACL privileges. An `ADMIN` user is required.

- pyqAppendHostACE Procedure
- pyqGetHostACE Function
- pyqRemoveHostACE Procedure

Use the following to manage authorization tokens:

- pyqSetAuthToken Procedure
- pyqIsTokenSet Function

Workflow

The typical workflow for using the SQL API for Embedded Python Execution with Autonomous AI Database is:

Copyright © 2024, 2026, Oracle and/or its affiliates.

1. Connect to PDB as the `ADMIN` user, and add a normal user `OMLUSER` to the ACL list of the cloud host of which the root domain is `adb.us-region-1.oraclecloudapps.com`:

   ```
   exec pyqAppendHostAce('OMLUSER','adb.us-region-1.oraclecloudapps.com');
   ```

2. The OML Rest URLs can be obtained from the Oracle Autonomous AI Database that is provisioned.

   a. Sign into your Oracle Cloud Infrastructure account. You will need your OCI user name and password.

   b. Click the hamburger menu and select Autonomous AI Database instance that is provisioned. For more information on provisioning an Autonomous AI Database, see: Provision an Oracle Autonomous AI Database.

   c. Click **Service Console** and then click **Devlopment**.

   d. Scroll down to **Oracle Machine Learning RESTful Services** tile and click **Copy** to obtain the following URLs for:

      - Obtaining the REST authentication token for REST APIs provided by OML:

        ```
        <oml-cloud-service-location-url>/omlusers/
        ```

   The URL `<oml-cloud-service-location-url>` includes the tenancy ID, location, and database name. For example, `https://qtraya2braestch-omldb.adb.us-sanjose-1.oraclecloudapps.com`.

   In this example,

   - `qtraya2braestch` is the tenancy ID
   - `omldb` is the database name
   - `us-sanjose-1` is the datacenter region
   - `oraclecloudapps.com` is the root domain

3. The Oracle Machine Learning REST API uses tokens to authenticate an Oracle Machine Learning user. To authenticate and obtain an access token, send a POST request to the Oracle Machine Learning User Management Cloud Service REST endpoint `/oauth2/v1/token` with your OML username and password.

   ```
   curl -X POST --header 'Content-Type: application/json' --header 'Accept:
   application/json'
   -d '{"grant_type":"password", "username":"'${username}'", "password":"'$
   {password}'"}'
   "<oml-cloud-service-location-url>/omlusers/api/oauth2/v1/token"
   ```

   The example uses the following values:

   - `username` is the OML username.
   - `password` is the OML user password.
   - `oml-cloud-service-location-url` is a variable containing the REST server portion of the Oracle Machine Learning User Management Cloud Service instance URL that includes the tenancy ID, database name, and the location name. You can obtain the omlserver URL from the Development tab in the Service Console of your Autonomous AI Database instance.

Copyright © 2024, 2026, Oracle and/or its affiliates.

> **Note**
>
> When a token expires, all calls to the OML Services REST endpoints with return a message stating that the token has expired along with the HTTP error: `HTTP/1.1 401 Unauthorized`

4. Connect to PDB as `OMLUSER`, set the access token, and run `pyqIndexEval`:

```
exec pyqSetAuthToken('<access token>');
select *
    from table(pyqIndexEval(
        par_lst => NULL,
        out_fmt => '{"ID":"number", "RES":"varchar2(3)"}',
        times_num => 3,
        scr_name => 'idx_ret_df'));

        ID RES
---------- ---
         1 a
         2 b
         3 c

3 rows selected.
```

The following topics describe the steps to manage ACL privileges and authorization tokens.

Topics:

- pyqAppendHostACE Procedure
  The `pyqAppendHostACE` procedure appends an access control entry (ACE) to the access control list (ACL) of the cloud host. The ACL controls access to the cloud host from the database, and the ACE specifies the connect privilege granted to the specified user name.
- pyqGetHostACE Function
  The `pyqGetHostACE` function gets the existing host access control entry (ACE) for the specified user. An exception is raised if the host ACE doesn't exist for the specified user.
- pyqRemoveHostACE Procedure
- pyqSetAuthToken Procedure
  The `pyqSetAuthToken` procedure sets the access token in the token store.
- pyqIsTokenSet Function
  The `pyqIsTokenSet` function returns whether the authorization token is set or not.

14.7.1.1 pyqAppendHostACE Procedure

The `pyqAppendHostACE` procedure appends an access control entry (ACE) to the access control list (ACL) of the cloud host. The ACL controls access to the cloud host from the database, and the ACE specifies the connect privilege granted to the specified user name.

Syntax

```
PROCEDURE SYS.pyqAppendHostACE(
    username IN VARCHAR2,
    host_root_domain IN VARCHAR2
)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter

username - Database user to whom the connect privilege to the cloud host is granted.

host_root_domain - Root domain of the cloud host. For example, if the URL is `https://qtraya2braestch-omldb.adb.us-sanjose-1.oraclecloudapps.com`, the root domain of the cloud host is: `adb.us-sanjose-1.oraclecloudapps.com`.

Example

```
exec pyqAppendHostAce('OMLUSER','adb.us-region-1.oraclecloudapps.com');
```

> **Note**
>
> OML username is case sensitive

14.7.1.2 pyqGetHostACE Function

The `pyqGetHostACE` function gets the existing host access control entry (ACE) for the specified user. An exception is raised if the host ACE doesn't exist for the specified user.

Syntax

```
FUNCTION sys.pyqGetHostACE(
  p_username IN VARCHAR2
)
```

Parameter

p_username - Database user to look for the host ACE.

Example

If user `OMLUSER` has access to the cloud host, i.e., `ibuwlq4mjqkeils-omlrgpy1.adb.us-region-1.oraclecloudapps.com`, the ADMIN user can run the following to check the user's privileges:

```
SQL> set serveroutput on
DECLARE
    hostname VARCHAR2(4000);
BEGIN
    hostname := pyqGetHostACE('OMLUSER');
    DBMS_OUTPUT.put_line ('hostname: ' || hostname);
END;
/
SQL> hostname: ibuwlq4mjqkeils-omlrgpy1.adb.us-region-1.oraclecloudapps.com
PL/SQL procedure successfully completed.
```

14.7.1.3 pyqRemoveHostACE Procedure

The `pyqRemoveHostACE` procedure removes the existing host access control entry (ACE) from the specified *username*. If an access token was set for the cloud host, the token is also removed. An exception is raised if the host ACE does not exist.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Syntax

```
PROCEDURE SYS.pyqRemoveHostACE(
   username IN  VARCHAR2
)
```

Parameter

username - Database user from whom the connect privilege to the cloud host is revoked.

14.7.1.4 pyqSetAuthToken Procedure

The `pyqSetAuthToken` procedure sets the access token in the token store.

Syntax

```
PROCEDURE SYS.pyqSetAuthToken(
  access_token IN VARCHAR2
)
```

14.7.1.5 pyqIsTokenSet Function

The `pyqIsTokenSet` function returns whether the authorization token is set or not.

Syntax

```
FUNCTION SYS.pyqIsTokenSet() RETURN BOOLEAN
```

Example

The following example shows how to use the `pyqSetAuthToken` procedure and the `pyqIsTokenSet` function.

```
DECLARE
    is_set BOOLEAN;
BEGIN
    pyqSetAuthToken('<access token>');
    is_set := pyqIsTokenSet();
    IF (is_set) THEN
        DBMS_OUTPUT.put_line ('token is set');
    END IF;
END;
/
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

14.7.2 Embedded Python Execution Functions (Autonomous Database)

The SQL API for Embedded Python Execution with Autonomous AI Database functions are described in the following topics.

Topics

- pyqListEnvs Function (Autonomous AI Database)
  The function `pyqListEnvs` when used in Oracle Autonomous AI Database, lists the environments saved in an Object Storage.
- pyqEval Function (Autonomous AI Database)
  The function `pyqEval`, when used in Oracle Autonomous AI Database, calls a user-defined Python function. Users can pass arguments to the user-defined Python function.
- pyqTableEval Function (Autonomous AI Database)
  The function `pyqTableEval` function when used in Oracle Autonomous AI Database, runs a user-defined Python function on data from an Oracle AI Database table.
- pyqRowEval Function (Autonomous AI Database)
  The function `pyqRowEval` when used in Oracle Autonomous AI Database, chunks data into sets of rows and then runs a user-defined Python function on each chunk.
- pyqGroupEval Function (Autonomous AI Database)
  The function `pyqGroupEval` when used in Oracle Autonomous AI Database, groups data by one or more columns and runs a user-defined Python function on each group.
- pyqIndexEval Function (Autonomous AI Database)
  The function `pyqIndexEval` when used in Oracle Autonomous AI Database, runs a user-defined Python function multiple times as required in the Python engines spawned by the database environment.
- pyqListAllJobs Function
  Use the `pyqListAllJobs` function to return the list of jobs submitted.
- pyqDeleteJob Function
  Use the `pyqDeleteJob` function to delete a job.
- pyqGrant Function (Autonomous AI Database)
  This topic describes the `pyqGrant` function when used in Oracle Autonomous AI Database.
- pyqRevoke Function (Autonomous AI Database)
  This topic describes the `pyqRevoke` function when used in Oracle Autonomous AI Database.
- pyqScriptCreate Procedure (Autonomous AI Database)
  This topic describes the `pyqScriptCreate` procedure in Oracle Autonomous AI Database. Use the `pyqScriptCreate` procedure to create a user-defined Python function and add it to the OML4Py script repository.
- pyqScriptDrop Procedure (Autonomous AI Database)
  This topic describes the `pyqScriptDrop` procedure in Oracle Autonomous AI Database. Use the `pyqScriptDrop` procedure to remove a user-defined Python function from the OML4Py script repository.

14.7.2.1 pyqListEnvs Function (Autonomous AI Database)

The function `pyqListEnvs` when used in Oracle Autonomous AI Database, lists the environments saved in an Object Storage.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Syntax

```
FUNCTION PYQSYS.pyqListEnvs
RETURN SYS.AnyDataSet
```

Result

The function returns a rowset with the following columns:

Column Name	Data Type	Description
name	VARCHAR2(256)	Name of the Conda environment.
size	VARCHAR2(32)	Size (e.g., "1GB") of the enviorment
description	VARCHAR2(4000)	Description of the environment.
number_of_installed_packages	NUMBER	Number of Python packages installed in the environment.

Example

Issue a query that calls the `pyqListEnvs` function and lists the environments present.

```
select * from table(pyqListEnvs());
```

The output is in a tabular format similar to the following:

Table 14-2 output

name	size	description	number_of_installed_packages
mypyenv	2.1 GiB	Install Python seaborn and tensorflow packages	148
mykerasenv	210.3 MiB	Install keras packages	25
py312	1.3 GiB	R env with Python 3.12	357
mypyenv3	1.7 GiB	huggingface with tensorflow	172
sbenv	545.5 MiB	Conda environment with seaborn	81

5 rows selected.

14.7.2.2 pyqEval Function (Autonomous AI Database)

The function `pyqEval`, when used in Oracle Autonomous AI Database, calls a user-defined Python function. Users can pass arguments to the user-defined Python function.

The function `pyqEval` does not automatically load the data. Within the user-defined Python function, the user may explicitly access and/or retrieve data using the transparency layer or an ROracle database connection.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Syntax

```
FUNCTION PYQSYS.pyqEval(
    PAR_LST    VARCHAR2,
    OUT_FMT    VARCHAR2,
    SCR_NAME   VARCHAR2,
    SCR_OWNER  VARCHAR2 DEFAULT NULL,
    ENV_NAME   VARCHAR2 DEFAULT NULL
    )
    RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}' **See also:** Special Control Arguments (Autonomous AI Database).
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. • The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. **See also:** Output Formats (Autonomous AI Database).
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

This example defines a Python function and stores it in the OML4Py script repository. It calls the pyqEval function on the user-defined Python functions.

In a PL/SQL block, create a Python function that is stored in script repository with the name pyqFun1.

```
begin
    sys.pyqScriptCreate('pyqFun1',
        'def fun_tab():
                import pandas as pd
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                names = ["demo_"+str(i) for i in range(10)]
                ids = [x for x in range(10)]
                floats = [float(x)/10 for x in range(10)]
                d = {''ID'': ids, ''NAME'': names, ''FLOAT'': floats}
                scores_table = pd.DataFrame(d)
                return scores_table
',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/
```

Next, call the `pyqEval` function, which runs the user-defined Python function.

The `PAR_LST` argument specifies using `LOW` service level with the special control argument `oml_service_level`.

In the `OUT_FMT` argument, the string `'JSON'`, specifies that the table returned contains a CLOB that is a JSON string.

The `SCR_NAME` parameter specifies the `pyqFun1` function in the script repository as the Python function to call.

The JSON output is a CLOB. You can call `set long [length]` to get more output.

```
set long 500
select *
    from table(pyqEval(
        par_lst => '{"oml_service_level":"LOW"}',
        out_fmt => 'JSON',
        scr_name => 'pyqFun1'));
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------------
[{"FLOAT":0,"ID":0,"NAME":"demo_0"},{"FLOAT":0.1,"ID":1,"NAME":"demo_1
"},{"FLOAT":0.2,"ID":2,"NAME":"demo_2"},{"FLOAT":0.3,"ID":3,"NAME":"de
mo_3"},{"FLOAT":0.4,"ID":4,"NAME":"demo_4"},{"FLOAT":0.5,"ID":5,"NAME"
:"demo_5"},{"FLOAT":0.6,"ID":6,"NAME":"demo_6"},{"FLOAT":0.7,"ID":7,"N
AME":"demo_7"},{"FLOAT":0.8,"ID":8,"NAME":"demo_8"},{"FLOAT":0.9,"ID":
9,"NAME":"demo_9"}]

1 row selected.
```

Issue another query that invokes the same `pyqFun1` script. The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the structured table output.

```
select *
    from table(pyqEval(
        par_lst => '{"oml_service_level":"LOW"}',
        out_fmt => '{"ID":"number", "NAME":"VARCHAR2(8)",
"FLOAT":"binary_double"}',
        scr_name => 'pyqFun1'));
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

The output is the following:

```
ID NAME FLOAT
   0 demo_0 0.0
   1 demo_1 0.1
   2 demo_2 0.2
   3 demo_3 0.3
   4 demo_4 0.4
   5 demo_5 0.5
   6 demo_6 0.6
   7 demo_7 0.7
   8 demo_8 0.8
   9 demo_9 0.9


10 rows selected.
```

Use the following code to create the "seaborn" environment based on Python version 3.12.6 and upload the environment to object storage owned by the Pluggable Database (PDB).

> **Note**
>
> Admin privilege is required to create and manage the Conda environments.

```
create -n seaborn python=3.12.6 seaborn
upload seaborn --overwrite --description 'Python package for seaborn' -t
application OML4PY
```

The data visualization library 'seaborn' is installed in the environment.

Use the following code to create the script 'test_seaborn_noinp':

```
begin
  sys.pyqScriptCreate('test_seaborn_noinp',
  'def fun_tab():
  import seaborn as sns
  import matplotlib.pyplot as plt
  import numpy as np
  import pandas as pd
  data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]],
size=2000)
  data = pd.DataFrame(data, columns=["x", "y"])
  sns.displot(data["x"])
  plt.title("Dist plot")
  plt.show()
  return "hello world" ',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/
```

This example calls the pyqEval function, which runs the specified Python script.

The `PAR_LST` argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

Copyright © 2024, 2026, Oracle and/or its affiliates.

In the OUT_FMT arguments, the string 'PNG', specifies returning a table with BLOB containing the images generated by the Python function.

The SCR_NAME parameter specifies the 'test_seaborn_noinp' script in the script repository as the Python function to call.

The ENV_NAME parameter specifies 'seaborn', which is the Conda environment to run the Python function.

```
select *
  from table(pyqEval(
        par_lst => '{"oml_graphics_flag":true}',
        out_fmt => 'PNG',
        scr_name => 'test_seaborn_noinp',
        scr_owner => NULL,
        env_name => 'seaborn'
));
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--

         1
"hello world"

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
Lineplot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
73

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000000F6100000F6101
A8
3FA7690000682C49444154789CEDDD797C5355FE3FFE579236E9BEB7E942DBB414286B0B2D9482
0A
4AC7023A82A2022E2C83B801A37674147F0A2EDFCF1415114719D119293AC280CC208EC8A050D9
84

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
```

In a PL/SQL block, define the Python function `create_iris_table` and store in the script repository with the name `create_iris_table`, overwriting any existing user-defined Python function stored in the script repository with the same name.

The `create_iris_table` function imports and loads the iris data set, creates two `pandas.DataFrame` objects, and then returns the concatenation of those objects.

```
BEGIN
    sys.pyqScriptCreate('create_iris_table',
        'def create_iris_table():
            from sklearn.datasets import load_iris
            import pandas as pd
            iris = load_iris()
            x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                "Sepal_Width", "Petal_Length", "Petal_Width"])
            y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                2: "virginica"}[x], iris.target)),\
                columns = ["Species"])
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
            return pd.concat([y, x], axis=1)',
          FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
   END;
/
CREATE TABLE IRIS AS
(SELECT * FROM pyqEval(
    NULL,
    '{"Species":"VARCHAR2(10)","Sepal_Length":"number",
      "Sepal_Width":"number","Petal_Length":"number",
      "Petal_Width":"number"}',
    'create_iris_table'
));
```

14.7.2.3 pyqTableEval Function (Autonomous AI Database)

The function `pyqTableEval` function when used in Oracle Autonomous AI Database, runs a user-defined Python function on data from an Oracle AI Database table.

Pass data to the user-defined Python function from the table name specified in the `INP_NAM` parameter. Pass arguments to the user-defined Python function with the `PAR_LST` parameter.

The user-defined Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
FUNCTION PYQSYS.pyqTableEval(
   INP_NAM    VARCHAR2,
   PAR_LST    VARCHAR2,
   OUT_FMT    VARCHAR2,
   SCR_NAME   VARCHAR2,
   SCR_OWNER  VARCHAR2 DEFAULT NULL,
   ENV_NAME   VARCHAR2 DEFAULT NULL
   )
   RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
`INP_NAM`	The name of a table or view that specifies the data to pass to the Python function specified by the `SCR_NAME` parameter. If using a table or view owned by another user, use the format `<owner name>.<table/view name>`. You must have read access to the specified table or view.
`PAR_LST`	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the `SCR_NAME` parameter. Special control arguments, which start with `oml_`, are not passed to the function specified by `SCR_NAME`, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as `pandas.DataFrame`, use: `'{"oml_input_type":"pandas.DataFrame"}'` **See also:** Special Control Arguments (Autonomous AI Database).

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a `pandas.DataFrame`, a `numpy.ndarray`, a `tuple`, or a list of `tuples`. • The string `'JSON'`, which specifies that the table returned contains a CLOB that is a JSON string. • The string `'XML'`, which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string `'PNG'`, which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. **See also:** Output Formats (Autonomous AI Database).
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is `NULL`. If `NULL`, will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

Define the Python function `fit_model` and store it with the name `myLinearRegressionModel` as a private function in the script repository, overwriting any existing user-defined Python function stored with that name.

The `fit_model` function fits a regression model to the input data `dat` and then saves the fitted model as an object specified by the `modelName` argument to the datastore specified by the `datastoreName` argument. The `fit_model` function returns the fitted model in a string format.

By default, Python objects are saved to a new datastore with the specified `datastoreName`. To save an object to an existing datastore, either set the `overwrite` or `append` argument to `True` in the `oml.ds.save` invocation.

```
BEGIN
    sys.pyqScriptCreate('myLinearRegressionModel',
        'def fit_model(dat, modelName, datastoreName):
            import oml
            from sklearn import linear_model
            regr = linear_model.LinearRegression()
            regr.fit(dat.loc[:, ["Sepal_Length", "Sepal_Width", \
                                 "Petal_Length"]],
                     dat.loc[:,["Petal_Width"]])
            oml.ds.save(objs={modelName:regr}, name=datastoreName,
                        overwrite=True)
            return str(regr)',
        FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Use the following code to create the 'test_seaborn_inp' script:

```
begin  sys.pyqScriptCreate('test_seaborn_inp',
         'def fun_tab(dat):
          import seaborn as sns
          import matplotlib.pyplot as plt
          sns.lineplot(x="Sepal_Length", y="Sepal_Width", data=dat)
          plt.title("Iris plot")
          plt.show()
          return "hello world"
          ',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/
```

This example calls the pyqTableEval function, which runs the specified Python function on the specified data set.

`The INP_NAM` argument specifies the data in the IRIS table to pass to the Python function.

`The PAR_LST` argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

The `OUT_FMT` arguments specifies returning a table with BLOB containing the images generated by the Python function.

The `SCR_NAME` parameter specifies the 'test_seaborn_inp' script, which is the name in the script repository of the user-defined Python function to invoke.

The `ENV_NAME` parameter specifies 'seaborn', which is a Conda environment created in pyqEval Function (Autonomous AI Database) .

```
select *
  from table(pyqTableEval(
        inp_nam =>  'IRIS',
        par_lst => '{"oml_graphics_flag":true}',
        out_fmt => 'PNG',
        scr_name => 'test_seaborn_inp',
        scr_owner => NULL,
        env_name => 'seaborn'
));
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--

         1
"hello world"

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
Iris plot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000 00F6100000F6101
A8
3FA7690000B9BC49444154789CECDD797CDC759D3FF0D7F79A2B9399DC499BA44DEF527A41B9CA
61
3945C042576559500AABAC2BE2AE8ABA520459442CBA80E0FA0304517057B60A0B28972C22E5A6
F4

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--
IMAGE
--------------------------------------------------------------------------------
--
```

This example uses the IRIS table created in the example shown in `pyqEval` Function (Autonomous AI Database). Run a `SELECT` statement that invokes the `pyqTableEval` function. The `INP_NAM` parameter of the `pyqTableEval` function specifies the IRIS table as the data to pass to the Python function. The `PAR_LST` parameter specifies the names of the model and datastore to pass to the Python function. The `OUT_FMT` parameter specifies returning the value in XML format and the `SCR_NAME` parameter specifies the `myLinearRegressionModel` function in the script repository as the Python function to invoke. The XML output is a CLOB; you can call `set long [length]` to get more output.

```
SELECT *
FROM table(pyqTableEval(
    inp_nam => 'IRIS',
    par_lst => '{"modelName":"linregr",
                 "datastoreName":"pymodel"}',
out_fmt => 'XML',
scr_name => 'myLinearRegressionModel'));
```

The output is the following:

```
NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
<root><str>LinearRegression()</str></root>
1 row selected.
```

14.7.2.4 pyqRowEval Function (Autonomous AI Database)

The function `pyqRowEval` when used in Oracle Autonomous AI Database, chunks data into sets of rows and then runs a user-defined Python function on each chunk.

The `ROW_NUM` parameter specifies the maximum number of rows to pass to each invocation of the user-defined Python function. The last set of rows may have fewer rows than the number specified.

The user-defined Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. You can define the form of the returned value with the `OUT_FMT` parameter.

Syntax

```
FUNCTION PYQSYS.pyqRowEval(
    INP_NAM    VARCHAR2,
    PAR_LST    VARCHAR2,
    OUT_FMT    VARCHAR2,
    ROW_NUM    NUMBER,
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
SCR_NAME   VARCHAR2,
SCR_OWNER  VARCHAR2 DEFAULT NULL,
ENV_NAME   VARCHAR2 DEFAULT NULL
)
RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}' **See also:** Special Control Arguments (Autonomous AI Database).
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. • The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. **See also:** Output Formats (Autonomous AI Database).
ROW_NUM	The number of rows in a chunk. The Python script is executed in each chunk.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

This example calls the pyqRowEval function, which runs the specified Python script on each chunk of rows in the specified data set.

The INP_NAM argument specifies the data in the IRIS table to pass to the Python function.

The PAR_LST argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The OUT_FMT arguments specifies returning a table with BLOB containing the images generated by the Python function.

The ROW_NUM argument specifies that 50 rows are included in each invocation of the function specified by SCR_NAME.

The SCR_NAME parameter specifies the 'test_seaborn_inp' script, which is created in pyqTableEval Function (Autonomous AI Database).

The ENV_NAME parameter specifies 'seaborn', which is a Conda environment created in pyqEval Function (Autonomous AI Database) .

```
select *
  from table(pyqRowEval(
        inp_nam =>  'IRIS',
        par_lst => '{"oml_graphics_flag":true}',
        out_fmt => 'PNG',
        row_num => 50,
        scr_name => 'test_seaborn_inp',
        scr_owner => NULL,
        env_name => 'seaborn'
));
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
CHUNK_1
         1
"hello world"

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--
Iris plot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000 00F6100000F6101
A8
3FA769000081254944415 4789CEDDD7774D5F5FD3FF0E767DC99BD13C82081B0041450101C888A
0A
54455B57ADA3167F75B46AF5EBB7DA6FADB5D6E2AAA3B52A6A155A6B69B56A97685DE042050105
8A

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--

CHUNK_2
         1

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--
IMAGE
--------------------------------------------------------------------------------
--
"hello world"
Iris plot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973 00000F6100000F6101
A8
3FA7690000ABB149444154789CECDD79985C65993EFEFB2C55A7BABBBA7A4D6F49670F21210B10
02

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
84C5B093804846C5AF8E4C4445470467181CD1388A0A32114401C71FA88802A318070750190920
92

CHUNK_3

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
         1
"hello world"
Iris plot

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000 00F6100000F6101
A8

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
3FA76900008C7149444154789CEDDD77945BD5D536F0E7AA774D2F9EE25EB14D316D3060209862
E2
50120204B009900487040229E00402841843288 1242FA663E7230402015E9AF14B33C5543730B6
71
```

Example

This example loads the Python model `linregr` to predict row chunks of sample iris data. The model is created and saved in the datastore `pymodel`, which is shown in the example for pyqTableEval Function (Autonomous AI Database).

Copyright © 2024, 2026, Oracle and/or its affiliates.

The example defines a Python function and stores it in the OML4Py script repository. It uses the user-defined Python function to create a database table as the result of the `pyqEval` function. It defines a Python function that runs a prediction function on a model loaded from the OML4Py datastore. It then invokes the `pyqTableEval` function to invoke the function on chunks of rows from the database table.

In a PL/SQL block, define the function `sample_iris_table` and store it in the script repository. The function loads the iris data set, creates two `pandas.DataFrame` objects, and then returns a sample of the concatenation of those objects.

```
BEGIN
    sys.pyqScriptCreate('sample_iris_table',
        'def sample_iris_table(size):
            from sklearn.datasets import load_iris
            import pandas as pd
            iris = load_iris()
            x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                            "Sepal_Width","Petal_Length","Petal_Width"])
            y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                                    2: "virginica"}[x], iris.target)),\
                            columns = ["Species"])
            return pd.concat([y, x], axis=1).sample(int(size))',
        FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Create the `SAMPLE_IRIS` table in the database as the result of a `SELECT` statement, which invokes the `pyqEval` function on the `sample_iris_table` user-defined Python function saved in the script repository with the same name. The `sample_iris_table` function returns an iris data sample of size `size`.

```
CREATE TABLE sample_iris AS
SELECT *
    FROM TABLE(pyqEval(
                '{"size":20}',
                '{"Species":"varchar2(10)","Sepal_Length":"number",
                "Sepal_Width":"number","Petal_Length":"number",
                "Petal_Width":"number"}',
                'sample_iris_table'));
```

Define the Python function `predict_model` and store it with the name `linregrPredict` in the script repository. The function predicts the data in `dat` with the Python model specified by the `modelName` argument, which is loaded from the datastore specified by the `datastoreName` argument. The function also plots the actual petal width values with the predicted values. The predictions are finally concatenated and returned with `dat` as the object that the function returns.

```
BEGIN
    sys.pyqScriptCreate('linregrPredict',
        'def predict_model(dat, modelName, datastoreName):
            import oml
            import pandas as pd
            import matplotlib.pyplot as plt
            # Load model from the datastore
            objs = oml.ds.load(name=datastoreName, to_globals=False)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
            # Make predictions
            pred = objs[modelName].predict(dat[["Sepal_Length",
"Sepal_Width", "Petal_Length"]])
            # Add predictions to the DataFrame
            dat = pd.concat([dat, pd.DataFrame(pred,
columns=["Pred_Petal_Width"])], axis=1)
            dat_sorted = dat.sort_values(by="Petal_Length")
            # Create the plot
            plt.plot(dat_sorted["Petal_Length"],
dat_sorted["Pred_Petal_Width"])
            plt.title("Prediction of Petal Width")
            plt.xlabel("Petal Length")
            plt.ylabel("Predicted Petal Width")
            plt.show()
            return dat',
        FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
```

Run a `SELECT` statement that invokes the `pyqRowEval` function, which runs the specified Python function on each chunk of rows in the specified data set.

The `INP_NAM` argument specifies the data in the `SAMPLE_IRIS` table to pass to the Python function.

The `PAR_LST` argument specifies passing the input data as a pandas. DataFrame with the special control argument `oml_input_type`, along with values for the function arguments `modelName` and `datastoreName`.

In the `OUT_FMT` argument, the JSON string specifies the column names and data types of the structured table output.

The `ROW_NUM` argument specifies that five rows are included in each invocation of the function specified by `SCR_NAME`.

The `SCR_NAME` parameter specifies `linregrPredict`, which is the name in the script repository of the user-defined Python function to invoke.

```
SELECT *
    FROM table(pyqRowEval(
        inp_nam => 'SAMPLE_IRIS',
        par_lst => '{"oml_input_type":"pandas.DataFrame",
                     "modelName":"linregr", "datastoreName":"pymodel"}',
        out_fmt => '{"Species":"varchar2(12)", "Petal_Length":"number",
"Pred_Petal_Width":"number"}',
        row_num => 5,
        scr_name => 'linregrPredict'));
```

The output is the following.

```
Species      Petal_Length   Pred_Petal_Width
setosa                1.2       0.0653133202
versicolor            4.5        1.632087234
setosa                1.3       0.2420812759
setosa                1.9       0.5181904241
setosa                1.4       0.2162518989
setosa                1.4       0.1732424372
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
setosa                  1.5       0.2510460971
setosa                  1.3       0.1907951829
versicolor              3.9       1.1999981051
versicolor              4.2       1.4017887483
versicolor                4       1.2332360562
versicolor              4.8        1.765473067
virginica               5.6       2.0095892178
versicolor              4.7       1.5824801232

Species      Petal_Length   Pred_Petal_Width
virginica               5.4       2.0623088225
versicolor              4.7       1.6524411804
virginica               5.6       1.9919751044
virginica               5.8       2.1206308288
virginica               5.1       1.7983383572
versicolor              4.4       1.3677441077


20 rows selected.
```

Run a `SELECT` statement that invokes the `pyqRowEval` function and return the XML output. Each invocation of script `linregrPredict` is applied to 10 rows of data in the `SAMPLE_IRIS` table. The XML output is a CLOB; you can call `set long [length]` to get more output.

```
set long 300
SELECT *
    FROM table(pyqRowEval(
        inp_nam => 'SAMPLE_IRIS',
        par_lst => '{"oml_input_type":"pandas.DataFrame",
                     "modelName":"linregr", "datastoreName":"pymodel",
"oml_parallel_flag":true, "oml_service_level":"MEDIUM"}',
        out_fmt => 'XML',
        row_num => 10,
        scr_name => 'linregrPredict'));
```

The output is the following:

```
NAME VALUE
       <root><pandas_dataFrame><ROW-pandas_dataFrame><Species>setosa</
Species><Sepal_Length>5</Sepal_Length><Sepal_Width>3.2</
Sepal_Width><Petal_Length>1.2</Petal_Length><Petal_Width>0.2</
Petal_Width><Pred_Petal_Width>0.0653133201897007</Pred_Petal_Width></ROW-
pandas_dataFrame><ROW-pandas_dataFrame><Species>
```

14.7.2.5 pyqGroupEval Function (Autonomous AI Database)

The function `pyqGroupEval` when used in Oracle Autonomous AI Database, groups data by one or more columns and runs a user-defined Python function on each group.

The user-defined Python function can return a `boolean`, a `dict`, a `float`, an `int`, a `list`, a `str`, a `tuple` or a `pandas.DataFrame` object. Define the form of the returned value with the `OUT_FMT` parameter.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Syntax

```
FUNCTION PYQSYS.pyqGroupEval(
    INP_NAM    VARCHAR2,
    PAR_LST    VARCHAR2,
    OUT_FMT    VARCHAR2,
    GRP_COL    VARCHAR2,
    ORD_COL    VARCHAR2,
    SCR_NAME   VARCHAR2,
    SCR_OWNER  VARCHAR2 DEFAULT NULL,
    ENV_NAME   VARCHAR2 DEFAULT NULL
    )
    RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
INP_NAM	The name of a table or view that specifies the data to pass to the Python function specified by the SCR_NAME parameter. If using a table or view owned by another user, use the format <owner name>.<table/view name>. You must have read access to the specified table or view.
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: '{"oml_input_type":"pandas.DataFrame"}' **See also:** Special Control Arguments (Autonomous AI Database).
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. • The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. **See also:** Output Formats (Autonomous AI Database).
GRP_COL	The names of the grouping columns by which to partition the data. Use commas to separate multiple columns. For example, to group by GENDER and YEAR: "GENDER,YEAR"
ORD_COL	Comma-separated column names to order the input data. For example to order by GENDER: "GENDER" If specified, the input data will first be ordered by the ORD_COL columns and then grouped by the GRP_COL columns.
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

This example calls the pyqGroupEval function, which runs the specified Python script on each partition of data in the specified data set.

The INP_NAM argument specifies the data in the IRIS table to pass to the Python function.

The PAR_LST argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

The OUT_FMT arguments specifies returning a table with BLOB containing the images generated by the Python function.

The GRP_COL argument specifies to group the specified data by the 'Species' column.

The SCR_NAME parameter specifies the 'test_seaborn_inp' script, which is created in pyqTableEval Function (Autonomous AI Database).

The ENV_NAME parameter specifies 'seaborn', which is a Conda environment created in pyqEval Function (Autonomous AI Database) .

```
select *
  from table(pyqGroupEval(
        inp_nam =>  'IRIS',
        par_lst => '{"oml_graphics_flag":true}',
        out_fmt => 'PNG',
        grp_col => 'Species',
        ord_col => NULL,
        scr_name => 'test_seaborn_inp',
        scr_owner => NULL,
        env_name => 'seaborn'
));
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
GROUP_setosa
         1
"hello world"

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
Iris plot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000 00F6100000F6101
A8
3FA76900006AC649444154789CEDDD797854E5DD3EF0FBCC9EC92CD9F7B02624EC088A804B4440
05
AAD2AAB5D4166DD5F7ADDAB75A972AD60D375C706B2D2E588BED5B6A5FFD556AA91B5551145490
45

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
IMAGE
--------------------------------------------------------------------------------
--

GROUP_versicolor
         1

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
"hello world"
Iris plot
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000 00F6100000F6101
A8
3FA76900009E9149444154789CECDD77785CE5993FFCEF2973CE548D7AB124F76E6C03A69966AA
C1

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
B0244E42360B01872CC96F43CC2659922C980D9B42884902A9FB420229904D1CD22064E90EC1A6
19

GROUP_virginica

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
         1
"hello world"
Iris plot

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000 00F6100000F6101
A8

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
3FA7690000838E49444154789CEDDD797854E5D906F07BF6996496EC7B2010F64D14C1065450C1
05
8A625B6B2D0A56ED82B8B756B12AA245D4AAD5B61FA8B8606BA9AD56DC91E2120465DFF73D0442
42
```

Example

This example uses the IRIS table created in the example shown in pyqEval Function (Autonomous AI Database).

Define the Python function `group_count` and store it with the name `mygroupcount` in the script repository. The function returns a `pandas.DataFrame` generated on each group of data `dat`. The function also plots the sepal length with the petal length values on each group.

```
BEGIN
    sys.pyqScriptCreate('mygroupcount',
        'def group_count(dat):
        import pandas as pd
        import matplotlib.pyplot as plt
        plt.plot(dat[["Sepal_Length"]], dat[["Petal_Length"]], ".")
        plt.xlabel("Sepal Length")
        plt.ylabel("Petal Length")
        plt.title("{}".format(dat["Species"][0]))
        return pd.DataFrame([(dat["Species"][0], dat.shape[0])],\
        columns = ["Species", "CNT"]) ',
        FALSE, TRUE); -- V_GLOBAL, V_OVERWRITE
END;
/
```

Issue a query that invokes the `pyqGroupEval` function. In the function, the `INP_NAM` argument specifies the data in the IRIS table to pass to the function.

The `PAR_LST` argument specifies the special control argument `oml_input_type`.

The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the table returned by `pyqGroupEval`.

The `GRP_COL` parameter specifies the column to group by.

The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `mygroupcount` in the script repository.

```
SELECT *
    FROM table(
        pyqGroupEval(
            inp_nam => 'IRIS',
            par_lst => '{"oml_input_type":"pandas.DataFrame"}',
            out_fmt => '{"Species":"varchar2(10)", "CNT":"number"}',
            grp_col => 'Species',
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
            ord_col => NULL,
            scr_name => 'mygroupcount'));
```

The output is the following:

```
Species CNT
---------- ----------
virginica 50
setosa 50
versicolor 50
3 rows selected.
```

Run the same script with IRIS data and return the XML output. The `PAR_LST` argument specifies the special control argument `oml_graphics_flag` to capture images rendered in the script. Both structured data and images are included in the XML output. The XML output is a CLOB; you can call `set long [length]` to get more output.

```
set long 300
SELECT *
    FROM table(
        pyqGroupEval(
            inp_nam => 'IRIS',
            par_lst => '{"oml_input_type":"pandas.DataFrame",
"oml_graphics_flag":true, "oml_parallel_flag":true",
"oml_service_level":"MEDIUM"}',
            out_fmt => 'XML',
            grp_col => 'Species',
            ord_col => NULL,
            scr_name => 'mygroupcount'));
```

The output is the following.

```
NAME VALUE
virginica <root><Py-data><pandas_dataFrame><ROW-
pandas_dataFrame><Species>virginica</Species><CNT>50</CNT></ROW-
pandas_dataFrame></pandas_dataFrame></Py-data><images><image><img
src="data:image/pngbase64"><!
[CDATA[iVBORw0KGgoAAAANSUhEUgAAAoAAAAHgCAYAAAA10dzkAAAAOXRFWHRTb2Z0d2FyZQBNYXR
wbG90bGliIHZlcnNpb24zLjMu
setosa <root><Py-data><pandas_dataFrame><ROW-
pandas_dataFrame><Species>setosa</Species><CNT>50</CNT></ROW-
pandas_dataFrame></pandas_dataFrame></Py-data><images><image><img
src="data:image/pngbase64"><!
[CDATA[iVBORw0KGgoAAAANSUhEUgAAAoAAAAHgCAYAAAA10dzkAAAAOXRFWHRTb2Z0d2FyZQBNYXR
wbG90bGliIHZlcnNpb24zLjMuMyw
versicolor <root><Py-data><pandas_dataFrame><ROW-
pandas_dataFrame><Species>versicolor</Species><CNT>50</CNT></ROW-
pandas_dataFrame></pandas_dataFrame></Py-data><images><image><img
src="data:image/pngbase64"><!
[CDATA[iVBORw0KGgoAAAANSUhEUgAAAoAAAAHgCAYAAAA10dzkAAAAOXRFWHRTb2Z0d2FyZQBNYXR
wbG90bGliIHZlcnNpb24zLjM
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Run the same script with IRIS data and get the PNG output. The `PAR_LST` argument specifies the special control argument `oml_graphics_flag` to capture images.

```
column name format a7
column value format a15
column title format a16
column image format a15
SELECT *
    FROM table(
        pyqGroupEval(
            inp_nam => 'IRIS',
            par_lst => '{"oml_input_type":"pandas.DataFrame",
"oml_graphics_flag":true}',
            out_fmt => 'PNG',
            grp_col => 'Species',
            ord_col => NULL,
            scr_name => 'mygroupcount'));
```

The output is the following:

```
NAME            ID VALUE           TITLE            IMAGE
------- ---------- --------------- ---------------- ---------------
GROUP_s          1 [{"Species":"se setosa           89504E470D0A1A0
etosa               tosa","CNT":50}                 A0000000D494844
                    ]                               520000028000000
                                                    1E0080600000035
                                                    D1DCE4000000397
                                                    4455874536F6674
                                                    77617265004D617
                                                    4706C6F746C6962
                                                    2076657273696F6
                                                    E332E332E332C20
                                                    6874747073

NAME            ID VALUE           TITLE            IMAGE
------- ---------- --------------- ---------------- ---------------

GROUP_v          1 [{"Species":"ve versicolor       89504E470D0A1A0
ersicol             rsicolor","CNT"                 A0000000D494844
or                  :50}]                           520000028000000
                                                    1E0080600000035
                                                    D1DCE4000000397
                                                    4455874536F6674
                                                    77617265004D617
                                                    4706C6F746C6962
                                                    2076657273696F6
                                                    E332E332E332C20

NAME            ID VALUE           TITLE            IMAGE
------- ---------- --------------- ---------------- ---------------
                                                    6874747073

GROUP_v          1 [{"Species":"vi virginica        89504E470D0A1A0
irginic             rginica","CNT":                 A0000000D494844
a                   50}]                            520000028000000
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
1E0080600000035
D1DCE4000000397
4455874536F6674
77617265004D617
4706C6F746C6962
2076657273696F6
```

14.7.2.6 pyqIndexEval Function (Autonomous AI Database)

The function `pyqIndexEval` when used in Oracle Autonomous AI Database, runs a user-defined Python function multiple times as required in the Python engines spawned by the database environment.

Syntax

```
FUNCTION PYQSYS.pyqIndexEval(
    PAR_LST VARCHAR2,
    OUT_FMT VARCHAR2,
    TIMES_NUM NUMBER,
    SCR_NAME VARCHAR2,
    SCR_OWNER VARCHAR2 DEFAULT NULL,
    ENV_NAME VARCHAR2 DEFAULT NULL
)
RETURN SYS.AnyDataSet
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameters

Parameter	Description
PAR_LST	A JSON string that contains additional parameters to pass to the user-defined Python function specified by the SCR_NAME parameter. Special control arguments, which start with oml_, are not passed to the function specified by

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	SCR_NAME, but instead control what happens before or after the invocation of the function. For example, to specify the input data type as pandas.DataFrame, use: `'{"oml_input_type":"pandas.DataFrame"}'` **See also:** Special Control Arguments

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	(Autonomous AI Database).

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
OUT_FMT	The format of the output returned by the function. It can be one of the following: • A JSON string that specifies the column nam

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	esanddatatypesofthetableretur nedbythefunction.Anyimageda

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	tais discarded. The Python function must return a pandas.DataFra

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	me, anumpy.ndarray, atuple, oralistoftuples. The string 'JSON

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	', which specifies that the table returned contains a CLOB that is a

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	JSONstring. The string 'XML', which specifies that the table returnu

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	rnedcontainsaCLOBthatisanXMLstring.TheXMLcancontainbothS

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	tructureddataandimages,withhstructuredorsemi-structuredPy

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	thonobjectsfirst, followedbytheimageorimagesgeneratedbyth

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	• ePythonfunction. The string 'PNG', which specifies that the table

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	returned contains a BLOB that has the image or images generated by

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	the Python function. Images are returned as base64 encoding of th

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
	ePNG representation. **See also:** Output Formats (Autonomous AI Database).
TIMES_NUM	The number of times to execute the Python script.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
SCR_NAME	The name of a user-defined Python function in the OML4Py script repository.
SCR_OWNER	The owner of the registered Python script. The default value is NULL. If NULL, will search for the Python script in the user's script repository.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
ENV_NAME	The name of the conda environment that should be used when running the named user-defined Python function.

Example

This example defines a Python function to use with Conda environment.

Use the following code to create the 'test_seaborn_idx' script:

```
begin
  sys.pyqScriptCreate('test_seaborn_idx',
  'def fun_tab(idx):
  import seaborn as sns
  import matplotlib.pyplot as plt
  import numpy as np
  import pandas as pd
  data = np.random.multivariate_normal([0, 0], [[5, 2], [2, 2]], size=2000)
  data = pd.DataFrame(data, columns=["x", "y"])
  sns.displot(data["x"])
  plt.title("Title {}".format(idx))
  plt.show()
  return idx
  ',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/
```

This example calls the pyqIndexEval function, which runs the specified Python function multiple times.

Copyright © 2024, 2026, Oracle and/or its affiliates.

The PAR_LST argument specifies capturing images rendered in the script with the special control argument oml_graphics_flag.

The OUT_FMT arguments specifies returning a table with BLOB containing the images generated by the Python function.

The TIMES_NUM argument specifies to run the specified script 2 times.

The SCR_NAME parameter specifies the 'test_seaborn_idx' script as the Python function to invoke.

The ENV_NAME parameter specifies 'seaborn', which is a Conda environment created in pyqEval Function (Autonomous AI Database) .

```
select *
  from table(pyqIndexEval(
        par_lst => '{"oml_graphics_flag":true}',
        out_fmt => 'PNG',
        times_num => 2,
        scr_name => 'test_seaborn_idx',
        scr_owner => NULL,
        env_name => 'seaborn'
));
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
TIME_1
         1
1

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--------------------------------------------------------------------------------
--
Title 1
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58
74536F667477617265004D6174706C6F746C6962207665727369 6F6E332E332E332C2068747470
73

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000 00F6100000F6101
A8
3FA769000066674944154789CEDDD797C5355FE3FFE579236E9BEB7495BBA52A0AC2D1428C505
94
7E2DA0A3082AA0332083B80C30424747F1A720CE5254441C65649C11706340E68338A283426551
28

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--

TIME_2
         1

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
2
Title 2
89504E470D0A1A0A0000000D4948445200000280000001E0080600000035D1DCE4000000397445
58

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
74536F667477617265004D6174706C6F746C69622076657273696F6E332E332E332C2068747470
73
3A2F2F6D6174706C6F746C69622E6F72672FC897B79C0000000970485973000000F6100000F6101
A8
3FA769000068764944415478 9CEDDD79785355FE3FF0F74DDAA47BBA6FD0D2859DB216A8451495
7E

NAME
--------------------------------------------------------------------------------
--
        ID
----------
VALUE
--------------------------------------------------------------------------------
--
TITLE
--------------------------------------------------------------------------------
--
IMAGE
--------------------------------------------------------------------------------
--
2DC2A8082A220E8A88CA80A3561DADBF19709919500171D491D1914505451CF7A55A2A204BD95A
CA
```

Example

Define the Python function `fit_lm` and store it with the name `myFitMultiple` in the script repository. The function returns a `pandas.DataFrame` containing the index and prediction score of the fitted model on the data sampled from `scikit-learn`'s IRIS dataset.

```
begin
    sys.pyqScriptCreate('myFitMultiple',
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
        'def fit_lm(i, sample_size):
            from sklearn import linear_model
            from sklearn.datasets import load_iris
            import pandas as pd

            import random
            random.seed(10)

            iris = load_iris()
            x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                        "Sepal_Width","Petal_Length","Petal_Width"])
            y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                                    2: "virginica"}[x], iris.target)),\
                         columns = ["Species"])
            dat = pd.concat([y, x], axis=1).sample(sample_size)
            regr = linear_model.LinearRegression()
            regr.fit(x.loc[:, ["Sepal_Length", "Sepal_Width", \
                               "Petal_Length"]],
                     x.loc[:,["Petal_Width"]])
            sc = regr.score(dat.loc[:, ["Sepal_Length", "Sepal_Width", \
                               "Petal_Length"]],
                            dat.loc[:,["Petal_Width"]])
            return pd.DataFrame([[i,sc]],columns=["id","score"])
        ',FALSE,TRUE); -- V_GLOBAL, V_OVERWRITE
end;
/
```

Issue a query that invokes the `pyqIndexEval` function. In the function, the `PAR_LST` argument specifies the function argument `sample_size`. The `OUT_FMT` argument specifies a JSON string that contains the column names and data types of the table returned by `pyqIndexEval`. The `TIMES_NUM` parameter specifies the number of times to execute the script. The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `myFitMultiple` in the script repository.

```
select *
    from table(pyqIndexEval(
        par_lst => '{"sample_size":80,
                     "oml_parallel_flag":true",
"oml_service_level":"MEDIUM"}',
        out_fmt => '{"id":"number","score":"number"}',
        times_num => 3,
        scr_name => 'myFitMultiple'));
```

The output is the following:

```
        id score
---------- ----------
         1 .943550631
         2 .927836941
         3 .937196049
3 rows selected.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

14.7.2.7 pyqListAllJobs Function

Use the `pyqListAllJobs` function to return the list of jobs submitted.

You can query the history of all embedded Python execution jobs submitted. `pyqListAllJobs()` retrieves job creation timestamps, unique IDs, status, and elapsed time to monitor, audit, or troubleshoot runs.

Syntax

```
select * from pyqListAllJobs();
```

Results

Returns a rowset with the following columns:

Column Name	Data Type	Description
`createdDate`	TIMESTAMP(9) WITH TIME ZONE	Job submission timestamp
`job_id`	VARCHAR2(3000)	Unique Job Id
`status`	VARCHAR2(10)	Job state (for example: `RUNNING`, `FINISH`, `ERROR`, `TIMEOUT`)
`elapsedTime`	INTERVAL DAY(9) TO SECOND(9)	Total execution interval

Example

The following example shows a `pyqJobResult` call and its output.

```
SQL> select * from pyqListAllJobs();


createdDate
---------------------------------------------------------------------------
job_id
--------------------------------------------------------------------------------
--
status
----------
elapsedTime
---------------------------------------------------------------------------
07-JAN-26 07.21.01.000000000 PM UTC
OML-EBPY-691b7220-64f8-489e-bc73-b6acbeafa8e5-2537b107-
LOW-1767813661224_2026-01
-07-07-21-01
FINISH
+000000000 00:00:05.000000000


1 row selected.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

14.7.2.8 pyqDeleteJob Function

Use the `pyqDeleteJob` function to delete a job.

You can delete a specific job using its unique job ID by using the `pyqDeleteJob()` function.

Syntax

```
FUNCTION pyqDeleteJob(
   job_id  VARCHAR2
)
```

Returns these columns: `NAME`(Status message such as example "job deleted successfully"), `VALUE` (Additional value (typically empty on success)).

Parameters

`job_id`: Job ID of the job that you are trying to delete.

Example

The following example shows a `pyqDeleteJob` call and its output.

```
SELECT * FROM pyqDeleteJob('<job_id>');


NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
job deleted successfully

1 row selected.
```

14.7.2.9 pyqGrant Function (Autonomous AI Database)

This topic describes the `pyqGrant` function when used in Oracle Autonomous AI Database.

The `pyqGrant` function grants read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```
pyqGrant (
    V_NAME          VARCHAR2    IN
    V_TYPE          VARCHAR2    IN
    V_USER          VARCHAR2    IN     DEFAULT)
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameters

Parameter	Description
V_NAME	The name of an OML4Py datastore or a script in the OML4Py script repository.
V_TYPE	For a datastore, the type is `datastore`; for script the type is `pyqScript`.
V_USER	The name of the user to whom to grant access.

Example 14-31 Granting Read Access to a script

```
-- Grant read privilege access to Scott.
BEGIN
  pyqGrant('pyqFun1', 'pyqscript', 'SCOTT');
END;
/
```

Example 14-32 Granting Read Access to a datastore

```
-- Grant read privilege access to datastore ds1 to SCOTT.
BEGIN
  pyqGrant('ds1', 'datastore', 'SCOTT');
END;
/
```

Example 14-33 Granting Read Access to a Script to all Users

```
-- Grant read privilege access to script RandomRedDots to all users.
BEGIN
  pyqGrant('pyqFun1', 'pyqscript', NULL);
END;
/
```

Example 14-34 Granting Read Access to a datastore to all Users

```
-- Grant read privilege access to datastore ds1 to all users.
BEGIN
  pyqGrant('ds1', 'datastore', NULL);
END;
/
```

14.7.2.10 pyqRevoke Function (Autonomous AI Database)

This topic describes the `pyqRevoke` function when used in Oracle Autonomous AI Database.

The `pyqRevoke` function revokes read privilege access to an OML4Py datastore or to a script in the OML4Py script repository.

Syntax

```
pyqRevoke (
    V_NAME          VARCHAR2    IN
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
    V_TYPE          VARCHAR2    IN
    V_USER          VARCHAR2    IN     DEFAULT)
```

Parameters

Parameter	Description
V_NAME	The name of an OML4Py datastore or a script in the OML4Py script repository.
V_TYPE	For a datastore, the type is `datastore`; for script the type is `pyqScript`.
V_USER	The name of the user from whom to revoke access.

Example 14-35 Revoking Read Access to a script

```
-- Revoke read privilege access to script pyqFun1 from SCOTT.
BEGIN
  pyqRevoke('pyqFun1', 'pyqscript', 'SCOTT');
END;
/
```

Example 14-36 Revoking Read Access to a datastore

```
-- Revoke read privilege access to datastore ds1 from SCOTT.
BEGIN
  pyqRevoke('ds1', 'datastore', 'SCOTT');
END;
/
```

Example 14-37 Revoking Read Access to a script from all Users

```
-- Revoke read privilege access to script pyqFun1 from all users.
BEGIN
  pyqRevoke('pyqFun1', 'pyqscript', NULL);
END;
/
```

Example 14-38 Revoking Read Access to a datastore from all Users

```
-- Revoke read privilege access to datastore ds1 from all users.
BEGIN
  pyqRevoke('ds1', 'datastore', NULL);
END;
/
```

14.7.2.11 pyqScriptCreate Procedure (Autonomous AI Database)

This topic describes the `pyqScriptCreate` procedure in Oracle Autonomous AI Database. Use the `pyqScriptCreate` procedure to create a user-defined Python function and add it to the OML4Py script repository.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Syntax

```
sys.pyqScriptCreate (
    V_NAME          VARCHAR2    IN
    V_SCRIPT        CLOB        IN
    V_GLOBAL        BOOLEAN     IN     DEFAULT
    V_OVERWRITE     BOOLEAN     IN     DEFAULT)
```

Parameter	Description
V_NAME	A name for the user-defined Python function in the OML4Py script repository.
V_SCRIPT	The definition of the Python function.
V_GLOBAL	TRUE specifies that the user-defined Python function is public; FALSE specifies that the user-defined Python function is private.
V_OVERWRITE	If the script repository already has a user-defined Python function with the same name as V_NAME, then TRUE replaces the content of that user-defined Python function with V_SCRIPT and FALSE does not replace it.

Example 14-39 Using the pyqScriptCreate Procedure

This example creates a private user-defined Python function named pyqFun2 in the OML4Py script repository.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
       import numpy as np
       import pickle
       z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
       [float(x)/10 for x in range(10)],
       [x for x in range(10)],
       [bool(x%2) for x in range(10)],
       [pickle.dumps(x) for x in range(10)],
       ["test"+str(x**2) for x in range(10)])],
       dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
       ("e", "S20"), ("f", "O")])
       return z');
END;
/
```

This example creates a global user-defined Python function named pyqFun2 in the script repository and overwrites any existing user-defined Python function of the same name.

```
BEGIN
  sys.pyqScriptCreate('pyqFun2',
    'def return_frame():
       import numpy as np
       import pickle
       z = np.array([y for y in zip([str(x)+"demo" for x in range(10)],
       [float(x)/10 for x in range(10)],
       [x for x in range(10)],
       [bool(x%2) for x in range(10)],
       [pickle.dumps(x) for x in range(10)],
       ["test"+str(x**2) for x in range(10)])],
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
       dtype=[("a", "U10"), ("b", "f8"), ("c", "i4"), ("d", "?"),
       ("e", "S20"), ("f", "O")])
       return z',
       TRUE,  -- Make the user-defined Python function global.
       TRUE); -- Overwrite any global user-defined Python function
              -- with the same name.
END;
/
```

This example creates a private user-defined Python function named `create_iris_table` in the script repository.

```
BEGIN
  sys.pyqScriptCreate('create_iris_table',
    'def create_iris_table():
       from sklearn.datasets import load_iris
       import pandas as pd
       iris = load_iris()
       x = pd.DataFrame(iris.data, columns = ["Sepal_Length",\
                        "Sepal_Width","Petal_Length","Petal_Width"])
       y = pd.DataFrame(list(map(lambda x: {0:"setosa", 1: "versicolor",\
                                 2: "virginica"}[x], iris.target)),\
                        columns = ["Species"])
       return pd.concat([y, x], axis=1)');
END;
/
```

Display the user-defined Python functions owned by the current user.

```
SELECT * from USER_PYQ_SCRIPTS;
```

```
NAME              SCRIPT
----------------
-----------------------------------------------------------------------
create_iris_table  def create_iris_table():       from sklearn.datasets
import load_iris ...
pyqFun2           def return_frame():       import numpy as np       import
pickle      ...
```

Display the user-defined Python functions available to the current user.

```
SELECT * from ALL_PYQ_SCRIPTS;
```

```
OWNER     NAME              SCRIPT
--------   ----------------
-----------------------------------------------------------------------
OML_USER   create_iris_table  "def create_iris_table():  from
sklearn.datasets import load_iris ...
OML_USER   pyqFun2            "def return_frame():  import numpy as np
import pickle      ...
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
PYQSYS     pyqFun2           "def return_frame():  import numpy as np
import pickle      ...
```

14.7.2.12 pyqScriptDrop Procedure (Autonomous AI Database)

This topic describes the `pyqScriptDrop` procedure in Oracle Autonomous AI Database. Use the `pyqScriptDrop` procedure to remove a user-defined Python function from the OML4Py script repository.

Syntax

```
sys.pyqScriptDrop (
    V_NAME          VARCHAR2    IN
    V_GLOBAL        BOOLEAN     IN     DEFAULT
    V_SILENT        BOOLEAN     IN     DEFAULT)
```

Parameter	Description
V_NAME	A name for the user-defined Python function in the OML4Py script repository.
V_GLOBAL	A BOOLEAN that specifies whether the user-defined Python function to drop is a global or a private user-defined Python function. The default value is FALSE, which indicates a private user-defined Python function. TRUE specifies that the user-defined Python function is public.
V_SILENT	A BOOLEAN that specifies whether to display an error message when `sys.pyqScriptDrop` encounters an error in dropping the specified user-defined Python function. The default value is FALSE.

Example 14-40 Using the sys.pyqScriptDrop Procedure

This example drops the private user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
  sys.pyqScriptDrop('pyqFun2');
 END;
 /
```

This example drops the global user-defined Python function `pyqFun2` from the script repository.

```
BEGIN
  sys.pyqScriptDrop('pyqFun2', TRUE);
 END;
 /
```

14.7.3 Asynchronous Jobs (Autonomous AI Database)

When a function is run asynchronously, it's run as a job which can be tracked by using the `pyqJobStatus` and `pyqJobResult` functions.

Asynchronous Jobs is described in the following topics.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Topics:

- oml_async_flag Argument
  The special control argument `oml_async_flag` determines if a job is run synchronously or asynchronously. The default value is false.
- pyqJobStatus Function
  Use the `pyqJobStatus` function to look up the status of an asynchronous job. If the job is pending, it returns `job is still running` . If the job is completed, the function returns a URL.
- pyqJobResult Function
  Use the `pyqJobResult` function to return the job result.
- Asynchronous Job Example
  The following examples shows how to submit asynchronous jobs with non-XML output and with XML output.

14.7.3.1 oml_async_flag Argument

The special control argument `oml_async_flag` determines if a job is run synchronously or asynchronously. The default value is false.

Set the `oml_async_flag` Argument

- To run a function in synchronous mode, set `oml_async_flag` to `false`.

  In synchronous mode, the SQL API waits for the HTTP call to finish and returns when the HTTP response is ready.

  By default, `pyq*Eval` functions are executed synchronously. The default connection timeout limit is 60 seconds. Synchronous mode is used if `oml_async_flag` is not set or if it's set to `false`.
- To run a function in asynchronous mode, set `oml_async_flag` to `true`.

  In asynchronous mode, the SQL API returns a URL directly after the asynchronous job is submitted to the web server. The URL contains a job ID, which can be used to fetch the job status and result in subsequent SQL calls.

Submit Asynchronous Job Example

This example uses the table `GRADE`, created as follows:

```
CREATE TABLE GRADE (
NAME VARCHAR2(30),
GENDER VARCHAR2(1),
STATUS NUMBER(10),
YEAR NUMBER(10),
SECTION VARCHAR2(1),
SCORE NUMBER(10),
FINALGRADE NUMBER(10)
);

insert into GRADE values('Abbott', 'F', 2, 97, 'A', 90, 87);
insert into GRADE values('Branford', 'M', 1, 98, 'A', 92, 97);
insert into GRADE values('Crandell', 'M', 2, 98, 'B', 81, 71);
insert into GRADE values('Dennison', 'M', 1, 97, 'A', 85, 72);
insert into GRADE values('Edgar', 'F', 1, 98, 'B', 89, 80);
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
insert into GRADE values('Faust', 'M', 1, 97, 'B', 78, 73);
insert into GRADE values('Greeley', 'F', 2, 97, 'A', 82, 91);
insert into GRADE values('Hart', 'F', 1, 98, 'B', 84, 80);
insert into GRADE values('Isley', 'M', 2, 97, 'A', 88, 86);
insert into GRADE values('Jasper', 'M', 1, 97, 'B', 91, 83);
```

In the following code, the Python function `score_diff` is defined and stored with the name `computeGradeDiff` as a private function in the script repository. The function returns a `pandas.DataFrame` after assigning a new `DIFF` column by computing the difference between the `SCORE` and `FINALGRADE` column of the input data.

```
begin
    sys.pyqScriptCreate('computeGradeDiff','def score_diff(dat):
        import numpy as np
        import pandas as pd
        df = dat.assign(DIFF=dat.SCORE-dat.FINALGRADE)
        return df
    ');
end;
/
```

Run the saved `computeGradeDiff` script as follows:

```
select *
   from table(pyqTableEval(
         inp_nam => 'GRADE',
         par_lst => '{"oml_async_flag":true}',
         out_fmt => NULL,
         scr_name => 'computeGradeDiff',
         scr_owner => NULL
   ));
```

The `VALUE` column of the result contains a URL containing the job ID of the asynchronous job:

```
NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v2/jobs/<job id>
1 row selected.
```

14.7.3.2 pyqJobStatus Function

Use the `pyqJobStatus` function to look up the status of an asynchronous job. If the job is pending, it returns `job is still running` . If the job is completed, the function returns a URL.

Syntax

```
FUNCTION PYQSYS.pyqJobStatus(
  job_id       VARCHAR2
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
)
RETURN PYQSYS.pyqClobSet
```

Parameters

Parameter	Description
job_id	The ID of the asynchronous job.

Example

The following example shows a pyqJobStatus call and its output.

```
SQL> select * from pyqJobStatus(
       job_id => '<job id>'
);


NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v2/jobs/<job id>/
result

1 row selected.
```

14.7.3.3 pyqJobResult Function

Use the pyqJobResult function to return the job result.

> **Note**
>
> The asynchronously job results are stored in a table within the central CDB system and are accessible for up to 6 months.

Syntax

```
FUNCTION PYQSYS.pyqJobResult(
  job_id       VARCHAR2,
  out_fmt      VARCHAR2 DEFAULT 'JSON'
)
RETURN SYS.AnyDataSet
```

Parameters

Parameter	Description
job_id	The ID of the asynchronous job.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Parameter	Description
out_fmt	The format of the job result. It can be one of the following: • A JSON string that specifies the column names and data types of the table returned by the function. Any image data is discarded. The Python function must return a pandas.DataFrame, a numpy.ndarray, a tuple, or a list of tuples. • The string 'JSON', which specifies that the table returned contains a CLOB that is a JSON string. • The string 'XML', which specifies that the table returned contains a CLOB that is an XML string. The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. • The string 'PNG', which specifies that the table returned contains a BLOB that has the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation.

Example

The following example shows a `pyqJobResult` call and its output.

```
SQL> select * from pyqJobResult(
    job_id => '<job id>',
    out_fmt =>
'{"NAME":"varchar2(7)","SCORE":"number","FINALGRADE":"number","DIFF":"number"}
'
);


NAME       SCORE      FINALGRADE DIFF
---------- ---------- ---------- ----------
Abbott     90         87          3
Branford   92         97         -5
Crandell   81         71         10
Dennison   85         72         13
Edgar      89         80          9
Faust      78         73          5
Greeley    82         91         -9
Hart       84         80          4
Isley      88         86          2
Jasper     91         83          8

10 rows selected.
```

14.7.3.4 Asynchronous Job Example

The following examples shows how to submit asynchronous jobs with non-XML output and with XML output.

Non-XML Output

When submitting asynchronous jobs, for JSON, PNG and relational outputs, set the `OUT_FMT` argument to `NULL` when submitting the job. When fetching the job result, specify `OUT_FMT` in the `pyqJobResult` call.

Copyright © 2024, 2026, Oracle and/or its affiliates.

This example uses the IRIS table created in the example shown in the pyqTableEval Function (Autonomous AI Database) topic and the `linregrPredict` script created in the example shown in the pyqRowEval Function (Autonomous AI Database) topic.

Issue a `pyqGroupEval` function call to submit an asynchronous job. In the function, the `INP_NAM` argument specifies the data in the IRIS table to pass to the function.

The `PAR_LST` argument specifies submitting the job asynchronously with the special control argument `oml_async_flag`, capturing the images rendered in the script with the special control argument `oml_graphics_flag`, passing the input data as a `pandas.DataFrame` with the special control argument `oml_input_type`, along with values for the function arguments `modelName` and `datastoreName`.

The `OUT_FMT` argument is `NULL`.

The `GRP_COL` parameter specifies the column to group by.

The `SCR_NAME` parameter specifies the user-defined Python function stored with the name `linregrPredict` in the script repository.

The asynchronous call returns a job status URL in CLOB, you can call `set long [length]` to get the full URL.

```
set long 150
  select *
  from table(pyqGroupEval(
  inp_nam => 'IRIS',
  par_lst => '{"oml_input_type":"pandas.DataFrame",
  "oml_async_flag":true, "oml_graphics_flag":true,
              "modelName":"linregr", "datastoreName":"pymodel"}',
  out_fmt => NULL,
  grp_col => 'Species',
  ord_col => NULL,
  scr_name => 'linregrPredict',
  scr_owner => NULL
));
```

The output is the following:

```
NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v2/jobs/<job id>

1 row selected.
```

Run a `SELECT` statement that invokes the `pyqJobStatus` function, which returns a resource URL containing the job ID when the job result is ready.

```
select * from pyqJobStatus(
job_id => '<job id>');
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

The output is the following when the job is still pending.

```
NAME
--------------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------------
job is still running
1 row selected.
```

The output is the following when the job finishes.

```
NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--

https://<oml-cloud-service-location-url>/oml/api/py-scripts/v2/jobs/<job id>/
result

1 row selected.
```

Run a `SELECT` statement that invokes the `pyqJobResult` function.

In the `OUT_FMT` argument, the string `'PNG'` specifies to include both return value and images (titles and image bytes) in the result.

```
column name format a7
column value format a15
column title format a16
column image format a15
select * from pyqJobResult(
        job_id => '<job id>',
        out_fmt => 'PNG'
);
```

The output is the following.

```
NAME            ID VALUE           TITLE            IMAGE
------- ---------- --------------- ---------------- ---------------
GROUP_s          1 [{"Species":"se Prediction of Pe 6956424F5277304
etosa              tosa","Sepal_Le tal Width        B47676F41414141
                   ngth":4.6,"Sepa                  4E5355684555674
                   l_Width":3.6,"P                  141416F41414141
                   etal_Length":1.                  486743415941414
                   0,"Petal_Width"                  1413130647A6B41
                   :0.2,"Pred_Peta                  41414142484E435
                   l_Width":0.1325                  356514943416749
                   345443},{"Speci                  6641686B6941414
                   es":"setosa","S                  141416C7753466C
                                                    7A4141415059514
                                                    141443245427144
                                                    2B6E61514141414
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                                                    468305256683055
                                                    32396D644864686
                                                    36D554162574630
                                                    634778766447787
                                                    0596942325A584A
                                                    7A615739754D793
                                                    4784C6A49734947

GROUP_v         1 [{"Species":"ve Prediction of Pe 6956424F5277304
ersicol           rsicolor","Sepa tal Width        B47676F41414141
or                l_Length":5.1,"                  4E5355684555674
                  Sepal_Width":2.                  141416F41414141
                  5,"Petal_Length                  486743415941414
                  ":3.0,"Petal_Wi                  1413130647A6B41
                  dth":1.1,"Pred_                  41414142484E435
                  Petal_Width":0.                  356514943416749
                  8319563387},{"S                  6641686B6941414
                  pecies":"versic                  141416C7753466C
                                                   7A4141415059514
                                                   141443245427144
                                                   2B6E61514141414
                                                   468305256683055
                                                   32396D644864686
                                                   36D554162574630
                                                   634778766447787
                                                   0596942325A584A
                                                   7A615739754D793
                                                   4784C6A49734947

GROUP_v         1 [{"Species":"vi Prediction of Pe 6956424F5277304
irginic           rginica","Sepal tal Width        B47676F41414141
a                  _Length":5.7,"S                 4E5355684555674
                  epal_Width":2.5                  141416F41414141
                  ,"Petal_Length"                  486743415941414
                  :5.0,"Petal_Wid                  1413130647A6B41
                  th":2.0,"Pred_P                  41414142484E435
                  etal_Width":1.7                  356514943416749
                  55762924},{"Spe                  6641686B6941414
                  cies":"virginic                  141416C7753466C
                                                   7A4141415059514
                                                   141443245427144
                                                   2B6E61514141414
                                                   468305256683055
                                                   32396D644864686
                                                   36D554162574630
                                                   634778766447787
                                                   0596942325A584A
                                                   7A615739754D793
                                                   4784C6A49734947

3 rows selected.
```

XML Ouput

If XML output is expected from the asynchronous job, set the `OUT_FMT` argument to `'XML'` when submitting the job and fetching the job result.

Copyright © 2024, 2026, Oracle and/or its affiliates.

This example uses the script `myFitMultiple` created in the example shown in the pyqIndexEval Function (Autonomous AI Database) topic.

Issue a `pyqIndexEval` function call to submit an asynchronous job. In the function, the `PAR_LST` argument specifies submitting the job asynchronously with the special control argument `oml_async_flag`, along with values for the function arguments `sample_size`.

The asynchronous call returns a job status URL in CLOB, you can call `set long [length]` to get the full URL.

```
set long 150 select *
  from table(pyqIndexEval(
        par_lst => '{"sample_size":80,"oml_async_flag":true}',
        out_fmt => 'XML',
        times_num => 3,
        scr_name => 'myFitMultiple',
        scr_owner => NULL
));
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v2/jobs/<job id>

1 row selected.
```

Run a `SELECT` statement that invokes the `pyqJobStatus` function, which returns a resource URL containing the job id when the job result is ready.

```
select * from pyqJobStatus(
job_id => '<job id>'
);
```

The output is the following when the job is still pending.

```
NAME
--------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------
job is still running
1 row selected.
```

The output is the following when the job result is ready.

```
NAME
--------------------------------------------------------------------------------
--
VALUE
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
--------------------------------------------------------------------------------
--
https://<oml-cloud-service-location-url>/oml/api/py-scripts/v2/jobs/<job id>/
result

1 row selected.
```

Run a `SELECT` statement that invokes the `pyqJobResult` function.

In the `OUT_FMT` argument, the string `'XML'` specifies that the table returned contains a CLOB that is an XML string.

```
select * from pyqJobResult(
         job_id => '<job id>',
         out_fmt => 'XML'
);
```

The output is the following.

```
NAME
--------------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------------
1
<root><pandas_dataFrame><ROW-pandas_dataFrame><id>1</id><score>0.94355
0631313753</score></ROW-pandas_dataFrame></pandas_dataFrame></root>

2
<root><pandas_dataFrame><ROW-pandas_dataFrame><id>2</id><score>0.92783
6941437123</score></ROW-pandas_dataFrame></pandas_dataFrame></root>

3
<root><pandas_dataFrame><ROW-pandas_dataFrame><id>3</id><score>0.93719
6049031545</score></ROW-pandas_dataFrame></pandas_dataFrame></root>


3 rows selected.
```

14.7.4 Special Control Arguments (Autonomous AI Database)

Use the `PAR_LST` parameter to specify special control arguments and additional arguments to be passed into the Python script.

Argument	Syntax and Description
`oml_input_type`	**Syntax** **`oml_input_type`** : **`'pandas.DataFrame'`**, **`'numpy.recarray'`**, or **`'default'`** (default) **Description** Specifies the type of object to construct from data in the Autonomous AI Database. By default, a two-dimensional `numpy.ndarray` of type `numpy.float64` is constructed if all columns are numeric. Otherwise, a `pandas.DataFrame` is constructed.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Argument	Syntax and Description
`oml_na_omit`	**Syntax** **`oml_na_omit : bool, false`** (default) **Description** Determines if rows with any missing values will be omitted from the table to be evaluated. If `true`, omit all rows with missing values from the table. If `false`, do not omit rows with missing values from the table.
`oml_async_flag`	**Syntax** **`oml_async_flag`**: **`bool`**, **`false`** (default) **Description** If `true`, the job will be submitted asynchronously. If `false`, the job will be executed in synchronous mode.
`oml_graphics_flag`	**Syntax** **`oml_graphics_flag`**: **`bool`**, **`false`** (default) **Description** If `true`, the server will capture images rendered in the Python script. If `false`, the server will not capture images rendered in the Python script.
`oml_parallel_flag`	**Syntax** **`oml_parallel_flag`**: **`bool`**, **`false`** (default) **Description** If `true`, the Python script will be run with data parallelism. Data parallelism is only applicable to `pyqRowEval`, `pyqGroupEval`, and `pyqIndexEval`. If `false`, the Python script will not be run with data parallelism.
`oml_service_level`	**Syntax** **`oml_service_level`** : **`string`**, allowed values: **`'LOW'`**(default), **`'MEDIUM'`**, **`'HIGH'`** **Description** Controls the different levels of performance and concurrency in Autonomous AI Database.
`timeout`	**Syntax** **`timeout`** : **`int, 1800`** (default) **Description** The asynchronous timeout limit, expressed in seconds, must be used with the argument `oml_async_flag` set to `true`. Allowed values are in the range [1800, 43200].

Examples

- Input data is `pandas.DataFrame`:

```
par_lst => '{"oml_input_type":"pandas.DataFrame"}'
```

- Drop rows with missing values from input data:

```
par_lst => '{"oml_na_omit":true}'
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

- Submit a job in asynchronous mode:

  ```
  par_lst => '{"oml_async_flag":true}'
  ```

- Use `MEDIUM` service level:

  ```
  par_lst => '{"oml_service_level":"MEDIUM"}'
  ```

- Use `timeout` argument:

  ```
  par_lst => '{"oml_async_flag":true, "timeout": 4000}'
  ```

14.7.5 Output Formats (Autonomous AI Database)

The `OUT_FMT` parameter controls the format of output returned by the table functions `pyqEval`, `pyqGroupEval`, `pyqIndexEval`, `pyqRowEval`, `pyqTableEval`, and `pyqJobResult`.

The output formats are:

- JSON
- Relational
- XML
- PNG
- Asynchronous Mode Output

JSON

When `OUT_FMT` is set to `JSON`, the table functions return a table containing a CLOB that is a JSON string.

The following example invokes the `pyqEval` function on the `'pyqFun1'` created in the `pyqEval` function section.

```
SQL> select *
    from table(pyqEval(
        par_lst => '{"oml_service_level":"MEDIUM"}',
        out_fmt => 'JSON',
        scr_name => 'pyqFun1'));


NAME
--------------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------------
[{"FLOAT":0,"ID":0,"NAME":"demo_0"},{"FLOAT":0.1,"ID":1,"NAME":"demo_1
"},{"FLOAT":0.2,"ID":2,"NAME":"demo_2"},{"FLOAT":0.3,"ID":3,"NAME":"de
mo_3"},{"FLOAT":0.4,"ID":4,"NAME":"demo_4"},{"FLOAT":0.5,"ID":5,"NAME"
:"demo_5"},{"FLOAT":0.6,"ID":6,"NAME":"demo_6"},{"FLOAT":0.7,"ID":7,"N
AME":"demo_7"},{"FLOAT":0.8,"ID":8,"NAME":"demo_8"},{"FLOAT":0.9,"ID":
9,"NAME":"demo_9"}]

1 row selected.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Relational

When `OUT_FMT` is specified with a JSON string where column names are mapped to column types, the table functions return the response by reshaping it into table columns.

For example, if `OUT_FMT` is specified with `{"NAME":"varchar2(7)", "DIFF":"number"}`, the output should contain a `NAME` column of type `VARCHAR2(7)` and a `DIFF` column of type `NUMBER`. The following example uses the table `GRADE` and the script `'computeGradeDiff'` (created in Asynchronous Jobs (Autonomous AI Database)) and invokes the `computeGradeDiff` function:

```
SQL> select *
  from table(pyqTableEval(
        inp_nam => 'GRADE',
        par_lst => '{"oml_input_type":"pandas.DataFrame"}',
        out_fmt => '{"NAME":"varchar2(7)","DIFF":"number"}',
        scr_name => 'computeGradeDiff'));


NAME DIFF
------- ----------
Abbott   3
Branfor -5
Crandel 10
Denniso 13
Edgar    9
Faust    5
Greeley -9
Hart     4
Isley    2
Jasper   8

10 rows selected.
```

XML

When `OUT_FMT` is specified with `XML`, the table functions return the response in a table with fixed columns. The output consists of two columns. The `NAME` column contains the name of the row. The `NAME` column value is `NULL` for `pyqEval`, `pyqTableEval`,`pyqRowEval` function returns. For `pyqGroupEval`, `pyqIndexEval`, the `NAME` column value is the group/index name. The `VALUE` column contains the XML string.

The XML can contain both structured data and images, with structured or semi-structured Python objects first, followed by the image or images generated by the Python function. Images are returned as a base 64 encoding of the PNG representation. To include images in the XML string, the special control argument `oml_graphics_flag` must be set to true.

In the following code, the python function `gen_two_images` is defined and stored with name `plotTwoImages` in the script repository. The function renders two subplots with random dots in red and blue color and returns the number of columns of the input data.

```
begin
        sys.pyqScriptCreate('plotTwoImages','def gen_two_images (dat):
              import numpy as np
              import matplotlib.pyplot as plt
              np.random.seed(22)
              fig = plt.figure(1);
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                fig2 = plt.figure(2);
                ax = fig.add_subplot(111);
                ax.set_title("Random red dots")
                ax2 = fig2.add_subplot(111);
                ax2.set_title("Random blue dots")
                ax.plot(range(100), np.random.normal(size=100), marker = "o",
                        color = "red", markersize = 2)
                ax2.plot(range(100,0,-1), marker = "o", color = "blue",
markersize = 2)
                return dat.shape[1]
       ',FALSE,TRUE);
end;
/
```

The following example shows the XML output of a `pyqRowEval` function call where both structured data and images are included in the result:

```
SQL> select *
    from table(pyqRowEval(
         inp_nam => 'GRADE',
         par_lst => '{"oml_graphics_flag":true}',
         out_fmt => 'XML',
         row_num => 5,
         scr_name => 'plotTwoImages'
));


NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
1
<root><Py-data><int>7</int></Py-data><images><image><img src="data:ima
ge/pngbase64"><![CDATA[iVBORw0KGgoAAAANSUhEUgAAAoAAAAHgCAYAAAA10dzkAAA
ABHNCSVQICAgIfAhkiAAAAAlwSFlzAAAPYQAAD2EBqD+naQAAADh0RVh0U29mdHdhcmUAb
WF0cGxvdGxpYiB2ZXJzaW9uMy4xLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9yZy8li6FKAAA
gAElEQVR4nOydeZwcVb32n6ruqu6e6Z6ZTGYmmSyTBAIJCQkkQAgBAgQIIIuAgIgKKIqKXhXv
lVdFRb2Kr+vrdkVFRVFBRBBkCQQIEAgJ2ffJMpNMZp+eXqq7uqu7uqreP6qrpqq7eqZnMpPM
JOf7+fCBqa6uOlV1zu93fs/vOb/DqKqqgkKhUCgUCoVy2sCe6gZQKBQKhUKhUE4u1ACkUCgU
CoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBS
KBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkG
NQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK
5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAo
FAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACk
UCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMM
agBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgU
ymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQ
KBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFI
oVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ
1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAo
lNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKh
UCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQ
QqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4z
qAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQ
KKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVC
oVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYg
hUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxm
UAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKh
UE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqF
QqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1A
CoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnN
oAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVC
oZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQK
hUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqA
FAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKa
QQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqF
QjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigU
CoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUA
KRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0
gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQK
hXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAo
FAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoA
UigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMpp
BjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgU
CuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQ
KBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQA
pFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTT
DGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAo
FMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKh
UCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gB
SKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCin
GdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQ
KJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVC
oVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlAD
kEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBO
M6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKh
UCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqF
QqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAG
IIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGc
ZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVC
oVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQK
hUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkEN
QAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5
zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqF
QqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkU
CoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMa
gBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVy
mkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQK
hUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIo
FAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1
ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArl
NIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgU
CoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQ
KBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxq
AFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTK
aQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAo
FArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUih
UCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnU
AKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU
0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQ
KBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BC
oVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOo
AUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAo
pxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKh
UCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCF
QqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQ
A5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQ
TjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVC
oVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAK
hUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2g
BiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKh
nGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqF
QqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAU
CoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppB
DUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVC
Oc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQK
hUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQAp
FAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSD
GoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqF
cppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgU
CoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBS
KBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkG
NQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK
5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAo
FAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACk
UCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMM
agBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgU
ymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQ
KBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFI
oVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ
1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAo
lNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKh
UCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQ
QqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4z
qAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQ
KKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVC
oVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYg
hUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxm
UAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKh
UE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqF
QqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1A
CoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnN
oAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVC
oZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQK
hUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqA
FAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKa
QQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqF
QjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigU
CoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUA
KRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0
gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQK
hXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAo
FAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoA
UigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMpp
BjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgU
CuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQ
KBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQA
pFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTT
DGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAo
FMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKh
UCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gB
SKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCin
GdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQ
KJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVC
oVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlAD
kEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBO
M6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKh
UCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqF
QqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAG
IIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGc
ZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVC
oVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQK
hUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkEN
QAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5
zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqF
QqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkU
CoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMa
gBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVy
mkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQK
hUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIo
FAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1
ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArl
NIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgU
CoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQ
KBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxq
AFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTK
aQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAo
FArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUih
UCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnU
AKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU
0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQ
KBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BC
oVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOo
AUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAo
pxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKh
UCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCF
QqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQ
A5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQ
TjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVC
oVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppBDUAK
hUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2g
BiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQKhUKh
nGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQApFAqF
QqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSDGoAU
CoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqFcppB
DUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgUCoVC
Oc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBSKBQK
hUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkGNQAp
FAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK5TSD
GoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAoFAqF
cppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACkUCgU
CoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMMagBS
KBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgUymkG
NQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQKBQK
5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFIoVAo
FAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ1ACk
UCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAolNMM
agBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKhUCgU
ymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQQqFQ
KBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4zqAFI
oVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQKKcZ
1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVCoVAo
lNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYghUKh
UCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxmUAOQ
QqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKhUE4z
qAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqFQqFQ
KKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1ACoVC
oVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnNoAYg
hUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVCoZxm
UAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQKhUKh
UE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqAFAqF
QqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKaQQ1A
CoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqFQjnN
oAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigUCoVC
oZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUAKRQK
hUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0gxqA
FAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQKhXKa
QQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAoFAqF
QjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoAUigU
CoVCoZxmUAOQQqFQKBQK5TSDGoAUCoVCoVAopxnUAKRQKBQKhUI5zaAGIIVCoVAoFMppBjUA
KRQKhUKhUE4zqAFIoVAoFAqFcppBDUAKhUKhUCiU0wxqAFIoFAqFQqGcZlADkEKhUCgUCuU0
gxqAFAqFQqFQKKcZ1ACkUCgUCoVCOc2gBiCFQqFQKBTKaQY1ACkUCoVCoVBOM6gBSKFQKBQK
hXKaQQ1ACoVCoVAolNMMagBSKBQKhUKhnGZQA5BCoVAoFArlNIMagBQKhUKhUCinGdQApFAo
FAqFQjnNoAYghUKhUCgUymkGNQApFAqFQqFQTjOoAUihUCgUCoVymkENQAqFQqFQKJTTDGoA
UigU
```

The PNG output consists of four columns. The `NAME` column contains the name of the row. The `NAME` column value is `NULL` for `pyqEval` and `pyqTableEval` function returns. For `pyqRowEval`, `pyqGroupEval`, `pyqIndexEval`, the `NAME` column value is the chunk/group/index name. The `ID` column indicates the ID of the image. The `VALUE` column contains the return value of the executed script. The `TITLE` column contains the titles of the rendered PNG images. The `IMAGE` column is a BLOB column containing the bytes of the `PNG` images rendered by the executed script.

The following example shows the PNG output of a `pyqRowEval` function call.

```
SQL> column name format a7
column valueformat a5
column title format a16
column image format a15
select *
from table(pyqRowEval(
    inp_nam => 'GRADE',
    par_lst => '{"oml_graphics_flag":true}',
    out_fmt => 'PNG',row_num => 5,
    scr_name => 'plotTwoImages',
    scr_owner =>NULL
));


NAME            ID VALUE TITLE            IMAGE
------- --------- ----- ---------------- ---------------
CHUNK_1          1 7     Random red dots  6956424F5277304
                                          B47676F41414141
                                          4E5355684555674
                                          141416F41414141
                                          486743415941414
                                          1413130647A6B41
                                          41414142484E435
                                          356514943416749
                                          6641686B6941414
                                          141416C7753466C
                                          7A41414150

CHUNK_1          2 7     Random blue dots 6956424F5277304
                                          B47676F41414141
                                          4E5355684555674
                                          141416F41414141
                                          486743415941414
                                          1413130647A6B41
                                          41414142484E435
                                          356514943416749
                                          6641686B6941414
                                          141416C7753466C
                                          7A41414150

CHUNK_2          1 7     Random red dots  6956424F5277304
                                          B47676F41414141
                                          4E5355684555674
                                          141416F41414141
                                          486743415941414
                                          1413130647A6B41
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                                        41414142484E435
                                        356514943416749
                                        6641686B6941414
                                        141416C7753466C
                                        7A41414150

CHUNK_2          2 7     Random blue dots 6956424F5277304
                                        B47676F41414141
                                        4E5355684555674
                                        141416F41414141
                                        486743415941414
                                        1413130647A6B41
                                        41414142484E435
                                        356514943416749
                                        6641686B6941414
                                        141416C7753466C
                                        7A41414150

4 rows selected.
```

Asynchronous Mode Output

When you set `oml_async_flag` to `true` to run an asynchronous job, set `OUT_FMT` to `NULL` for jobs that return non-XML results, or set it to XML for jobs that return XML results, as described below.

See also oml_async_flag Argument.

Asynchronous Mode: Non-XML Output

When submitting asynchronous jobs, for JSON, PNG, and relational outputs, set `OUT_FMT` to `NULL` when submitting the job. When fetching the job result, specify `OUT_FMT` in the `pyqJobResult` call.

The following example shows how to get the JSON output from an asynchronous `pyqIndexEval` function call:

```
SQL> select *
    from table(pyqGroupEval(
        inp_nam => 'GRADE',
        par_lst => '{"oml_async_flag":true, "oml_graphics_flag":true}',
        out_fmt => NULL,
        grp_col => 'GENDER',
        ord_col => NULL,
        scr_name => 'inp_twoimgs',
        scr_owner => NULL
    ));


NAME
--------------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------------

https://<host name>/oml/tenants/<tenant name>/databases/<database
name>/api/py-scripts/v2/jobs/<job id>
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
1 row selected.


SQL> select * from pyqJobStatus(
        job_id => '<job id>');


NAME
--------------------------------------------------------------------------------
VALUE
--------------------------------------------------------------------------------

https://<host name>/oml/tenants/<tenant name>/databases/<database
 name>/api/py-scripts/v2/jobs/<job id>/result

1 row selected.


SQL> column name format a7
column value format a5
column title format a16
column image format a15
select * from pyqJobResult(
     job_id => '<job id>',
     out_fmt => 'PNG'
     );


NAME            ID VALUE TITLE            IMAGE
------- ---------- ----- ---------------- ---------------
GROUP_F          1 7     Random red dots  6956424F5277304
                                          B47676F41414141
                                          4E5355684555674
                                          141416F41414141
                                          486743415941414
                                          1413130647A6B41
                                          41414142484E435
                                          356514943416749
                                          6641686B6941414
                                          141416C7753466C
                                          7A4141415059514
                                          141443245427144
                                          2B6E61514141414
                                          468305256683055
                                          32396D644864686
                                          36D554162574630
                                          634778766447787
                                          0596942325A584A
                                          7A615739754D793
                                          4784C6A49734947

GROUP_F          2 7     Random blue dots 6956424F5277304
                                          B47676F41414141
                                          4E5355684555674
                                          141416F41414141
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                                                  486743415941414
                                                  1413130647A6B41
                                                  41414142484E435
                                                  356514943416749
                                                  6641686B6941414
                                                  141416C7753466C
                                                  7A4141415059514
                                                  141443245427144
                                                  2B6E61514141414
                                                  468305256683055
                                                  32396D644864686
                                                  36D554162574630
                                                  634778766447787
                                                  0596942325A584A
                                                  7A615739754D793
                                                  4784C6A49734947

GROUP_M          1 7     Random red dots          6956424F5277304
                                                  B47676F41414141
                                                  4E5355684555674
                                                  141416F41414141
                                                  486743415941414
                                                  1413130647A6B41
                                                  41414142484E435
                                                  356514943416749
                                                  6641686B6941414
                                                  141416C7753466C
                                                  7A4141415059514
                                                  141443245427144
                                                  2B6E61514141414
                                                  468305256683855
                                                  32396D644864686
                                                  36D554162574630
                                                  634778766447787
                                                  0596942325A584A
                                                  7A615739754D793
                                                  4784C6A49734947

GROUP_M          2 7     Random blue dots         6956424F5277304
                                                  B47676F41414141
                                                  4E5355684555674
                                                  141416F41414141
                                                  486743415941414
                                                  1413130647A6B41
                                                  41414142484E435
                                                  356514943416749
                                                  6641686B6941414
                                                  141416C7753466C
                                                  7A4141415059514
                                                  141443245427144
                                                  2B6E61514141414
                                                  468305256683055
                                                  32396D644864686
                                                  36D554162574630
                                                  634778766447787
                                                  0596942325A584A
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
                                                7A615739754D793
                                                4784C6A49734947

4 rows selected
```

Asynchronous Mode: XML Output

If XML output is expected from the asynchronous job, you must set `OUT_FMT` to `XML` when submitting the job and fetching the job result.

The following example shows how to get the XML output from an asynchronous `pyqIndexEval` function call.

```
SQL> select *
    from table(pyqIndexEval(
        par_lst => '{"oml_async_flag":true}',
        out_fmt => 'XML',
        times_num => 3,
        scr_name => 'idx_ret_df',
        scr_owner => NULL
));


NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
https://<host name>/oml/tenants/<tenant name>/databases/<database
name>/api/py-scripts/v2/jobs/<job id>

1 row selected.


SQL> select * from pyqJobStatus(
    job_id => '<job id>'
);


  2
NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--

https://<host name>/oml/tenants/<tenant name>/databases/<database
name>/api/py-scripts/v2/jobs/<job id>/result

1 row selected.


SQL> select * from pyqJobResult(
        job_id => '<job id>',
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
        out_fmt => 'XML'
);


  2    3    4
NAME
--------------------------------------------------------------------------------
--
VALUE
--------------------------------------------------------------------------------
--
1
<root><pandas_dataFrame><ROW-
pandas_dataFrame><ID>1</ID><RES>a</RES></ROW-pandas
_dataFrame></pandas_dataFrame></root>

2
<root><pandas_dataFrame><ROW-
pandas_dataFrame><ID>2</ID><RES>b</RES></ROW-pandas
_dataFrame></pandas_dataFrame></ro

3
<root><pandas_dataFrame><ROW-
pandas_dataFrame><ID>3</ID><RES>c</RES></ROW-pandas
_dataFrame></pandas_dataFrame></root>

3 rows selected
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

15

Manage System Processes in OML4Py Using psutil

If you find that your Python process is consuming too many of your machine's resources, or causing your machine to crash, you can get information about, or set limits for, the resources Python is using.

The Python system and process utilities library `psutil` is a cross-platform library for retrieving information on running processes and system utilization, such as CPU, memory, disks, network, and sensors, in Python. It is useful for system monitoring, profiling, limiting process resources, and the management of running processes.

The function `psutil.Process.rlimit` gets or sets process resource limits. In `psutil`, process resource limits are constants with names beginning with `psutil.RLIMIT_`. Each resource is controlled by a soft limit and hard limit tuple.

For example, `psutil.RLIMIT_AS` represents the maximum size (in bytes) of the virtual memory (address space) used by the process. The default limit of `psutil.RLIMIT_AS` can be -1 (`psutil.RLIM_INFINITY`). You can lower the resource limit of `psutil.RLIMIT_AS` to prevent your Python program from loading too much data into memory, as shown in the following example.

Example 15-1 Resource Control with psutil.RLIMIT_AS

```
import psutil
import numpy as np

# Get the current OS process.
p = psutil.Process()

# Get a list of available resources.
[attr for attr in dir(psutil) if attr[:7] == 'RLIMIT_']

# Display the Virtual Memory Size of the current process.
p.memory_info().vms

# Get the process resource limit RLIMIT_AS.
soft, hard = p.rlimit(psutil.RLIMIT_AS)
print('Original resource limits of RLIMIT_AS (soft/hard): {}/{}'.format(soft,
hard))

# Check the constant used to represent the limit for an unlimited resource.
psutil.RLIM_INFINITY

# Set resource RLIMIT_AS (soft, hard) limit to (1GB, 2GB).
p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*1, pow(1024,3)*2))

# Get the current resource limit of RLIMIT_AS.
cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
print('Current resource limits of RLIMIT_AS (soft/hard): {}/
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
{}'.format(cur_soft, cur_hard))

# Define a list of sizes to be allocated in MB (megabytes).
sz = [5, 10, 20]

# Define a megabyte variable in bytes.
MB = 1024*1024

# Allocate an increasing amount of data.
for val in sz:
    stmt = "Allocate %s MB " % val
    try:
        print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
        m = np.arange(val*MB/8, dtype="u8")
        print(stmt + " Success.")
    except:
        print(stmt + " Fail.")
        raise

# Delete the allocated variable.
del m

# Raise the soft limit of RLIMIT_AS to 2GB.
p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*2, pow(1024,3)*2))

# Get the current resource limit of RLIMIT_AS.
cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
print('Current resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(cur_soft, cur_hard))

# Retry: allocate an increasing amount of data.
for val in sz:
    stmt = "Allocate %s MB " % val
    try:
        print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
        m = np.arange(val*MB/8, dtype="u8")
        print(stmt + " Success.")
    except:
        print(stmt + " Fail.")
        raise
```

Listing for This Example

```
>>> import psutil
>>> import numpy as np
>>>
>>> # Get the current OS process.
... p = psutil.Process()
>>>
>>> # Get a list of available resources.
... [attr for attr in dir(psutil) if attr[:7] == 'RLIMIT_']
['RLIMIT_AS', 'RLIMIT_CORE', 'RLIMIT_CPU', 'RLIMIT_DATA',
 'RLIMIT_FSIZE', 'RLIMIT_LOCKS', 'RLIMIT_MEMLOCK', 'RLIMIT_MSGQUEUE',
 'RLIMIT_NICE', 'RLIMIT_NOFILE', 'RLIMIT_NPROC', 'RLIMIT_RSS',
 'RLIMIT_RTPRIO', 'RLIMIT_RTTIME', 'RLIMIT_SIGPENDING', 'RLIMIT_STACK']
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
>>>
>>> # Display the Virtual Memory Size of the current process.
... p.memory_info().vms
413175808
>>>
>>> # Get the process resource limit RLIMIT_AS.
... soft, hard = p.rlimit(psutil.RLIMIT_AS)
>>> print('Original resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(soft, hard))
Original resource limits of RLIMIT_AS (soft/hard): -1/-1
>>>
>>> # Check the constant used to represent the limit for an unlimited
resource.
... psutil.RLIM_INFINITY
-1
>>>
>>> # Set the resource RLIMIT_AS (soft, hard) limit to (1GB, 2GB).
... p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*1, pow(1024,3)*2))
>>>
>>> # Get the current resource limit of RLIMIT_AS.
... cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
>>> print('Current resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(cur_soft, cur_hard))
Current resource limits of RLIMIT_AS (soft/hard): 1073741824/2147483648
>>>
>>> # Define a list of sizes to be allocated in MB (megabytes).
... sz = [100, 200, 500, 1000]
>>>
>>> # Define a megabyte variable in bytes.
... MB = 1024*1024
>>>
>>> # Allocate an increasing amount of data.
... for val in sz:
...     stmt = "Allocate %s MB " % val
...     try:
...         print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
...         m = np.arange(val*MB/8, dtype="u8")
...         print(stmt + " Success.")
...     except:
...         print(stmt + " Fail.")
...         raise
...
virtual memory: 394 MB
Allocate 100 MB Success.
virtual memory: 494 MB
Allocate 200 MB Success.
virtual memory: 594 MB
Allocate 500 MB Fail.
Traceback (most recent call last):
  File "<stdin>", line 6, in <module>
MemoryError
>>>
>>> # Delete the allocated variable.
... del m
>>>
>>> # Raise the soft limit of RLIMIT_AS to 2GB.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
... p.rlimit(psutil.RLIMIT_AS, (pow(1024,3)*2, pow(1024,3)*2))
>>>
>>> # Get the current resource limit of RLIMIT_AS.
... cur_soft, cur_hard = p.rlimit(psutil.RLIMIT_AS)
>>> print('Current resource limits of RLIMIT_AS (soft/hard): {}/
{}'.format(cur_soft, cur_hard))
Current resource limits of RLIMIT_AS (soft/hard): 2147483648/2147483648
>>>
>>> # Retry: allocate an increasing amount of data.
... for val in sz:
...     stmt = "Allocate %s MB " % val
...     try:
...         print("virtual memory: %d MB" % int(p.memory_info().vms/MB))
...         m = np.arange(val*MB/8, dtype="u8")
...         print(stmt + " Success.")
...     except:
...         print(stmt + " Fail.")
...         raise
...
virtual memory: 458 MB
Allocate 100 MB Success.
virtual memory: 558 MB
Allocate 200 MB Success.
virtual memory: 658 MB
Allocate 500 MB Success.
virtual memory: 958 MB
Allocate 1000 MB Success.
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

16

Python Classes to Convert Pretrained Models to ONNX Models (Deprecated)

This topic provides the functions, attributes and example usage of the deprecated python classes `EmbeddingModelConfig` and `EmbeddingModel`.

> **Note**
>
> This API is updated for the database 23.7. The version used for the database 23.6 and below used python packages EmbeddingModel and EmbeddingModelConfig. These packages are replaced with ONNXPipeline and ONNXPipelineConfig respectively.
>
> Oracle recommends that you use the latest version. If a you choose to use a deprecated class, a warning message will be shown indicating that the classes will be removed in the future and advising the user to switch to the new class. You can find the details of the updated python classes in Python Classes to Convert Pretrained Models to ONNX Models.

Examples for converting pretrained text models to ONNX format using EmbeddingModel, EmbeddingModelConfig

1. To load the python classes on the OML4Py client :

   ```
   from oml.utils import EmbeddingModel, EmbeddingModelConfig
   ```

2. You can get a list of all preconfigured models by running the following:

   ```
   EmbeddingModelConfig.show_preconfigured()
   ```

3. To get a list of available templates:

   ```
   EmbeddingModelConfig.show_templates()
   ```

4. Generate an ONNX file from the preconfigured model "sentence-transformers/all-MiniLM-L6-v2":

   ```
   em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
   em.export2file("your_preconfig_file_name",output_dir=".")
   ```

5. Generate an ONNX model from the preconfigured model "sentence-transformers/all-MiniLM-L6-v2" in the database:

   ```
   em = EmbeddingModel(model_name="sentence-transformers/all-MiniLM-L6-v2")
   em.export2db("your_preconfig_model_name")
   ```


Copyright © 2024, 2026, Oracle and/or its affiliates.

6. Generate an ONNX file using the provided text template:

```
config = EmbeddingModelConfig.from_template("text",max_seq_length=512)
em = EmbeddingModel(model_name="intfloat/e5-small-v2",config=config)
em.export2file("your_template_file_name",output_dir=".")
```

Functions and Attributes of EmbeddingModelConfig

The `EmbeddingModelConfig` class contains the properties required for the package to perform downloading, exporting, augmenting, validation, and storing of an ONNX model. The class provides access to configuration properties using the dot operator. As a convenience, well-known configurations are provided as templates.

Parameters

This table describes the functions and properties of the `EmbeddingModelConfig` class.

Functions	Parameter Type	Returns	Description
`from_template(name, **kwargs)`	• `name` (`String`): The name of the template • `**kwargs`: template properties to override or add	Instance of `EmbeddingModelConfig`	A static function that creates an `EmbeddingModelConfig` object based on a predefined template given by the name parameter. You can use named arguments to override the template properties.
`show_templates()`	NA	List of existing templates	A static function that returns a list of existing templates by name.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Functions	Parameter Type	Returns	Description
`show_preconfigured()`	• `include_properties` (`bool, optional`): A flag indicating whether properties should be included in the results. Defaults to `False` so only names will be included by default. • `model_name` (`str, optional`): A model name to filter by when including properties. This argument will be ignored if `include_properties` is `False`. Otherwise only the properties of this model will be included in the results.	A list of preconfigured model names or properties.	Shows a list of preconfigured model names, or properties. By default, this function returns a list of names only. If the properties are required, pass the `include_properties` parameter as `True`. The returned list will contain a single dict where each key of the dict is the name of a preconfigured model and the value is the property set for that model. Finally, if only a single set of properties for a specific model is required, pass the name of the model in the `model_name` parameter (the `include_properties` parameter should also be `True`). This will return a list of a single dict with the properties for the specified model.

Template Properties

The text template has configuration properties shown below:

```
"do_lower_case": true,
"post_processors":[{"name":"Pooling","type":"mean"},{"name":"Normalize"}]
```

> **Note**
>
> All other properties in the Properties table will take the default values. Any property without a default value must be provided when creating the `EmbeddingModelConfig` instance.

Properties

This table shows all properties that can be configured. preconfigured models already have these properties set to specific values. Templates will use the default values unless a user overrides it when using the `from_template` function on `EmbeddingModelConfig`.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Property	Description
post_processors	An array of post_processors that will be loaded after the model is loaded or initialized. The list of known and supported post_processors is provided later in this section. Templates may define a list of post_processors for the types of models they support. Otherwise, an empty array is the default.
max_seq_length	This property is applicable for text-based models only. The maximum length of input to the model as number of tokens. There is no default value. Specify this value for models that are not preconfigured.
do_lower_case	Specifies whether or not to lowercase the input when tokenizing. The default value is True.
quantize_model	Perform quantization on the model. This could greatly reduce the size of the model as well as speed up the process. It may however result in different results for the embedding vector (against the original model) and possibly small reduction in accuracy. The default value is False.
distance_metrics	An array of names of suitable distance metrics for the model. The names must be name of distance metrics used for Oracle vector distance operator. Only used when exporting the model to the database. Supported list is ["EUCLIDEAN","COSINE","MANHATTAN","HAMMING","DOT","EUCLIDEAN_SQUARED", "JACCARD"]. The default value is an empty array.
languages	An array of language (Abbreviation) supported in the Database. Only used when exporting the model to the database. For a supported list of languages, see Languages. The default value is an empty array.
use_float16	Specifies whether or not to convert the exported onnx model to float16. The default value is False.

Properties of post_processors

This table describes the built-in post_processors and their configuration parameters.

post_processor	Parameters	Description
Pooling	• name: Pooling. • type: Valid values should be mean(Default), max, cls	The Pooling post_processor summarizes the output of the transformer model into a fixed-length vector.
Normalize	• name: Specify Normalize	The Normalize post_processor bounds the vector values to a range using L2 normalization.

Copyright © 2024, 2026, Oracle and/or its affiliates.

post_processor	Parameters	Description
`Dense`	• `name`: Dense • `in_features`: Input feature size • `out_features`: Output feature size • `bias`: Whether to learn an additive bias. The default value is `True`. • `activation_function`: Activation function of the dense layer. Currently only supports `Tanh` as the activation function.	Applies transformation to the incoming data.

Example: Configure post_processors

In this example, you override the post_processors in the sentence-transformers template with a `Max` Pooling post_processor followed by Normalization.

```
config = EmbeddingModelConfig.from_template("text")
config.post_processors = [{"name":"Pooling","type":"max"},
{"name":"Normalize"}]
```

Functions and Attributes of EmbeddingModel

Use the `EmbeddingModel` class to convert transformer models to the ONNX format with post_processing steps embedded into the final model.

Parameters

This table describes the signature and properties of the `EmbeddingModel` class.

Functions	Parameters	Description
`EmbeddingModel(model_name, configuration=None,settings={})`	• `model_name`: The name of the model to be used. For example, `medicalai/ClinicalBERT` • `configuration`: An initialized `EmbeddingModelConfig` object. This parameter must be specified when using a template. If not specified, the model will be assumed to be a preconfigured model. • `settings`: A dictionary of various settings that are global and control various operations such as logging levels and locations for files.	Creates a new instance of the `EmbeddingModel` class.

Copyright © 2024, 2026, Oracle and/or its affiliates.

Settings

The settings object is a dictionary passed to the `EmbeddingModel` class. It provides global properties for the `EmbeddingModel` class that are used for non-model-specific operations, such as logging.

Property	Default Value	Description
`cache_dir`	`$HOME/.cache/OML`	The base directory used for downloads. Model files will be downloaded from the repository to directories relative to the `cache_dir`. If the `cache_dir` does not exist at time of execution, it will be created.
`logging_level`	`ERROR`	The level for logging. Valid values are ['DEBUG', 'INFO', 'WARNING', 'ERROR', 'CRITICAL']. **Note:** This log level is also applied globally to all python packages and is also mapped to the ONNXRuntime libraries.
`force_download`	`False`	Forces download of model files instead of reloading from cache.
`ignore_checksum_error`	`False`	Ignores any errors caused by mismatch in checksums when using preconfigured models.

Functions

This table describes the function and properties of the `EmbeddingModel` class.

Function	Parameters	Description
`export2file(export_name,output_dir=None)`	• `export_name(string)`: The name of the file. The file will be saved with the file extension `.onnx` • `output_dir(string)`: An optional output directory. If not specified the file will be saved to the current directory	Exports the model to a file.
`export2db(export_name)`	• `export_name(string)`: The name that will be used for the mining model object. This name must be compliant with existing rules for object names in the database.	Exports the model to the database.

Example: Preconfigured Model

This example illustrates the preconfigured embedding model that comes with the Python package. You can use this model without any additional configurations.

```
"sentence-transformers/distiluse-base-multilingual-cased-v2": {
        "max_seq_length": 128,
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

```
        "do_lower_case": false,
        "post_processors":[{"name":"Pooling","type":"mean"},
{"name":"Dense","in_features":768, "out_features":512, "bias":true,
"activation_function":"Tanh"}],
        "quantize_model":true,
        "distance_metrics": ["COSINE"],
        "languages": ["ar", "bg", "ca", "cs", "dk", "d", "us", "el", "et",
"fa", "sf", "f", "frc", "gu", "iw", "hi", "hr", "hu", "hy", "in", "i", "ja",
"ko", "lt",
                      "lv", "mk", "mr", "ms", "n", "nl", "pl", "pt", "ptb",
"ro", "ru", "sk", "sl", "sq", "lsr", "s", "th", "tr", "uk", "ur", "vn",
"zhs", "zht"]
    }
```


Copyright © 2024, 2026, Oracle and/or its affiliates.

Glossary

Copyright © 2024, 2026, Oracle and/or its affiliates.

Index

Numerics

3rd party package, *13*
3rd party packages, *9*

A

ADMIN, *9*
algorithm selection class, *6*
algorithms
  Apriori, *22*
  attribute importance, *20*
  Automated Machine Learning, *1*
  Automatic Data Preparation, *12*
  automatically selecting, *15*
  Decision Tree, *28*
  Expectation Maximization, *35*
  Explicit Semantic Analysis, *49*
  Exponential Smoothing, *118*
  Generalized Linear Model, *55*
  k-Means, *65*
  machine learning, *2*
  Minimum Description Length, *20*
  Naive Bayes, *72*
  Neural Network, *80*
  Non-Negative Matrix Factorization, *112*
  ONNX, *90*
  Random Forest, *93*
  settings common to all, *5*
  Singular Value Decomposition, *100*
  Support Vector Machine, *106*
  XGBoost, *128*
ALL_PYQ_DATASTORE_CONTENTS view, *4*
ALL_PYQ_DATASTORES view, *5*
ALL_PYQ_SCRIPTS view, *7*
anomaly detection models, *106*
Apriori algorithm, *22*
attribute importance, *20*
auto model search
  automated model search, *4*
  automatic model search, *4*
Automated Machine Learning
  about, *1*
Automatic Data Preparation algorithm, *12*
Automatic Machine Learning
  connection parameter, *2*
Autonomous Database, *1*

C

classes
  Automated Machine Learning, *1*
  GlobalFeatureImportance, *13*
  machine learning, *2*
  oml.ai, *20*
  oml.ar, *22*
  oml.automl.AlgorithmSelection, *6*
  oml.automl.FeatureSelection, *8*
  oml.automl.ModelSelection, *15*
  oml.automl.ModelTuning, *11*
  oml.dt, *12*, *28*
  oml.em, *35*
  oml.esa, *49*
  oml.esm, *118*
  oml.glm, *55*
  oml.graphics, *41*
  oml.km, *65*
  oml.nb, *72*
  oml.nn, *80*
  oml.rf, *93*
  oml.svd, *100*
  oml.svm, *106*
  oml.xgb, *128*
  onnx, *90*
classification algorithm, *93*
classification and regression algorithm, *128*
classification and regression models, *128*
classification models, *12*, *28*, *55*, *72*, *80*, *93*, *106*
client
  installing for Linux for Autonomous Database, *1*
  installing for Linux on-premises, *22*, *46*, *79*, *99*
Clustering algorithm, *118*
clustering models, *35*, *49*, *65*
Clustering models, *118*
conda enviroment, *9*
connection
  creating a on-premises database, *4*
  functions, *2*
control arguments, *9*
convert Python to SQL, *4*

Copyright © 2024, 2026, Oracle and/or its affiliates.

creating
proxy objects, 13, 15
cx_Oracle package, 2
cx_Oracle.connect function, 2

D

data
about moving, 9
exploring, 19
filtering, 13
preparing, 1
selecting, 4
Data lineage
build_source
query used for build data, 4
data parallel processing, 9
database
connecting to an on-premises, 4
datastores
about, 20
database views for, 4–6
deleting objects, 28
describing objects in, 27
getting information about, 25
granting or revoking access to, 30
loading objects from, 24
saving objects in, 21
Date Types, 33
DCLI
Exadata, 5
python, 2
Decision Tree algorithm, 28
Distributed Command Line Interface, 2
doc2vec, 3
Download environment from object storage, 13
dropping
tables, 15

E

EM model, 35
Embedded Python Execution
about, 9
about the SQL interface for, 36
SQL interface for, 35, 56
ESA embeddings, 3
ESA model, 49
Exadata, 1
compute nodes, 2
DCLI, 5
Expectation Maximization, 4
Expectation Maximization algorithm, 35
explainability, 13
Explicit Semantic Analysis algorithm, 49
Exponential Smoothing Model, 4, 118
export
OML4Py function, 32
script function, 32
User defined function, 32
exporting models, 8

F

feature extraction algorithm, 49
feature extraction class, 100
feature selection class, 8
function
pyqGrant, 50, 113
functions
cx_Oracle.connect, 2
Embedded Python Execution, 9
for graphics, 41
for managing user-defined Python functions, 25
oml.check_embed, 2, 4
oml.connect, 2, 4
oml.create, 15
oml.cursor, 9, 15
oml.dir, 9, 13
oml.disconnect, 2, 4
oml.do_eval, 11
oml.drop, 15
oml.ds.delete, 28
oml.ds.describe, 27
oml.ds.dir, 25
oml.ds.load, 24
oml.ds.save, 21
oml.grant, 30
oml.graphics.boxplot, 41
oml.graphics.hist, 41
oml.group_apply, 15
oml.index_apply, 22
oml.isconnected, 2, 4
oml.row_apply, 19
oml.script.create, 25
oml.script.dir, 28
oml.script.drop, 31
oml.script.load, 30
oml.set_connection, 2
oml.sync, 13
oml.table_apply, 12
pyqEval, 37
pyqGroupEval, 47
pyqRowEval, 43
pyqTableEval, 40

G

GLM link functions, 3
GLM models, 55

Copyright © 2024, 2026, Oracle and/or its affiliates.

granting
access to scripts and datastores, 30
user privileges, 15, 40, 73, 93
graphics
rendering, 41

I

import
OML4Py function, 34
script function, 34
User defined function, 34
importing models, 8
improved data preparation, 4
installing
client for Linux for Autonomous Database, 1
client for Linux on-premises, 22, 46, 79, 99
server for Linux on-premises, 10, 35, 68, 88
Instant Client
installing for Linux on-premises, 21, 45, 78, 98

K

KM model, 65
KMeans, 4

L

libraries inOML4Py, 6
Linux
installing Python for, 3, 30, 62, 84
uninstalling on-premises client for, 18, 26, 27, 50, 51, 83, 103, 104
uninstalling on-premises server for, 19, 43, 75, 96
Linux for Autonomous Database
installing client for, 1
Linux on-premises
installing client for, 22, 46, 79, 99
installing Oracle Instant Client for, 21, 45, 78, 98
installing server for, 10, 35, 68, 88
supporting packages for, 6, 32, 65, 86

M

machine learning
classes, 2
methods
drop, 13
drop_duplicates, 13
dropna, 13
for exploring data, 19
for preparing data, 1
pull, 11
Minimum Description Length algorithm, 20
model selection, 15
model tuning, 11
models
association rules, 22
attribute importance, 20
Clustering, 118
Decision Tree, 12, 28
Expectation Maximization, 35
explainability, 13
Explicit Semantic Analysis, 49
exporting and importing, 8
for anomaly detection, 106
for classification, 12, 28, 55, 72, 80, 93, 106
for classification and regression, 128
for clustering, 35, 65
for Clustering, 118
for feature extraction, 49, 100
for regression, 55, 80, 106
Generalized Linear Model, 55
k-Means, 65
Naive Bayes, 72
Neural Network, 80
Non-Negative Matrix Factorization, 112
parametric, 55
persisting, 2
Random Forest, 93
Singular Value Decomposition, 100
Support Vector Machine, 106
XGBoost, 128
moving data
about, 9
to a local Python session, 11
to the database, 9

N

Naive Bayes model, 72
Neural Network model, 80
NMF models, 112

O

ODMS_BOXCOX, 4
oml_input_type argument, 9
oml_na_omit argument, 9
oml.ai class, 20
oml.ar class, 22
oml.automl.AlgorithmSelection class, 6
oml.automl.FeatureSelection class, 8
oml.automl.ModelSelection class, 15
oml.automl.ModelTuning class, 11
oml.check_embed function, 2, 4
oml.connect function, 2, 4
oml.create function, 15
oml.cursor function, 9, 15

Copyright © 2024, 2026, Oracle and/or its affiliates.

oml.Datetime, *33*
oml.dir function, *9*, *13*
oml.disconnect function, *2*, *4*
oml.do_eval function, *11*
oml.drop function, *15*
oml.ds.delete function, *28*
oml.ds.describe function, *27*
oml.ds.dir function, *25*
oml.ds.load function, *24*
oml.ds.save function, *21*
oml.dt class, *12*, *28*
oml.em class, *35*
oml.esa class, *49*
oml.esm class, *118*
oml.glm class, *55*
oml.grant function, *30*
oml.graphics class, *41*
oml.graphics.boxplot function, *41*
oml.graphics.hist function, *41*
oml.group_apply function, *15*
oml.index_apply function, *22*
oml.Integer, *33*
oml.isconnected function, *2*, *4*
oml.km class, *65*
oml.nb class, *72*
oml.nn class, *80*
oml.onnx class, *90*
oml.push function, *9*
oml.revoke function, *30*
oml.rf class, *93*
oml.row_apply function, *19*
oml.script.create function, *25*
oml.script.dir function, *28*
oml.script.drop function, *31*
oml.script.load function, *30*
oml.set_connection function, *2*, *4*
oml.svd class, *100*
oml.svm class, *106*
oml.sync function, *13*
oml.table_apply function, *12*
oml.Timedelta, *33*
oml.Timezone, *33*
oml.xgb class, *128*
OML4Py, *1*, *1*
  Exadata, *5*
on-premises client
  installing, *20*, *44*, *76*, *97*
  uninstalling, *18*, *26*, *27*, *50*, *51*, *83*, *103*, *104*
on-premises server
  installing, *9*, *34*, *66*, *87*
  uninstalling, *19*, *43*, *75*, *96*
ONNX, *33*
ONNX algorithm, *90*
ONNX model size, *33*
ONNX models, *33*
Oracle Machine Learning Notebooks, *1*
Oracle Machine Learning Python interpreter, *1*
Oracle wallets
  about, *3*
ore.nmf function, *112*

P

packages
  supporting for Linux on-premises, *6*, *32*, *65*, *86*
parallel processing, *9*
parametric models, *55*
PL/SQL procedures
  sys.pyqScriptCreate, *52*
  sys.pyqScriptDrop, *55*
predict method, *72*
predict.proba method, *72*
privileges
  required, *15*, *40*, *73*, *93*
proxy objects, *4*
  for database tables, *13*, *15*
  storing, *20*
pull method, *11*
PYQADMIN role, *15*, *40*, *73*, *93*
pyqEval function, *37*
pyqGrant function, *50*, *113*
pyqGroupEval function, *47*
pyqRowEval function, *43*
pyqTableEval function, *40*
pyquser.sql script, *16*, *41*, *74*, *94*
Python, *1*
  installing for Linux, *3*, *30*, *62*, *84*
  libraries in OML4Py, *6*
  version used, *6*
Python interpreter, *1*
Python objects
  storing, *20*
python packages, *9*
Python to SQL conversion, *4*

R

Random Forest algorithm, *93*
ranking
  attribute importance, *20*
read privilege
  granting or revoking, *30*
regression models, *55*, *80*
resources
  managing, *1*
revoking
  access to scripts and datastores, *30*
roles
  PYQADMIN, *15*, *40*, *73*, *93*

Copyright © 2024, 2026, Oracle and/or its affiliates.

S

scoring new data, *2*, *2*
script repository
  granting or revoking access to, *30*
  managing user-defined Python functions in, *25*
  registering a user-defined function, *25*
scripts
  pyquser, *16*, *41*, *74*, *94*
server
  installing for Linux on-premises, *10*, *35*, *68*, *88*
setting
  onnx, *90*
settings
  about model, *5*
  Apriori algorithm, *22*
  association rules, *22*
  Automatic data preparation algorithm, *12*
  Decision Tree algorithm, *28*
  Expectation Maximization model, *35*
  Explicit Semantic Analysis algorithm, *49*
  Exponential Smoothing Model, *118*
  Generalized Linear Model algorithm, *55*
  k-Means algorithm, *65*
  Minimum Description Length algorithm, *20*
  Naive Bayes algorithm, *72*
  Neural Network algorithm, *80*
  Random Forest algorithm, *93*
  shared algorithm, *5*
  Singular Value Decomposition algorithm, *100*
  sttribute importance, *20*
  Support Vector Machine algorithm, *106*
  XGBoost algorithm, *128*
special control arguments, *9*
SQL APIs
  pyqEval function, *37*
  pyqGrant function, *50*, *113*
  pyqGroupEval function, *47*
  pyqRowEval function, *43*
  pyqTableEval function, *40*
SQL to Python conversion, *4*
supporting packages
  for Linux on-premises, *6*, *32*, *65*, *86*
SVD model, *100*
SVM models, *106*
synchronizing database tables, *13*
sys.pyqScriptCreate procedure, *52*
sys.pyqScriptDrop procedure, *55*

T

tables
  creating, *15*
  dropping, *13*, *15*
  proxy objects for, *13*, *15*
task parallel processing, *9*
transparency layer, *4*

U

uninstalling
  on-premises client, *18*, *26*, *27*, *50*, *51*, *83*, *103*, *104*
  on-premises server, *19*, *43*, *75*, *96*
USER_PYQ_DATASTORES view, *6*
USER_PYQ_SCRIPTS view, *8*
user-defined Python functions
  Embedded Python Execution of, *9*
users
  creating new, *16*, *41*, *74*, *94*

V

views
  ALL_PYQ_DATASTORE_CONTENTS, *4*
  ALL_PYQ_DATASTORES, *5*
  ALL_PYQ_SCRIPTS, *7*
  USER_PYQ_DATASTORES, *6*
  USER_PYQ_SCRIPTS, *8*

W

wallets
  about Oracle, *3*
WINSORIZE, *4*

X

XGBoost algorithm, *128*
XGBoost constraints, *3*
XGBoost survival analysis, *3*

Copyright © 2024, 2026, Oracle and/or its affiliates.