Icon

Description automatically generatedVerticaPy Machine Learning V0.10.1 Cheat Sheet

VerticaPy Machine Learning supports the entire machine learning workflow via a Python interface. For more information about the capabilities of VerticaPy ML, see the [VerticaPy ML documentation](https://www.vertica.com/python/) or check out the [VerticaPy examples](https://www.vertica.com/python/examples/).

**Legend:** Grey text describes the function. Highlighted text represents some (and not all) of the optional parameters. Parameters are in purple. Strings are orange.

# 

## Load data [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.read_csv.html)

=> import verticapy as vp

=> VDataFrame=vp.read\_csv("filename.csv") – Creates a VDataFrame from a csv file.

## Summarize data [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.vDataFrame.describe.html)

=> VDataFrame.describe() – Aggregates the vDataFrame using multiple statistical aggregations.

=> VDataFrame.describe(columns=["column\_1", "column\_2", "column\_3", method="categorical"]) – Aggregates the selected columns using categorical statistical aggregations.

## Detect Outliers [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.vDataFrame.outliers_plot.html) and [(link)](https://www.vertica.com/python/documentation/1.0.x/html/verticapy.vDataFrame.outliers.html)

=> VDataFrame.outliers\_plot(["col1", "col2"]) – A 2D plot to visualize outliers based on the given two columns

=> VDataFrame.outliers(columns=["col1", "col2"], name="name of the outlier columns") – Create a new column which indicates whether a datapoint is an outlier.

## Measure correlations [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.vDataFrame.corr.html)

=> VDataFrame.corr(method="pearson") – Calculates and displays the Pearson correlation matrix.

=> VDataFrame.corr(["column\_1", "column\_2"], method="spearman") – Calculates and displays the Pearson correlation between two columns.

## Normalize Data [(link)](https://www.vertica.com/python/documentation/1.0.x/html/verticapy.vDataFrame.scale.html)

=> VDataFrame.scale() – Normalizes all the columns in the dataset using zscore method as default.

=> VDataFrame.scale(columns=["col1", "col2"], method="minmax") – Normalizes selective columns in the dataset using minimax method as default.

## Dimensionality Reduction [(link)](https://www.vertica.com/python/documentation/1.0.x/html/notebooks/data_prep/decomposition/index.html)

=> from verticapy.learn.decomposition import PCA – Importing PCA function.

=> model = PCA("PCA\_name") – Make a PCA object.

=> model.fit(VDataFrame) – Apply the PCA on the VDataFrame and display the results of PCA.

=> model.transform(n\_components=2) – Create a VDataFrame with columns as the principal components.

## Encode Categorical features [(link)](https://www.vertica.com/python/documentation/1.0.x/html/notebooks/data_prep/encoding/index.html)

=> VDataFrame.label\_encode() – Encodes a categorical column into numerical values.

=> VDataFrame["column\_name"].one\_hot\_encode() – One Hot Encoding for the desired column.

=> VDataFrame["column\_name"].mean\_encode() – Mean Encoding for the desired column.

## Impute missing values [(link)](https://www.vertica.com/python/documentation/1.0.x/html/notebooks/data_prep/missing_values/index.html)

=> VDataFrame.count\_percent() – Counts the percentage of missing values for reach column.

=> VDataFrame["col\_to\_fill"].fillna(method="auto") – Fills missing values by selecting mean of numeric values and mode for categorical.

=> VDataFrame["col\_to\_fill"].fillna(method="avg", by =["columns\_used\_in\_parition"]) – Fills missing values using the columns for prediction. This replaces the original column.

## Process imbalanced data [(link)](https://www.vertica.com/python/documentation/1.0.x/html/verticapy.vDataFrame.balance.html#verticapy.vDataFrame.balance)

=> VDataFrame.balance(column="column\_to\_balance") – Creates a view with an equal distribution of the input data based on response column. Default method is hybrid.

=> VDataFrame.balance(column="column\_to\_balance", method="under", x=0.5) – Creates a view with a custom distribution of the input data based on response column. Ratio(x) can be changed.

=> VDataFrame["column\_to\_balance"].topk(k=3) – Returns the count for the values in a column.

## 

**Preprocessing data**

Page 1

# 

## Sample data [(link)](https://www.vertica.com/python/documentation/1.0.x/html/verticapy.vDataFrame.sample.html)

=> VDataFrame.sample(x=0.2) – The entire table is randomly sampled using the given ratio(x).

=> VDataFrame.sample(n=100) – The entire table is randomly sampled using the number of elements required(n).

=> VDataFrame.sample(x=0.3, method="stratified") – The entire table is randomly sampled using the given ratio(x) and method (random, stratified or systematic).

**Training and predicting**

# 

# Regression – Model Building

## Linear Regression [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.linear_model.LinearRegression.html)

=> from verticapy.machine\_learning.vertica import LinearRegression – Import the Linear Regression function.

=> model = LinearRegression(name="public.Name\_of\_Model") – Build a Linear Regression model.

## Support Vector Machines (SVM) [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.svm.LinearSVR.html)

=> from verticapy.machine\_learning.vertica import LinearSVR

=> model = LinearSVR(name="Name\_of\_Model", acceptable\_error\_margin=0.5) – Build a LinearSVR object using the Vertica SVM (Support Vector Machine) algorithm.

## Random Forest [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.ensemble.RandomForestRegressor.html)

=> from verticapy.machine\_learning.vertica import RandomForestRegressor

=> model = RandomForestRegressor(name="Name\_of\_Model", n\_estimators=20, max\_features="auto", max\_leaf\_nodes=32, sample=0.7, max\_depth=3, min\_sample\_leaf=5, min\_info\_gain=0.0, nbins=32) – Creates a RandomForestRegressor object using the Vertica Random Forest function on the data.

## XGBoost [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.ensemble.XGBRegressor.html)

=> from verticapy.machine\_learning.vertica import XGBRegressor

=> model = XGBRegressor(name="Name\_of\_Model", max\_ntree=10, max\_depth=5, nbins=32, objective="squarederror", split\_proposal\_method="global", tot=0.001, learning\_rate=0.1, min\_split\_loss=0, weight\_reg=0, sample=1) – Creates a XGBoostRegressor object using the Vertica XGBoost algorithm. From all the available options, only name is mandatory.

## Autoregression [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.automl.AutoML.html)

=> from verticapy.machine\_learning.vertica import AutoML

=> model=AutoML(name="Name\_of\_Model", estimator\_type="regressor", cv=3, stepwise=True) – Tests multiple models to find which the ones which maximize the input score.

**Classification**

## Logistic Regression [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.linear_model.LogisticRegression.html)

=> from verticapy.machine\_learning.vertica import LogisitcRegression – Import the Logistic Regression function.

=> mode = LogisticRegression(name="Name\_of\_Model", penalty= "L2", tol=1e-4, C=1, max\_iter=100, solver= "CGD") – Creates a LogisticRegression object using Vertica LOGISTIC\_REG function.

## Support Vector Machines (SVM) [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.svm.LinearSVC.html)

=> from verticapy.machine\_learning.vertica import LinearSVC

=> model = LinearSVC(name="Name\_of\_Model", tol=1e-4, C=1.0, fit\_intercept= True, intercept\_model="regularized", max\_iter=100) – Build a LinearSVC object using the Vertica SVM (Support Vector Machine) algorithm.

## Random Forest [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.ensemble.RandomForestClassifier.html)

=> from verticapy.machine\_learning.vertica import RandomForestClassifier

=> model = RandomForestClassifier(name="Name\_of\_Model", n\_estimators=20, max\_features="auto", max\_leaf\_nodes=32, sample=0.7, max\_depth=3, min\_sample\_leaf=5, min\_info\_gain=0.0, nbins=32) – Creates a RandomForestRegressor object using the Vertica Random Forest function on the data.

## XGBoost [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.ensemble.XGBClassifier.html)

=> from verticapy.machine\_learning.vertica import XGBClassifier

=> model = XGBClassifier(name="Name\_of\_Model", max\_ntree=10, max\_depth=5, nbins=32, objective="squarederror", split\_proposal\_method="global", tot=0.001, learning\_rate=0.1, min\_split\_loss=0, weight\_reg=0, sample=1) – Creates a XGBoostRegressor object using the Vertica XGBoost algorithm. From all the available options, only name is mandatory.

Page 1

# 

## Autoclassification [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.automl.AutoML.html)

=> from verticapy.machine\_learning.vertica import AutoML

=> model=AutoML(name="Name\_of\_Model", estimator\_type="multi", cv=3, stepwise=True) – Tests multiple models to find which the ones which maximize the input score.

# Clustering

## K-neighbors [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.neighbors.KNeighborsClassifier.html)

=> from verticapy.machine\_learning.vertica import KNeighborsClassifier

=> model= KNeighborsClassifier(name="Name\_of\_Model", n\_neighbors=5, p=2) – Creates a KNeighborsClassifier object by using the k-nearest neighbors algorithm.

## K-nearest centroid [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.cluster.NearestCentroid.html)

=> from verticapy.machine\_learning.vertica import NearestCentroid

=> model= NearestCentroid(name="Name\_of\_Model", p=2) – Creates a NearestCentroid object by using the k-nearest centroid algorithm.

**Fitting, Predicting and Evaluating models**

# 

# Regression/Classification – Model Prediction

## Fitting

=> model.fit("public.Name\_of\_Model" , ["independent\_col\_1", "independent\_col\_2"], "dependent\_col") – Fit the model to the given independent inputs and dependent outputs.

## Prediction

=> model.predict(VDataFrame, X=["independent\_col\_1", "independent\_col\_2"], name="name\_of\_pred\_column") – Predicts and adds those values inside the VDataFrame using the new name of prediction columns.

# General Metrics

## Link to all [(link)](https://www.vertica.com/python/documentation/1.0.x/html/machine_learning_metrics.html)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mean Squared Error** | **R-squared** | **aic** | **bic** | **Explained Variance** |
| => model.score("mse") | => model.score("r2") | => model.score("aic") | => model.score("bic") | => model.score("var") |
| **Max error** | **R-squared adjusted** | **RMSE** | **Median Absolute Error** | **Mean Absolute Error** |
| => model.score("max") | => model.score("r2a") | => model.score("rmse") | => model.score("mae") | => model.score("mae") |

# Classification-specific Metrics

## Confusion Matrix [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.metrics.confusion_matrix.html)

=> model.confusion\_matrix(pos\_label="Label", cutoff=0.33) – Fit the model to the given independent inputs and dependent outputs.

## Lift Chart [(link)](https://www.vertica.com/python/documentation/1.0.x/html/chart_gallery_classification_curve.html)

=> from verticapy.machine\_learning.metrics import lift\_chart

=> lift\_chart("Response\_Column", "Prediction\_Probability", VDataFrame) – Draws a lift chart.

## ROC Curve [(link)](https://www.vertica.com/python/documentation/1.0.x/html/chart_gallery_classification_curve.html)

=> model.roc\_curve(nbins=12) – Plots the ROC curve.

**Managing models**

# 

# memModel To build models using their attributes [(link)](https://www.vertica.com/python/documentation/1.0.x/html/machine_learning_memmodels.html)

## For Linear Regression

=> from verticapy.machine\_learning.memmodel linear\_model import LinearModel

=> model=LinearModel (coefficients=[0.5, 1.2], intercept=2) – Builds a Linear Regression model from its attributes.

=> model.predict\_sql (["x1", "x2"]) – Generates the SQL code for deploying the model in Vertica.

# Generate SQL code [(link)](https://www.vertica.com/python/documentation/1.0.x/html/api/verticapy.machine_learning.vertica.ensemble.IsolationForest.to_sql.html)

## For Linear Regression

=> model.to\_sql() – Generates the SQL code for deploying the model in Vertica.