

# **Database Analytic Functions**

---

Release 17.20

June 2022

# **Copyright and Trademarks**

Copyright © 2017 - 2025 by Teradata. All Rights Reserved.

All copyrights and trademarks used in Teradata documentation are the property of their respective owners. See [Trademark Information](#).

## **Third-Party Materials**

Non-Teradata (i.e., third-party) sites, documents or communications ("Third-party Materials") may be accessed or accessible (e.g., linked or posted) in or in connection with a Teradata site, document or communication. Such Third-party Materials are provided for your convenience only and do not imply any endorsement of any third party by Teradata or any endorsement of Teradata by such third party. Teradata is not responsible for the accuracy of any content contained within such Third-party Materials, which are provided on an "AS IS" basis by Teradata. Such third party is solely and directly responsible for its sites, documents and communications and any harm they may cause you or others.

## **Warranty Disclaimer**

Except as may be provided in a separate written agreement with Teradata or required by applicable laws, all designs, specifications, statements, information, recommendations and content (collectively, "content") available from the Teradata Documentation website or contained in Teradata information products is presented "as is" and without any express or implied warranties, including, but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or noninfringement, which are hereby disclaimed. In no event shall Teradata corporation, its suppliers or partners be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of content, even if advised of the possibility of such damage.

The Content available from the Teradata Documentation website or contained in Teradata information products may contain references or cross-references to features, functions, products, or services that are not announced or available in your country. Such references do not imply that Teradata Corporation intends to announce such features, functions, products, or services in your country. Please consult your local Teradata Corporation representative for those features, functions, products, or services available in your country.

The Content available from the Teradata Documentation website or contained in Teradata information products may be changed or updated by Teradata at any time without notice. Teradata may also make changes in the products or services described in the Content at any time without notice.

The Content is subject to change without notice. Users are solely responsible for their application of the Content. The Content does not constitute the technical or other professional advice of Teradata, its suppliers or partners. Consult your own technical advisors before implementing any Content. Results may vary depending on factors not tested by Teradata.

## **Machine-Assisted Translation**

Certain materials on this website have been translated using machine-assisted translation software/tools. Machine-assisted translations of any materials into languages other than English are intended solely as a convenience to the non-English-reading users and are not legally binding. Anybody relying on such information does so at his or her own risk. No automated translation is perfect nor is it intended to replace human translators. Teradata does not make any promises, assurances, or guarantees as to the accuracy of the machine-assisted translations provided. Teradata accepts no responsibility and shall not be liable for any damage or issues that may result from using such translations. Users are reminded to use the English contents.

## **Feedback**

To maintain the quality of our products and services, e-mail your comments on the accuracy, clarity, organization, and value of this document to: [docs@teradata.com](mailto:docs@teradata.com).

Any comments or materials (collectively referred to as "Feedback") sent to Teradata Corporation will be deemed nonconfidential. Without any payment or other obligation of any kind and without any restriction of any kind, Teradata and its affiliates are hereby free to (1) reproduce, distribute, provide access to, publish, transmit, publicly display, publicly perform, and create derivative works of, the Feedback, (2) use any ideas, concepts, know-how, and techniques contained in such Feedback for any purpose whatsoever, including developing, manufacturing, and marketing products and services incorporating the Feedback, and (3) authorize others to do any or all of the above.

## **Confidential Information**

Confidential Information means any and all confidential knowledge, data or information of Teradata, including, but not limited to, copyrights, patent rights, trade secret rights, trademark rights and all other intellectual property rights of any sort.

The Content available from the Teradata Documentation website or contained in Teradata information products may include Confidential Information and as such, the use of such Content is subject to the non-use and confidentiality obligations and protections of a non-disclosure agreement or other such agreements to protect Confidential Information that you have executed with Teradata.

## **Product Safety**

Safety type	Description
 <b>NOTICE</b>	Indicates a situation which, if not avoided, could result in damage to property, such as to equipment or data, but not related to personal injury.
 <b>CAUTION</b>	For hardware products only Indicates a hazardous situation which, if not avoided, could result in minor or moderate personal injury.
 <b>WARNING</b>	For hardware products only Indicates a hazardous situation which, if not avoided, could result in death or serious personal injury.

# Contents

<b>Chapter 1: Introduction to Analytics Database Analytic Functions .....</b>	<b>6</b>
Changes and Additions .....	6
Analytics Database Analytic Function Categories .....	13
Usage Notes .....	21
<b>Chapter 2: Data Cleaning Functions .....</b>	<b>29</b>
Pack .....	29
StringSimilarity .....	35
TD_ConvertTo .....	43
TD_GetFutileColumns .....	48
TD_GetRowsWithoutMissingValues .....	53
TD_OutlierFilterFit .....	56
TD_OutlierFilterTransform .....	61
TD_SimpleImputeFit .....	65
TD_SimpleImputeTransform .....	69
Unpack .....	73
<b>Chapter 3: Data Exploration Functions .....</b>	<b>84</b>
MovingAverage .....	84
TD_CategoricalSummary .....	100
TD_ColumnSummary .....	104
TD_GetRowsWithMissingValues .....	109
TD_Histogram .....	112
TD_QQNorm .....	121
TD_UnivariateStatistics .....	126
TD_WhichMax .....	130
TD_WhichMin .....	133
<b>Chapter 4: Feature Engineering Transform Functions .....</b>	<b>136</b>
Antiselect .....	138
TD_BinCodeFit .....	142
TD_BinCodeTransform .....	150
TD_ColumnTransformer .....	153
TD_FunctionFit .....	158
TD_FunctionTransform .....	160
TD_NonLinearCombineFit .....	163
TD_NonLinearCombineTransform .....	166
TD_OneHotEncodingFit .....	168
TD_OneHotEncodingTransform .....	179

TD_OrdinalEncodingFit . . . . .	187
TD_OrdinalEncodingTransform . . . . .	201
TD_Pivoting . . . . .	207
TD_PolynomialFeaturesFit . . . . .	217
TD_PolynomialFeaturesTransform . . . . .	220
TD_RandomProjectionFit . . . . .	224
TD_RandomProjectionMinComponents . . . . .	228
TD_RandomProjectionTransform . . . . .	230
TD_RowNormalizeFit . . . . .	233
TD_RowNormalizeTransform . . . . .	237
TD_ScaleFit . . . . .	240
TD_ScaleTransform . . . . .	263
TD_TargetEncodingFit . . . . .	280
TD_TargetEncodingTransform . . . . .	295
TD_Unpivoting . . . . .	303
<b>Chapter 5: Feature Engineering Utility Functions . . . . .</b>	<b>313</b>
TD_FillRowID . . . . .	313
TD_NumApply . . . . .	316
TD_RoundColumns . . . . .	325
TD_StrApply . . . . .	330
<b>Chapter 6: Model Training Functions . . . . .</b>	<b>336</b>
TD_DecisionForest . . . . .	336
TD_GLM . . . . .	346
TD_KMeans . . . . .	369
TD_KNN . . . . .	381
TD_NaiveBayes . . . . .	391
TD_OneClassSVM . . . . .	397
TD_SVM . . . . .	405
TD_VectorDistance . . . . .	419
TD_XGBoost . . . . .	425
<b>Chapter 7: Model Scoring Functions . . . . .</b>	<b>451</b>
TD_DecisionForestPredict . . . . .	451
TD_GLMPredict . . . . .	459
TD_KMeansPredict . . . . .	465
TD_NaiveBayesPredict . . . . .	471
TD_OneClassSVMPredict . . . . .	477
TD_SVMPredict . . . . .	484
TD_XGBoostPredict . . . . .	492
<b>Chapter 8: Model Evaluation Functions . . . . .</b>	<b>508</b>
TD_SHAP . . . . .	508
TD_Silhouette . . . . .	525

TD_ClassificationEvaluator .....	533
TD_RegressionEvaluator .....	539
TD_ROC .....	546
TD_TrainTestSplit .....	551
<b>Chapter 9: Text Analytic Functions .....</b>	<b>558</b>
TD_NgramsSplitter .....	558
TD_NaiveBayesTextClassifierPredict .....	623
TD_NaiveBayesTextClassifierTrainer .....	630
TD_NERExtractor .....	634
TD_SentimentExtractor .....	641
TD_TextMorph .....	650
TD_TextParser .....	662
TD_TFIDF .....	673
TD_WordEmbeddings .....	679
<b>Chapter 10: Hypothesis Testing Functions .....</b>	<b>688</b>
Hypothesis Test Components .....	688
Hypothesis Test Types .....	689
TD_ANOVA .....	689
TD_Chisq .....	697
TD_FTest .....	705
TD_ZTest .....	715
<b>Chapter 11: Association Analysis Functions .....</b>	<b>729</b>
TD_CFilter .....	729
<b>Chapter 12: Path and Pattern Analysis Functions .....</b>	<b>740</b>
Terminology .....	740
Attribution .....	740
nPath .....	749
Sessionize .....	795
<b>Appendix A: How to Read Syntax .....</b>	<b>800</b>
<b>Appendix B: Additional Information .....</b>	<b>802</b>

# Introduction to Analytics Database Analytic Functions

The Advanced Analytics functions is a suite of scalable and distributed machine learning analytic functions to perform analytics on your dataset.

## Using the Analytics Database Analytical Content

The content describes the advanced analytic functions used for feature engineering, training, scoring, evaluation, pattern recognition, and text analytics use cases. The engine supports the use cases with a full dataset without compromising the end-to-end performance of an analytic workload.

## Why Would I Use this Content?

You can use the content to understand which function to use for the following use cases:

- Data cleaning (which includes use cases related to handling outliers, handling missing values, and parsing data)
- Data exploration (which includes use cases related to descriptive statistics and statistical tests)
- Feature Engineering (which includes use cases related to feature engineering utilities and categorical and continuous variable transform)
- Model building (which includes use cases related to model training, model evaluation, and model scoring)

## How Do I Use this Content?

You can use the content as follows:

1. Select a function from the [Overview](#) section.
2. Read the function description, syntax, syntax elements, and input and output sections for the selected function.
3. Download the zip file and use the dataset setup file to create the datasets.
4. Use the examples from the guide or SQL statements from the notepad and run the statements in the Teradata Studio or BTEQ environment.

## How Do I Get Started?

Before using the functions, read the following sections:

1. Read [Usage Notes](#).
2. Read the [How to Read Syntax](#) section.

## Changes and Additions

Date	Release	Description
September 2024	17.20.03.30	<p>New Features</p> <ul style="list-style-type: none"> <li>• TD_NERExtractor Perform Named Entity Recognition (NER) on input text according to user-defined dictionary words or regular expression (regex) patterns. See <a href="#">TD_NERExtractor</a>.</li> </ul>
August 2024	17.20.03.29	<p>New Features:</p> <ul style="list-style-type: none"> <li>• TD_TextMorph Generate morphs (standard form/Dictionary form) of the given tokens in the input dataset using lemmatization algorithm based on the English Dictionary. See <a href="#">TD_TextMorph</a>.</li> </ul> <p>Enhancements:</p> <ul style="list-style-type: none"> <li>• NGramSplitter: Added support for regular expressions (regex). See <a href="#">TD_Ngramsplitter</a>.</li> <li>• TD_TextParser: Added ability to count the occurrences of each token or stem (TokenFrequency) and obtain a comma separated list of positions for each token occurrence (ListPositions). See <a href="#">TD_TextParser</a>.</li> </ul>
July 2024	17.20.03.28	<p>New Features:</p> <ul style="list-style-type: none"> <li>• TD_CFilter Calculate several statistical measures of how likely each pair of items is to be purchased together. See <a href="#">TD_CFilter</a>.</li> <li>• TD_NaiveBayes Predicts the outcome of future observations based on their input variable values. You can use this function for solving classification problems. See <a href="#">TD_NaiveBayes</a>.</li> <li>• TD_NaiveBayesPredict Uses the model generated by the TD_NaiveBayes function to predict the outcomes for a test set of data. See <a href="#">TD_NaiveBayesPredict</a></li> </ul>
May 2024	17.20.03.26	<p>New Features:</p> <ul style="list-style-type: none"> <li>• TD_SHAP Compute the contribution of each feature in a prediction as as average marginal contribution of the feature value across all possible coalitions. See <a href="#">TD_SHAP</a>.</li> </ul> <p>Updated Information:</p> <p>The following functions were reorganized and updated content:</p> <ul style="list-style-type: none"> <li>• TD_CategoricalSummary. See <a href="#">TD_CategoricalSummary</a>.</li> <li>• TD_SVM. See <a href="#">TD_SVM</a>.</li> <li>• TD_SVMPredict. See <a href="#">TD_SVMPredict</a>.</li> <li>• TD_OneClassSVM. See <a href="#">TD_OneClassSVM</a>.</li> <li>• TD_OneClassSVMPredict. See <a href="#">TD_OneClassSVMPredict</a>.</li> </ul> <p>TD_DecisionForest. Best practice when ModelType is set to Classification. See <a href="#">TD_DecisionForest</a>.</p>
March 2024	17.20.03.24	Updated Information:

Date	Release	Description
		<ul style="list-style-type: none"> <li>TD_XGBoostPredict. Removed Detailed argument from syntax and tree_num from output table schema.</li> </ul>
February 2024	17.20.03.23	<p>Updated Information:</p> <ul style="list-style-type: none"> <li>Best practices when user session collation is set to Multinational. See <a href="#">Recommendations for Using Analytic Functions</a>.</li> </ul>
January 2024	17.20.03.22	<p>Enhancements:</p> <ul style="list-style-type: none"> <li>TD_ANOVA, TD_FTest, and TD_ZTest functions support alternate input table format (group-value), in addition to the already supported column-wise format. See <a href="#">TD_ANOVA</a>, <a href="#">TD_FTest</a>, and <a href="#">TD_ZTest</a>.</li> <li>TD_ScaleFit and TD_ScaleTransform functions accept input data in sparse format, in addition to the already supported dense format. You can perform scaling on input dataset which is present in sparse format and perform scaling on greater number of attributes. See <a href="#">TD_ScaleFit</a> and <a href="#">TD_ScaleTransform</a>.</li> </ul> <p>Updated Information:</p> <ul style="list-style-type: none"> <li>Use ASCII client collation when executing analytic functions. If your data has UNICODE characters, set the UTF8 client character set and Collation set to ASCII for the user session. See <a href="#">Recommendations for Using Analytic Functions</a>.</li> </ul>
December 2023	17.20.03.21	<p>New Features:</p> <ul style="list-style-type: none"> <li>TD_TFIDF Commonly used to determine how important a word is in relation to a document. See <a href="#">TD_TFIDF</a>.</li> </ul>
November 2023	17.20.03.20	<p>Enhancements:</p> <ul style="list-style-type: none"> <li>TD_XGBoost Performance and usability enhancements like supporting alternate argument names for NumBoostedTrees, ShrinkageFactor, and IterNum (inline with open source libraries). Some additional changes like added BaseScore argument in the function syntax, default values for ShrinkageFactor changed to 0.5 and RegularizationLambda changed to 1, and so on. See <a href="#">TD_XGBoost</a>.</li> <li>TD_XGBoostPredict Performance and usability enhancements like supporting alternate argument names for NumBoostRounds and IterNum (inline with open source libraries). Some additional changes like order by argument made optional, default values for IterNum increased from 3 to 10, and so on. See <a href="#">TD_XGBoostPredict</a>.</li> </ul>
October 2023	<ul style="list-style-type: none"> <li>17.20.03.19</li> <li>17.20.03.18</li> </ul>	<p>New Features:</p> <ul style="list-style-type: none"> <li>TD_Pivoting Pivot the data, that is, changes the data from sparse to dense format. See <a href="#">TD_Pivoting</a>.</li> <li>TD_Unpivoting Unpivots the data, that is, changes the data from dense format to sparse format. See <a href="#">TD_Unpivoting</a>.</li> </ul> <p>Enhancements:</p>

Date	Release	Description
		<ul style="list-style-type: none"> <li>TD_CategoricalSummary DistinctValue column supports VARCHAR (CHARACTER SET LATIN or UNICODE) data type in the output table schema. See <a href="#">TD_CategoricalSummary</a></li> <li>TD_DecisionForestPredict Performance and usability enhancements like changing accumulate columns at the end of the output. See <a href="#">TD_DecisionForestPredict</a>.</li> <li>TD_GLMPredict Optional Family <a href="#">TD_GLMPredict</a>.</li> <li>TD_KMeans TD_KMeans supports KMeans++ algorithm for initial centroids selection. KMeans++ algorithm is a way for choosing initial centroids far away from each other and reduces the possibility of initial centroids being chosen from the same cluster. KMeans++ improves the overall quality of clustering and can also speed up the convergence of KMeans algorithm. See <a href="#">TD_KMeans</a>.</li> <li>TD_KMeansPredict TD_KMeansPredict takes a table of cluster centroids output by the TD_KMeans function and an input table. It uses the model to assign the input data points to the cluster centroids. See <a href="#">TD_KMeansPredict</a>.</li> <li>TD_Scalefit and TD_ScaleTransform Supports PARTITION BY along with ParameterTable and AttributeTable to scale different input data partitions independently of each other. Added an optional argument IgnoreInvalidLocationScale that gives you an option to ignore errors for invalid values of location and scale parameters. See <a href="#">TD_ScaleFit</a> and <a href="#">TD_ScaleTransform</a>.</li> <li>TD_SVMPredict Optional ModelType argument added to TD_SVMPredict. The argument specifies the model type used by TD_SVM to train the dataset. See <a href="#">TD_SVMPredict</a>.</li> </ul> <p>Updated Information:</p> <ul style="list-style-type: none"> <li>Reorganized and updated content, <a href="#">TD_WordEmbeddings</a>. This function supports CHARACTER SET LATIN only.</li> <li>TD_GLM and TD_GLMPerSegment have been combined. TD_GLM supports PARTITION BY ANY and PARTITION BY <i>partition_by_column</i>. Use TD_GLM for model training. See <a href="#">TD_GLM</a>.</li> <li>TD_GLMPredict and TD_GLMPredictPerSegment have been combined. TD_GLMPredict supports PARTITION BY ANY and PARTITION BY <i>partition_by_column</i>. Use TD_GLMPredict for model scoring. See <a href="#">TD_GLMPredict</a>.</li> <li>Reorganized and updated content: <a href="#">TD_GetRowsWithoutMissingValues</a>, <a href="#">TD_KNN</a>, <a href="#">TD_NonLinearCombineFit</a>, <a href="#">TD_NumApply</a>, <a href="#">TD_OrdinalEncodingFit</a>, <a href="#">TD_OrdinalEncodingTransform</a>, <a href="#">TD_PolynomialFeaturesFit</a>, <a href="#">TD_PolynomialFeaturesTransform</a>, <a href="#">TD_RegressionEvaluator</a>, <a href="#">TD_SentimentExtractor</a>, <a href="#">TD_StrApply</a></li> </ul>
September 17	17.20.03. 17	Updated Information:

Date	Release	Description
		<ul style="list-style-type: none"> <li>TD_NaiveBayesTextClassifierPredict. Updated function name from NaiveBayesTextClassifierPredict to TD_NaiveBayesTextClassifierPredict. See <a href="#">TD_NaiveBayesTextClassifierPredict</a>.</li> <li>Information about redistributing input table rows to fewer AMPs is added. See <a href="#">Input Table Rows Redistribution</a> in <a href="#">Usage Notes</a>.</li> <li><a href="#">TD_XGBoost</a>. Reorganized and updated content. Redistributing the input table rows to one or fewer AMPs is recommended for improved training results.</li> <li><a href="#">TD_XGBoostPredict</a>. Reorganized and updated content.</li> </ul>
August	17.20.03.16	<p>Updated Information:</p> <ul style="list-style-type: none"> <li>TD_GLM and TD_GLMPerSegment have been combined. TD_GLM supports PARTITION BY ANY and PARTITION BY <i>partition_by_column</i>. Use TD_GLM for model training. See <a href="#">TD_GLM</a>.</li> <li>TD_GLMPredict and TD_GLMPredictPerSegment have been combined. TD_GLMPredict supports PARTITION BY ANY and PARTITION BY <i>partition_by_column</i>. Use TD_GLMPredict for model scoring. See <a href="#">TD_GLMPredict</a>.</li> </ul>
July 2023	17.20.03.15	<p>Updated Information:</p> <ul style="list-style-type: none"> <li><a href="#">TD_ClassificationEvaluator</a> argument to TD_GLMPredict. The a. Updated syntax and example, and data types for InputTable schema, and modified column name for primary Output Table schema.</li> <li><a href="#">TD_BinCodeTransform</a>. Reorganized and updated content.</li> <li><a href="#">TD_ChiSq</a>. Reorganized and updated content.</li> <li><a href="#">TD_ConvertTo</a>. Reorganized and updated content.</li> <li><a href="#">TD_OneHotEncodingFit</a>. Reorganized and updated content.</li> <li><a href="#">TD_OneHotEncodingTransform</a>. Reorganized and updated content.</li> </ul>
June 2023	17.20.03.14	<p>Updated Information:</p> <ul style="list-style-type: none"> <li>Information about resolving spool space exhaustion for large partitions at high concurrency for analytic functions is added in <a href="#">Usage Notes</a>. See <a href="#">Spool Space Exhaustion for Large Partitions at High Concurrency</a>.</li> </ul> <p>Enhancements:</p> <ul style="list-style-type: none"> <li>The AS InputTable alias is mandatory for the input table in the following functions: TD_GLM, TD_SVM, and TD_OneClassSVM. See <a href="#">TD_GLM Syntax</a>, <a href="#">TD_SVM Syntax</a>, and <a href="#">TD_OneClassSVM Syntax</a>.</li> <li>TD_GLM function supports StepwiseDirection. See <a href="#">TD_GLM</a>.</li> </ul>
May 2023	17.20.03.13	<p>Updated Information:</p> <ul style="list-style-type: none"> <li>Information about the size of a query allowed for analytic functions is added in <a href="#">Usage Notes</a>. See <a href="#">Size of the Query</a>.</li> <li><a href="#">TD_XGBoost</a>. The default value for the CoverageFactor argument is 1.0.</li> <li><a href="#">TD_WordEmbeddings</a>. The default value for the Operation argument is token-embedding, and the default value for the StemTokens argument is False.</li> <li><a href="#">TD_GLM</a>, <a href="#">TD_SVM</a>, <a href="#">TD_OneClassSVM</a>. The value for the BatchSize and IterNumNoChange arguments is a non-negative integer.</li> <li><a href="#">TD_KMeansPredict</a>. Reorganized and updated content.</li> </ul>

Date	Release	Description
April 2023	17.20.03.12	<p>Updated Information:</p> <ul style="list-style-type: none"> <li>Information about the maximum number of columns allowed in a database table for analytic functions is added in <a href="#">Usage Notes</a>. See <a href="#">Maximum Number of Columns in a Database Table</a>.</li> <li><a href="#">TD_UnivariateStatistics</a>. There may be different statistics values on different runs due to the precision difference for decimal and number columns.</li> <li><a href="#">TD_XGBoostPredict</a>. Test input can have a maximum of 2046 attributes.</li> <li><a href="#">TD_GLMPredict</a>. The ON clause alias changed from Model to ModelTable. See <a href="#">TD_GLMPredict Syntax</a>.</li> </ul>
March 2023	17.20.03.11	<p>New Features:</p> <ul style="list-style-type: none"> <li><a href="#">TD_TargetEncodingFit</a>. Generates hyperparameters for use by TD_TargetEncodingTransform.</li> <li><a href="#">TD_TargetEncodingTransform</a>. Uses the hyperparameters generated by TD_TargetEncodingFit to encode categorical values.</li> <li><a href="#">TD_GLMPerSegment</a>. Trains a whole data set by partitioning, and creates a single model for each partition.</li> <li><a href="#">TD_GLMPredictPerSegment</a>. Predicts target value (regression) and class label (classification) for test data using a corresponding GLM model trained using TD_GLMPerSegment.</li> </ul> <p>Updated Information:</p> <ul style="list-style-type: none"> <li><a href="#">TD_XGBoost</a>. For Classification, the SELECT statement must have a deterministic output.</li> <li><a href="#">TD_ANOVA</a>. One-Way ANOVA is supported. Syntax elements and Output Table schema are updated.</li> <li>[Legacy - Removed] SVMSparsePredict. Updated SQL Call in function example.</li> </ul>
February 2023	17.20.03.10	Information about non-deterministic behavior of function output is added in <a href="#">Usage Notes</a> . See <a href="#">Non-Deterministic Behavior</a> .
January 2023	17.20.03.09	<p>New Features:</p> <ul style="list-style-type: none"> <li><a href="#">TD_DecisionForestPredict</a>. Uses the model output by TD_DecisionForest function to analyze the input data and make predictions.</li> <li><a href="#">TD_TrainTestSplit</a>. Simulates how a model performs on new data.</li> <li><a href="#">TD_XGBoost</a>. Performs classification and regression analysis on data sets and generates a model for TD_XGBoostPredict to run the predictive algorithm.</li> <li><a href="#">TD_XGBoostPredict</a>. Runs the predictive algorithm based on the model generated by TD_XGBoost.</li> </ul> <p>Updated Information:</p> <ul style="list-style-type: none"> <li><a href="#">TD_DecisionForest</a>. Processing time is controlled by the number of trees, and by the complexity of the trees.</li> <li><a href="#">TD_OneHotEncodingFit</a>. Added Sparse input example.</li> <li><a href="#">TD_OneHotEncodingTransform</a>. Added Sparse input example.</li> </ul>
December 2022	17.20.03.08	<p>New Features:</p> <ul style="list-style-type: none"> <li><a href="#">TD_SVM</a></li> <li><a href="#">TD_SVMPredict</a></li> </ul>

Date	Release	Description
		<ul style="list-style-type: none"> <li>• <a href="#">TD_OneClassSVM</a></li> <li>• <a href="#">TD_OneClassSMPredict</a></li> </ul> <p>Enhancements:</p> <ul style="list-style-type: none"> <li>• <a href="#">nPath</a> supports CLOB output.</li> </ul>
November 2022	17.20.03.07	<p>New Features:</p> <ul style="list-style-type: none"> <li>• <a href="#">TD_WordEmbeddings</a></li> </ul> <p>Enhancements:</p> <ul style="list-style-type: none"> <li>• <a href="#">TD_OrdinalEncodingFit</a>: Multiple column support added. Changed syntax, CategoryTable and Output Table schema, and examples.</li> <li>• <a href="#">TD_OrdinalEncodingTransform</a>: Multiple column support added. Changed FitTable and Output Table schema, and examples.</li> <li>• <a href="#">TD_OneHotEncodingFit</a>: Multiple column support added. Changed syntax, Output Table schema for dense input, and examples. Added CategoryTable schema for dense input.</li> <li>• <a href="#">TD_OneHotEncodingTransform</a>: Multiple column support added. Changed FitTable and Output Table schema, and examples.</li> <li>• <a href="#">TD_Histogram</a>: Multiple column support added. Changed syntax, MinMaxTable and Output Table schema, and examples.</li> </ul>
Oct 2022	17.20.03.06	<a href="#">TD_KNN</a>
June 2022	17.20.03.01	<p>New Features:</p> <ul style="list-style-type: none"> <li>• <a href="#">TD_ANOVA</a></li> <li>• <a href="#">TD_TextParser</a></li> <li>• <a href="#">TD_SentimentExtractor</a></li> <li>• <a href="#">TD_NaiveBayesTextClassifierTrainer</a></li> <li>• <a href="#">TD_ROC</a></li> <li>• <a href="#">TD_RegressionEvaluator</a></li> <li>• <a href="#">TD_ClassificationEvaluator</a></li> <li>• <a href="#">TD_Silhouette</a></li> <li>• <a href="#">TD_GLMPredict</a></li> <li>• <a href="#">TD_KMeansPredict</a></li> <li>• <a href="#">TD_VectorDistance</a></li> <li>• <a href="#">TD_GLM</a></li> <li>• <a href="#">TD_KMeans</a></li> <li>• <a href="#">TD_DecisionForest</a></li> <li>• <a href="#">TD_ColumnTransformer</a></li> <li>• <a href="#">TD_NonLinearCombineFit</a></li> <li>• <a href="#">TD_NonLinearCombineTransform</a></li> <li>• <a href="#">TD_OrdinalEncodingFit</a></li> <li>• <a href="#">TD_OrdinalEncodingTransform</a></li> <li>• <a href="#">TD_RandomProjectionMinComponents</a></li> <li>• <a href="#">TD_RandomProjectionFit</a></li> <li>• <a href="#">TD_RandomProjectionTransform</a></li> </ul>

Date	Release	Description
		• <a href="#">TD_GetFutileColumns</a>

## Analytics Database Analytic Function Categories

- [Data Cleaning Functions](#)
- [Data Exploration Functions](#)
- [Feature Engineering Transform Functions](#)
- [Feature Engineering Utility Functions](#)
- [Model Training Functions](#)
- [Model Scoring Functions](#)
- [Model Evaluation Functions](#)
- [Text Analytic Functions](#)
- [Hypothesis Testing Functions](#)
- [Association Analysis Functions](#)
- [Path and Pattern Analysis Functions](#)

## Data Cleaning Functions

### [TD\\_GetFutileColumns](#)

Returns the futile column names.

### [TD\\_OutlierFilterFit](#)

Calculates the lower\_percentile, upper\_percentile, count of rows, and median for the specified input table columns.

### [TD\\_OutlierFilterTransform](#)

Filters outliers from the input table.

### [TD\\_GetRowsWithoutMissingValues](#)

Displays the rows that have non-NULL values in the specified input table columns.

### [TD\\_SimpleImputeFit](#)

Outputs a table of values to substitute for missing values in the input table.

### [TD\\_SimpleImputeTransform](#)

Substitutes specified values for missing values in the input table.

### [TD\\_ConvertTo](#)

Converts the specified input table columns to specified data types.

**Pack**

Compresses data in multiple columns into a single packed data column.

**Unpack**

Expands data from a single packed column to multiple unpacked columns.

**StringSimilarity**

Calculates the similarity between two strings, using the specified comparison method.

## Data Exploration Functions

**MovingAverage**

Computes average values in a series.

**TD\_CategoricalSummary**

Displays the distinct values and their counts for each specified input table column.

**TD\_ColumnSummary**

Displays a summary of each specified input table column.

**TD\_GetRowsWithMissingValues**

Displays the rows that have NULL values in the specified input table columns.

**TD\_Histogram**

Calculates the frequency distribution of a data set.

**TD\_QQNorm**

Checks whether the values in the specified input table columns are normally distributed.

**TD\_UnivariateStatistics**

Displays descriptive statistics for each specified numeric input table column.

**TD\_WhichMax**

Displays all rows that have the maximum value in a specified input table column.

**TD\_WhichMin**

Displays all rows that have the minimum value in specified input table column.

## Feature Engineering Transform Functions

**Antiselect**

AntiSelect returns all columns except those specified.

**TD\_BinCodeFit**

Converts numeric data to categorical data by binning the numeric data into multiple numeric bins (intervals).

**TD\_BinCodeTransform**

Transforms input table columns from the BinCodeFit function output.

**TD\_ColumnTransformer**

Transforms the input table columns in a single operation.

**TD\_FunctionFit**

Determines whether specified numeric transformations can be applied to specified input columns.

**TD\_FunctionTransform**

Applies numeric transformations to input columns.

**TD\_NonLinearCombineFit**

Returns the target columns and a specified formula which uses the non-linear combination of existing features.

**TD\_NonLinearCombineTransform**

Generates the values of the new feature using the specified formula from the TD\_NonLinearCombineFit function output.

**TD\_OneHotEncodingFit**

Outputs a table of attributes and categorical values to the TD\_OneHotEncodingTransform function.

**TD\_OneHotEncodingTransform**

Encodes specified attributes and categorical values as one-hot numeric vectors using the output from the TD\_OneHotEncodingFit function.

**TD\_OrdinalEncodingFit**

Encodes specified attributes and categorical values as one-hot numeric vectors using the output from the TD\_OneHotEncodingFit function.

**TD\_OrdinalEncodingTransform**

Maps the categorical value to a specified ordinal value using the TD\_OrdinalEncodingFit output.

**TD\_Pivoting**

Pivots the data, that is, changes the data from sparse to dense format.

**TD\_PolynomialFeaturesFit**

Stores all the specified values in the argument in a tabular format.

**TD\_PolynomialFeaturesTransform**

Extracts values of arguments from the output of the TD\_PolynomialFeaturesFit function and generates a feature matrix of all polynomial combinations of the features.

**TD\_RandomProjectionFit**

Returns a random projection matrix based on the specified arguments.

**TD\_RandomProjectionMinComponents**

Calculates the minimum number of components required for applying RandomProjection on the given dataset for the specified epsilon(distortion) parameter value.

**TD\_RandomProjectionTransform**

Converts the high-dimensional input data to a lower-dimensional space using the TD\_RandomProjectionFit function output.

**TD\_RowNormalizeFit**

Outputs a table of parameters and specified input columns to TD\_RowNormalizeTransform which normalizes the input columns row-wise.

**TD\_RowNormalizeTransform**

Normalizes the input columns row-wise using the output of the TD\_RowNormalizeFit function.

**TD\_ScaleFit**

Outputs a table of statistics to the TD\_ScaleTransform function.

**TD\_ScaleTransform**

Scales the specified input table columns using the output of the TD\_ScaleFit function.

**TD\_TargetEncodingFit**

Takes the InputTable and a CategoricalTable as input and generates the required hyperparameters to be used by the TD\_TargetEncodingTransform function for encoding the categorical values.

**TD\_TargetEncodingTransform**

Takes the InputTable and a FitTable generated by TD\_TargetEncodingFit for encoding the categorical values.

**TD\_Unpivoting**

Unpivots the data, that is, changes the data from dense format to sparse format.

## Feature Engineering Utility Functions

### [TD\\_FillRowID](#)

Adds a column of unique row identifiers to the input table.

### [TD\\_NumApply](#)

Applies a specified numeric operator to the specified input table columns.

### [TD\\_RoundColumns](#)

Rounds the values of each specified input table column to a specified number of decimal places.

### [TD\\_StrApply](#)

Applies a specified string operator to the specified input table columns.

## Model Training Functions

### [TD\\_DecisionForest](#)

Used for classification and regression predictive modeling.

### [TD\\_GLM](#)

Performs regression analysis on data sets where the response follows an exponential family distribution.

### [TD\\_KMeans](#)

Groups a set of observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid).

### [TD\\_KNN](#)

Predicts the test data by computing nearest neighbors from training data based on a similarity (distance) metric.

### [TD\\_NaiveBayes](#)

Predicts the outcome of future observations based on their input variable values.

### [TD\\_OneClassSVM](#)

Performs classification analysis on data sets to identify outliers or novelty in the data.

### [TD\\_SVM](#)

Performs classification and regression analysis on data sets.

**TD\_VectorDistance**

Accepts a table of target vectors and a table of reference vectors and returns a table that contains the distance between target-reference pairs.

**TD\_XGBoost**

Performs classification and regression analysis on data sets and generates a model for TD\_XGBoostPredict to run the predictive algorithm.

## Model Scoring Functions

**TD\_DecisionForestPredict**

Uses the model output by TD\_DecisionForest function to analyze the input data and make predictions.

**TD\_GLMPredict**

Predicts target values (regression) and class labels (classification) for test data using a GLM model of the TD\_GLM function.

**TD\_KMeansPredict**

Uses the cluster centroids in the TD\_KMeans function output to assign the input data points to the cluster centroids.

**TD\_NaiveBayesPredict**

Uses the model generated by TD\_NaiveBayes function to predict the outcomes for a test set of data.

**TD\_OneClassSVMPredict**

Predicts target class labels (classification) for test data using a one-class SVM model trained by TD\_OneClassSVM.

**TD\_SVMPredict**

Predicts target values (regression) and class labels (classification) for test data using an SVM model trained by TD\_SVM.

**TD\_XGBoostPredict**

Runs the predictive algorithm based on the model generated by TD\_XGBoost.

## Model Evaluation Functions

**TD\_SHAP**

Computes the contribution of each feature in a prediction as as average marginal contribution of the feature value across all possible coalitions.

**TD\_Silhouette**

Determines how well the data is clustered among clusters.

**TD\_ClassificationEvaluator**

Computes the Confusion matrix, precision, recall, and F1-score based on the observed labels (true labels) and the predicted labels.

**TD\_RegressionEvaluator**

Computes metrics to evaluate and compare multiple models and summarizes how close predictions are to their expected values.

**TD\_ROC**

Accepts a set of prediction-actual pairs for a binary classification model and calculates the True-positive rate (TPR), False-positive rate (FPR), The area under the ROC curve (AUC), and Gini coefficient values for a range of discrimination thresholds.

**TD\_TrainTestSplit**

Simulates model performance on new data.

## Text Analytic Functions

**TD\_Ngramsplitter**

Tokenizes (splits) an input stream and emits  $n$  multigrams, based on specified delimiter and reset parameters. Useful for sentiment analysis, topic identification, and document classification.

**TD\_NaiveBayesTextClassifierPredict**

Uses the model output by TD\_NaiveBayesTextClassifierTrainer function to analyze the input data and make predictions.

**TD\_NaiveBayesTextClassifierTrainer**

Calculates the conditional probabilities for token-category pairs, the prior probabilities, and the missing token probabilities for all categories.

**TD\_NERExtractor**

Performs Named Entity Recognition (NER) on input text according to user-defined dictionary words or regular expression (regex) patterns.

**TD\_SentimentExtractor**

Uses a dictionary model to extract the sentiment (positive, negative, or neutral) of each input document or sentence.

**TD\_TextMorph**

Generates morphs (standard form/Dictionary form) of the given tokens in the input dataset using lemmatization algorithm based on English Dictionary.

**TD\_TextParser**

Tokenizes an input stream of words and creates a row for each word in the output table.

**TD\_TFIDF**

Takes any document set and outputs the Term Frequency, Inverse Document Frequency, and Term Frequency - Inverse Document Frequency scores for each term.

**TD\_WordEmbeddings**

Uses training and prediction to determine the similarity between words and phrases.

## Hypothesis Testing Functions

**TD\_ANOVA**

Performs analysis of variance (ANOVA) test to analyze the difference between the means.

**TD\_ChiSq**

Performs Pearson's chi-squared test for independence.

**TD\_FTest**

Performs an *F*-test, for which the test statistic has an *F*-distribution under the null hypothesis.

**TD\_ZTest**

Performs a Z-test, for which the distribution of the test statistic under the null hypothesis can be approximated by normal distribution.

## Association Analysis Functions

**TD\_CFilter**

Calculates several statistical measures of how likely each pair of items is to be purchased together.

## Path and Pattern Analysis Functions

**Attribution**

Calculates attributions with a wide range of distribution models. Often used in web-page analysis.

**nPath**

Performs regular pattern matching over a sequence of rows from one or more inputs.

**nPath**

Maps each click in a clickstream to a unique session identifier.

## Usage Notes

These usage notes apply to every function in this document.

---

**Note:**

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).  
For information about PTCs, see *International Character Set Support*, B035-1125.
- This function does not support KanjiSJIS or Graphic data types.
- SELECT TOP gives non-deterministic results. Therefore, identical queries including this instruction may produce different results.

- 
- [Input Table Rows Redistribution](#)
  - [Function Syntax Descriptions](#)
  - [Size of the Query](#)
  - [AMP Configuration Impact on Function Execution](#)
  - [Limitations When Using PARTITION BY](#)
  - [Functions Ignore Disallowed Syntax Elements](#)
  - [Input Table Schemas](#)
  - [Function Names with and without TD Prefix](#)
  - [Accumulated Columns Impact on Function Performance](#)
  - [Datatype Change in Accumulated Columns](#)
  - [View Online Help for Database Analytic Functions](#)
  - [BC/BCE Timestamps](#)
  - [Workload Management Configuration for Analytics Database Analytic Functions](#)
  - [Avoid Deadlocks Using Volatile Tables](#)
  - [Non-Deterministic Behavior](#)
  - [Maximum Number of Columns in a Database Table](#)
  - [Spool Space Exhaustion for Large Partitions at High Concurrency](#)
  - [Recommendations for Using Analytic Functions](#)

## How to Read Syntax

To understand the function syntax, review [How to Read Syntax](#).

## Input Table Rows Redistribution

To redistribute input table rows to fewer AMPs, sparse map can be used. For example, to redistribute a table to four AMPs, first create a sparse map of four AMPs named sparse\_map\_4 using the following command:

```
CREATE MAP sparse_map_4 FROM td_map1 SPARSE ampcount=4;
```

Then, create a table using this sparse map, as follows.

```
CREATE TABLE Table1, MAP=sparse_map_4 (a1 INTEGER, b1 INTEGER) PRIMARY INDEX (a1);
```

All data inserted into this table is distributed to four AMPs only.

For more information about maps, see *Teradata® Database SQL Data Definition Language - Syntax and Examples*, *Teradata® Database Design*, and *Teradata® Database Administration*.

## Function Syntax Descriptions

### **SELECT Statement Clauses**

The function syntax descriptions in this document are SQL SELECT statements. For simplicity, the descriptions do not show every possible SELECT statement clause. However, you can use any valid SELECT statement clauses. For information about SELECT statement options, see *Teradata Vantage™ - SQL Data Manipulation Language*, B035-1146.

Many examples in this document use ORDER BY clauses that the function syntax descriptions do not show.

### **Function Syntax Element Order**

Function syntax elements must appear after the USING clause, but they need not appear in the order shown in the function syntax description.

Many examples in this document do not specify their syntax elements in the order shown in the function syntax description.

### **Function Input Types**

- PARTITION BY expression:

The syntax defines the distribution/partitioning of the input result set before the function operates on it. All AMPs process the partitions parallelly, but if there are multiple partitions per AMP, then each partition is processed sequentially.

- PARTITION BY ANY:

The syntax does not alter the data distribution of input and lets the function operate on the input result set without any redistribution. If there is no PARTITION BY expression syntax, it defaults to the PARTITION BY ANY syntax. The entire result set per AMP forms a single group or a partition.

- DIMENSION:

The syntax duplicates the input result set to all AMPs before the function operates on it. You can use this syntax for multiple inputs only.

- HASH BY expression:

The syntax defines the distribution/partitioning of the input result set before the function operates on it. However, unlike PARTITION BY expression, it does not sort the result set after partitioning, and the entire result set per AMP forms a single group or partition. The HASH BY expression is always supported with the PARTITION BY ANY clause.

See, [How to Read Syntax](#)

## Size of the Query

When the size of a query is larger than 64000 characters, the internal descriptor runs out of space and an error is reported.

The workaround is to decrease the query size. If column names are being specified in the query, try using indices.

## AMP Configuration Impact on Function Execution

The execution strategy of some analytic functions for the PARTITION BY ANY syntax varies with the number of AMPs in the system, and the function output may vary across different configurations.

The output is correct in all cases and does not impact the quality of results for large datasets.

If a degraded quality is observed in the results for small data sets, then you may restrict to one or few AMPs by carefully selecting the primary index to improve the quality of results.

The following functions whose output may vary by AMP configuration:

- TD\_GLM
- TD\_DecisionForest

## Limitations When Using PARTITION BY

For database analytic functions where PARTITION BY columns appear in the output schema, only LATIN columns can be used in PARTITION BY clause. UNICODE columns are not supported in such scenario.

## Functions Ignore Disallowed Syntax Elements

If you call a function with a disallowed syntax element, most functions ignore the syntax element without returning an error.

## Input Table Schemas

Input table schemas show only the columns that a function uses. Unless otherwise noted, input tables can have additional columns, but the function ignores them.

## Function Names with and without TD Prefix

The functions with the prefix TD are a new generation of advanced analytic functions that use the SQL-MapReduce framework and keep the resource usage under budget.

MapReduce is a framework for operating on large sets of data using massively parallel processing (MPP) systems. MapReduce enables complex analysis to be performed efficiently on extremely large sets of data, such as those obtained from weblogs and clickstreams. Its application areas include machine learning, scientific data analysis, and document classification.

The syntax of corresponding TD and non-TD functions may differ, but given the same inputs and syntax element values, they produce the same results (with the minor exceptions noted in specific functions).

## Accumulated Columns Impact on Function Performance

Consider the following points if the functions display the accumulated columns as the first columns of the output:

- The data type of the first column cannot be BLOB or CLOB.
- The first column becomes the Primary Index and must be selected carefully.
- The Primary Index column (by default, the first column) affects the data distribution and performance. The column with more unique values must be the Primary Index column and vice versa. If the first column in the output is the accumulated column, then you must select the Primary Index column explicitly (if the default column is not optimal) for better performance.

The following functions add the accumulated columns at the end of the output:

- NaiveBayesPredict
- DecisionTreePredict
- Pack
- Unpack
- TD\_KMeansPredict
- TD\_Silhouette

The following functions add the accumulated columns at the beginning of the output:

- GLMPredict

- DecisionForestPredict
- StringSimilarity
- TD\_Ngramsplitter
- TD\_GetRowsWithMissingValues
- TD\_GetRowsWithoutMissingValues
- TD\_ConvertTo
- TD\_qqnorm
- TD\_TextParser
- TD\_NumApply
- TD\_StrApply
- TD\_RoundColumns
- TD\_BincodeTransform
- TD\_NonLinearCombineTransform
- TD\_OrdinalEncodingTransform
- TD\_PolynomialFeaturesTransform
- TD\_RowNormalizeTransform
- TD\_ScaleTransform
- TD\_RandomProjectionTransform
- TD\_SentimentExtractor

## Datatype Change in Accumulated Columns

The following functions change the data type of a column in the input table to a different data type in the output table:

- TD\_NumApply
 

If you set the Inplace argument as false and specify the target columns in the Accumulate argument, then the data type of target columns in the output can be REAL or FLOAT.
- TD\_StrApply
 

If you set the Inplace argument as false and specify the target columns in the Accumulate argument, then for the following StringOperation argument values, the data type of target columns is VARCHAR (UNICODE) in the output:

STRINGCON, STRINGLIKE, STRINGPAD, STRINGTRIM, STRINGINDEX ]
- TD\_NonLinearCombineTransform
 

If you specify the target columns in the Accumulate argument, then the data type of the target columns in the output can be REAL or FLOAT.
- TD\_KMeansPredict
 

If you specify the target columns in the Accumulate argument, then the data type of the target columns in the output can be REAL or FLOAT.

- TD\_Silhouette
 

If you specify the target columns in the Accumulate argument, then the data type of the target columns in the output can be REAL or FLOAT.
- TD\_RandomProjectionTransform
 

If you specify the target columns in the Accumulate argument, then the data type of the target columns in the output can be REAL or FLOAT.
- TD\_SentimentExtractor
 

If you specify the Text column in the Accumulate argument, then the data type of the Text column in the output is VARCHAR (UNICODE).
- TD\_FunctionTransform
 

If a numeric column is not specified in the IDColumns argument, then the data type of the numeric column in the output can be REAL or FLOAT.
- Pack
 

If you set the Colcast argument as True and specify the target columns in the Accumulate argument, then the data type of target columns in the output can be VARCHAR.
- TD\_Qqnorm and TD\_PolynomialFeaturesTransform
 

The target columns are by default included in the output. The data type of the target columns included in the output can change to Double Precision or FLOAT.
- TD\_NBTCT
 

If you specify the datatype for the token or doccategory columns as Char or VARCHAR, then the data type of token or category in the output can be VARCHAR Unicode.
- TD\_Histogram
 

If you specify the datatype for label from minmaxtable table schema as BYTEINT, SMALLINT, INTEGER, or BIGINT, then the data type of the label in the output can be BIGINT. If you specify CHAR or VARCHAR as the datatype for the label, then the data type of the label in the output can be VARCHAR Unicode.

## View Online Help for Database Analytic Functions

Online help is available for each Analytics Database analytic function.

- For information about a function, type:  
`HELP 'function_name'`  
 For example:  
`HELP 'SQL NPATH'`

## BC/BCE Timestamps

Analytics Database functions do not support Before the Common Era (BCE) timestamps. BCE is an alternative to Before Christ (BC). These are examples of BC/BCE timestamps:

```
4713-01-01 11:07:11-07:52:58 BC  
4713-01-01 11:07:11 BC
```

## Workload Management Configuration for Analytics Database Analytic Functions

Analytics Database analytic functions can be memory- and compute-intensive and impact other workloads, depending on function parameters and input table sizes. To learn to use Workload Management throttles to limit concurrency and memory, see *Teradata Vantage™ - Workload Management User Guide*, B035-1197.

## Avoid Deadlocks Using Volatile Tables

When you use the CREATE TABLE syntax to directly save the function results, and if multiple such queries are running concurrently, deadlocks may occur due to the locks on the dictionary tables that get updated with the metadata of the new tables.

In this case, you must use VOLATILE tables rather than permanent tables to avoid dictionary locks. You can view the results in the VOLATILE tables and make a separate CREATE TABLE SQL call to store the results in a permanent table.

## Non-Deterministic Behavior

The output may be non-deterministic for some functions if the insertion order of the rows in the table change, or the cluster configuration (for example, different number of AMPs) changes. This is due to change in data distribution. Examples include TD\_SVM, TD\_GLM, TD\_DecisionForest, TD\_XGBoost, TD\_KMeans, and so on.

The output may be non-deterministic for some functions due to the random nature of the algorithm. This is typically controlled by the Seed argument. Examples include TD\_DecisionForest, TD\_KMeans, and so on.

## Maximum Number of Columns in a Database Table

A maximum of 2047 columns are allowed in a database table for analytic functions. These columns include the feature columns, response columns, id columns, partitioning/grouping, and ordering columns. Some functions have additional restrictions on the maximum allowed number of columns, which are listed in the function description.

## Spool Space Exhaustion for Large Partitions at High Concurrency

If a function that operates on partitions (the query has the PARTITION BY KEY syntax in the ON clause, possibly containing the PartitionColumns argument clause), is invoked on data with large partition sizes and high concurrency, the spool space for the database user may be exhausted during partitioning of data and an error is reported.

You can reduce concurrency or decrease partition sizes to resolve the issue. If partition sizes cannot be decreased, run the function on each partition separately using the PARTITION BY ANY syntax, where it is allowed.

## Recommendations for Using Analytic Functions

Use ASCII client collation when executing analytic functions. If your data has UNICODE characters, set the UTF8 client character set and Collation set to ASCII for the user session.

See Knowledge Article KB0052472, available at <https://docs.teradata.com/>. You must log in before you can search for it.

### Multinational Collation

If the user session collation is set to Multinational, analytic functions may report an error in certain situations. Best practice is to do the following:

- Reduce the size or number of input and output fields.
- Remove ORDER BY from larger output data fields.
- Run the query with the user session collation set to ASCII.

# Data Cleaning Functions

Data cleaning functions prepare the input data set for the next set of transformations. Use data cleaning functions for the following:

- Handling outliers
  - [TD\\_GetFutileColumns](#)
  - [TD\\_OutlierFilterFit](#)
  - [TD\\_OutlierFilterTransform](#)
- Handling missing values
  - [TD\\_GetRowsWithoutMissingValues](#)
  - [TD\\_SimpleImputeFit](#)
  - [TD\\_SimpleImputeTransform](#)
- Parsing data
  - [Pack](#)
  - [StringSimilarity](#)
  - [TD\\_ConvertTo](#)
  - [Unpack](#)

## Pack

The Pack function packs data from multiple input columns into a single column. The packed column has a virtual column for each input column. By default, virtual columns are separated by commas and each virtual column value is labeled with its column name.

Pack complements the function [Unpack](#), but you can use it on any columns that meet the input requirements.

Before packing columns, note their data types—you need them if you want to unpack the packed column.

---

### Note:

This function does not support locale-based formatting with the SDF file.

---

The Pack function is a powerful data profiling and cleansing tool that simplifies and organizes data sets. It works by combining data from multiple input columns into a single column. This helps to eliminate redundancy, making data sets more efficient and easier to manage. The resulting packed column includes virtual columns for each input column, which helps to identify the origin of each value and facilitate data cleaning efforts. By default, each virtual column is separated by commas and identified by its column name. This ensures that data is easily traceable and organized for analysis.

The Pack function is particularly useful for working with large data sets, where errors and inconsistencies can be more difficult to spot. By combining input columns into a single packed column, the Pack function can help to identify errors and inconsistencies more easily. Furthermore, the virtual columns created for each input column can assist with data cleaning efforts, making it easier to maintain data consistency and accuracy. In summary, the Pack function is a valuable data cleaning tool that streamlines data management, enhances data quality, and improves overall data organization.

## Function Information

- [Pack Syntax](#)
- [Required Syntax Elements for Pack](#)
- [Optional Syntax Elements for Pack](#)
- [Pack Input](#)
- [Pack Output](#)
- [Examples: How to Use Pack](#)

## Pack Syntax

```
Pack (
    ON { table | view | (query) }
    USING
        [ TargetColumns ({ 'target_column' | target_column_range }[,...]) ]
        [ Delimiter ('delimiter') ]
        [ IncludeColumnName ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
        OutputColumn ('output_column')
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
        [ ColCast ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for Pack

### ON clause

The ON clause accepts....

**OutputColumn**

Specify the name to give to the packed output column. The name must be a valid object name, as defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

## Optional Syntax Elements for Pack

**TargetColumns**

Specify the names of the input table columns to pack into a single output column. Column names must be valid object names, which are defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

These names become the column names of the virtual columns. If you specify this syntax element, but do not specify all input table columns, the function copies the unspecified input table columns to the output table.

Default behavior: All input table columns are packed into a single output column.

**Delimiter**

Specify the delimiter, a single Unicode character in Normalization Form C (NFC), that separates the virtual columns in the packed data. The delimiter is case-sensitive.

Default: ',' (comma)

**IncludeColumnName**

Specify whether to label each virtual column value with its column name (making the virtual column *target\_column:value*).

Default: 'true'

**Accumulate**

Specify the input columns to copy to the output table.

**ColCast**

Specify whether to cast each numeric *target\_column* to VARCHAR.

Specifying 'true' decreases run time for queries with numeric target columns.

Default: false

## Pack Input

### Input Table Schema

Column	Data Type	Description
<i>target_column</i>	Any	Column to pack, with other input columns, into single output column.
<i>accumulate_column or other_input_column</i>	Any	Column to copy to output table. Typically, one such column contains row identifiers.

## Pack Output

### Output Table Schema

Column	Data Type	Description
<i>output_column</i>	VARCHAR	Packed column. If a column of type DATE is packed into the output, its format is 'YYYY-MM-DD'.
<i>accumulate_column</i>	Nonnumeric column or numeric column with ColCast ('false'): Same as in input table Numeric column with ColCast ('true'): VARCHAR	Column copied from input table.
<i>other_input_column</i>	Same as in input table	[Column appears only without Accumulate, once for each specified <i>other_input_column</i> .] Column copied from input table.

## Examples: How to Use Pack

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Pack Examples Input](#)
- [Example: Default Options](#)
- [Example: Nondefault Options](#)

## Pack Examples Input

The input table, ville\_temperature contains temperature readings for the cities Nashville and Knoxville, in the state of Tennessee.

ville\_temperature

sn	city	state	period	temp_f
1	Nashville	Tennessee	2022-01-01 00:00:00	35.1
2	Nashville	Tennessee	2022-01-01 01:00:00	36.2
3	Nashville	Tennessee	2022-01-01 02:00:00	34.5
4	Nashville	Tennessee	2022-01-01 03:00:00	33.6
5	Nashville	Tennessee	2022-01-01 04:00:00	33.1
6	Knoxville	Tennessee	2022-01-01 03:00:00	33.2
7	Knoxville	Tennessee	2022-01-01 04:00:00	32.8
8	Knoxville	Tennessee	2022-01-01 05:00:00	32.4
9	Knoxville	Tennessee	2022-01-01 06:00:00	32.2
10	Knoxville	Tennessee	2022-01-01 07:00:00	32.4

## Example: Default Options

This example specifies the default options for Delimiter and IncludeColumnName.

### Pack with Default Options SQL Call

```
SELECT * FROM Pack (
    ON ville_temperature
    USING
        Delimiter(',')
        OutputColumn ('packed_data')
        IncludeColumnName ('true')
        TargetColumns ('[1:4]')
        Accumulate ('sn')
) AS dt ORDER BY 2;
```

### Pack with Default Options Output

The columns specified by TargetColumns are packed in the column packed\_data. Virtual columns are separated by commas, and each virtual column value is labeled with its column name. The input column sn, which was not specified by TargetColumns, is unchanged in the output table.

packed_data	sn
city:Nashville,state:Tennessee,period:2022-01-01 00:00:00,temp_f:35.1	1

packed_data	sn
city:Nashville,state:Tennessee,period:2022-01-01 01:00:00,temp_f:36.2	2
city:Nashville,state:Tennessee,period:2022-01-01 02:00:00,temp_f:34.5	3
city:Nashville,state:Tennessee,period:2022-01-01 03:00:00,temp_f:33.6	4
city:Nashville,state:Tennessee,period:2022-01-01 04:00:00,temp_f:33.1	5
city:Knoxville,state:Tennessee,period:2022-01-01 03:00:00,temp_f:33.2	6
city:Knoxville,state:Tennessee,period:2022-01-01 04:00:00,temp_f:32.8	7
city:Knoxville,state:Tennessee,period:2022-01-01 05:00:00,temp_f:32.4	8
city:Knoxville,state:Tennessee,period:2022-01-01 06:00:00,temp_f:32.2	9
city:Knoxville,state:Tennessee,period:2022-01-01 07:00:00,temp_f:32.4	10

## Example: Nondefault Options

This example specifies the pipe character (|) for Delimiter and 'false' for IncludeColumnName.

### Pack with Nondefault Options SQL Call

```
SELECT * FROM Pack (
    ON ville_temperature
    USING
    Delimiter ('|')
    OutputColumn ('packed_data')
    IncludeColumnName ('false')
    TargetColumns ('city', 'state', 'period', 'temp_f')
) AS dt ORDER BY 2;
```

### Pack with Nondefault Options Output

Virtual columns are separated by pipe characters and not labeled with their column names.

packed_data	sn
Nashville Tennessee 2022-01-01 00:00:00 35.1	1
Nashville Tennessee 2022-01-01 01:00:00 36.2	2
Nashville Tennessee 2022-01-01 02:00:00 34.5	3
Nashville Tennessee 2022-01-01 03:00:00 33.6	4
Nashville Tennessee 2022-01-01 04:00:00 33.1	5

packed_data	sn
Knoxville Tennessee 2022-01-01 03:00:00 33.2	6
Knoxville Tennessee 2022-01-01 04:00:00 32.8	7
Knoxville Tennessee 2022-01-01 05:00:00 32.4	8
Knoxville Tennessee 2022-01-01 06:00:00 32.2	9
Knoxville Tennessee 2022-01-01 07:00:00 32.4	10

## StringSimilarity

The StringSimilarity function calculates the similarity between two strings, using a specified comparison method.

---

### Note:

- When comparing strings, the function assumes that they are in the same Unicode script in Normalization Form C (NFC).
  - When used with this function, the ORDER BY clause supports only ASCII collation.
- 

### Function Information

- [StringSimilarity Syntax](#)
- [StringSimilarity Syntax Elements](#)
- [StringSimilarity Input](#)
- [StringSimilarity Output](#)
- [Examples: How to Use StringSimilarity](#)

## StringSimilarity Syntax

```
StringSimilarity (
    ON { table | view | (query) } [ PARTITION BY ANY ]
    USING
        ComparisonColumnPairs ('comparison_type (column1,column2[,constant])[ AS
output_column ]' [,...])
        [ CaseSensitive ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}[,...]) ]
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## StringSimilarity Syntax Elements

### ComparisonColumnPairs

Specify the names of the input table columns that contain strings to compare (*column1* and *column2*), how to compare them (*comparison\_type*), and (optionally) a constant and the name of the output column for their similarity (*output\_column*). The similarity is a value in the range [0, 1].

For *column1* and *column2*:

- If *column1* or *column2* includes any special characters (that is, characters other than letters, digits, or underscore (\_)), surround the column name with double quotation marks. For example, if *column1* and *column2* are c(col1) and c(col2), respectively, specify them as "c(col1)" and "c(col2)".

If *column1* or *column2* includes double quotation marks, replace each double quotation mark with a pair of double quotation marks. For example, if *column1* and *column2* are c1"c and c2"c, respectively, specify them as "c1""c" and "c2""c".

**Note:**

These rules do not apply to *output\_column*. For example, this is valid syntax:

```
ComparisonColumnPairs ('jaro ("c1""c", "c2""c") AS out"col')
```

- If *column1* or *column2* supports more than 200 characters, you can cast it to VARCHAR(200), as in the following example; however, the string may be truncated. For information about the CAST operation, see *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145.

```
SELECT * FROM StringSimilarity (
  ON (
    SELECT id, CAST(a AS VARCHAR(200)) AS a, CAST(b AS
VARCHAR(200)) AS b
    FROM max_varchar_strlen
  ) PARTITION BY ANY
  USING
  ComparisonColumnPairs ('ld(a,b) AS sim_fn')
```

```
Accumulate ('id')
) AS dt ORDER BY 1;
```

For *comparison\_type*, use one of these values:

<i>comparison_type</i>	Description
'jaro'	Jaro distance.
'jaro_winkler'	Jaro-Winkler distance: 1 for an exact match, 0 otherwise. If you specify this comparison type, you can specify the value of factor <i>p</i> with <i>constant</i> . $0 \leq p \leq 0.25$ . Default: <i>p</i> = 0.1
'n_gram'	<i>N</i> -gram similarity. If you specify this comparison type, you can specify the value of <i>N</i> with <i>constant</i> . Default: <i>N</i> = 2
'LD'	Levenshtein distance: Number of edits needed to transform one string into the other. Edits are insertions, deletions, or substitutions of individual characters.
'LDWS'	Levenshtein distance without substitution: Number of edits needed to transform one string into the other using only insertions or deletions of individual characters.
'OSA'	Optimal string alignment distance: Number of edits needed to transform one string into the other. Edits are insertions, deletions, substitutions, or transpositions of characters. A substring can be edited only once.
'DL'	Damerau-Levenshtein distance: Like 'OSA' except that a substring can be edited any number of times.
'hamming'	Hamming distance: For strings of equal length, number of positions where corresponding characters differ (that is, minimum number of substitutions needed to transform one string into the other). For strings of unequal length, -1.
'LCS'	Longest common substring: Length of longest substring common to both strings.
'jaccard'	Jaccard indexed-based comparison.
'cosine'	Cosine similarity.
'soundexcode'	Only for English strings: -1 if either string has a non-English character; otherwise, 1 if their soundex codes are the same and 0 otherwise.

The function ignores *constant* for every *comparison\_type* except 'jaro\_winkler' and 'n\_gram'.

You can specify a different *comparison\_type* for every pair of columns.

Default: *output\_column* is 'sim\_*i*', where *i* is the sequence number of the column pair.

**CaseSensitive**

[Optional] Specify whether string comparison is case-sensitive. You can specify either one value for all pairs or one value for each pair. If you specify one value for each pair, the *i*th value applies to the *i*th pair.

Default: 'false'

**Accumulate**

[Optional] Specify the names of input table columns to copy to the output table.

## StringSimilarity Input

Column	Data Type	Description
column1	CHARACTER or VARCHAR	String to compare to string in column2.
column2	CHARACTER or VARCHAR	String to compare to string in column1.
accumulate_column	Any	Column to copy to output table.

If any column1 or column2 in the input table schema supports more than 200 characters, you must cast it to VARCHAR(200). See example in [StringSimilarity Syntax Elements](#).

## StringSimilarity Output

### Output Table Schema

Column	Data Type	Description
accumulate_column	Any	Column copied from input table.
output_column	DOUBLE PRECISION	Similarity between strings in column pair.

## Examples: How to Use StringSimilarity

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [StringSimilarity with Specified Column Names](#)
- [StringSimilarity with Specified Column Ranges](#)

## StringSimilarity with Specified Column Names

### StringSimilarity Input

<b>id</b>	<b>src_text1</b>	<b>src_text2</b>	<b>tar_text</b>
1	astre	astter	aster
2	hone	fone	phone
3	acquiese	acquire	acquiesce
4	AAAACCCCCGGGGA	CCCGGGAACCAACC	CCAGGGAAACCCAC
5	alice	allen	allies
6	angela	angle	angels
7	senter	center	centre
8	chef	cheap	chief
9	circus	circle	circuit
10	debt	debut	debris
11	deal	dell	lead
12	bare	bear	bear

### StringSimilarity SQL Call with Specified Column Names

```

SELECT * FROM StringSimilarity (
    ON strsimilarity_input PARTITION BY ANY
    USING
        ComparisonColumnPairs ('jaro (src_text1, tar_text) AS jaro1_sim',
                               'LD (src_text1, tar_text) AS ld1_sim',
                               'n_gram (src_text1, tar_text, 2) AS ngram1_sim',
                               'jaro_winkler (src_text1, tar_text, 0.1) AS jw1_sim'
        )
    CaseSensitive ('true')
    Accumulate ('id', 'src_text1', 'tar_text')
) AS dt ORDER BY id;

```

## StringSimilarity Output with Specified Column Names

Columns 1-3

<b>id</b>	<b>src_text1</b>	<b>tar_text</b>
1	astre	aster
2	hone	phone
3	acquiese	acquiesce
4	AAAACCCCCGGGGA	CCAGGGAAACCCAC
5	alice	allies
6	angela	angels
7	senter	center
8	chef	chief
9	circus	circuit
10	debt	debris
11	deal	lead
12	bare	bear

Columns 4-7

<b>jaro1_sim</b>	<b>ld1_sim</b>	<b>ngram1_sim</b>	<b>jw1_sim</b>
0.933333333333333	0.6	0.5	0.953333333333333
0.933333333333333	0.8	0.75	0.933333333333333
0.925925925925926	0.7777777777777778	0.5	0.948148148148148
0.824175824175824	0.214285714285714	0.384615384615385	0.824175824175824
0.822222222222222	0.5	0.4	0.8577777777777778
0.888888888888889	0.833333333333333	0.8	0.933333333333333
0.822222222222222	0.5	0.4	0.822222222222222
0.933333333333333	0.8	0.5	0.9466666666666667
0.849206349206349	0.714285714285714	0.6666666666666667	0.90952380952381
0.75	0.5	0.4	0.825
0.6666666666666667	0.5	0.333333333333333	0.6666666666666667
0.833333333333333	0.5	0.333333333333333	0.85

## StringSimilarity with Specified Column Ranges

### StringSimilarity Input

<b>id</b>	<b>src_text1</b>	<b>src_text2</b>	<b>tar_text</b>
1	astre	astter	aster
2	hone	fone	phone
3	acquiese	acquire	acquiesce
4	AAAACCCCCGGGGA	CCCGGGAACCAACC	CCAGGGAAACCCAC
5	alice	allen	allies
6	angela	angle	angels
7	senter	center	centre
8	chef	cheap	chief
9	circus	circle	circuit
10	debt	debut	debris
11	deal	dell	lead
12	bare	bear	bear

### StringSimilarity SQL Call with Specified Column Ranges

```

SELECT * FROM StringSimilarity (
    ON strsimilarity_input PARTITION BY ANY
    USING
        ComparisonColumnPairs ('jaro (src_text1, tar_text) AS jaro1_sim',
                               'LD (src_text1, tar_text) AS ld1_sim',
                               'n_gram (src_text1, tar_text, 2) AS ngram1_sim',
                               'jaro_winkler (src_text1, tar_text, 0.1) AS jw1_sim'
        )
    CaseSensitive ('true')
    Accumulate ('[0:1]', 'tar_text')
) AS dt ORDER BY id;

```

## StringSimilarity Output with Specified Column Ranges

Columns 1-3

<b>id</b>	<b>src_text1</b>	<b>tar_text</b>
1	astre	aster
2	hone	phone
3	acquiese	acquiesce
4	AAAACCCCCGGGGA	CCAGGGAAACCCAC
5	alice	allies
6	angela	angels
7	senter	center
8	chef	chief
9	circus	circuit
10	debt	debris
11	deal	lead
12	bare	bear

Columns 4-7

<b>jaro1_sim</b>	<b>ld1_sim</b>	<b>ngram1_sim</b>	<b>jw1_sim</b>
0.933	0.6	0.5	0.953
0.933	0.8	0.75	0.933
0.926	0.778	0.5	0.948
0.824	0.214	0.385	0.824
0.822	0.5	0.4	0.858
0.889	0.833	0.8	0.933
0.822	0.5	0.4	0.822
0.933	0.8	0.5	0.947
0.849	0.714	0.667	0.91
0.75	0.5	0.4	0.825
0.667	0.5	0.333	0.667
0.833	0.5	0.333	0.85

## TD\_ConvertTo

TD\_ConvertTo converts the specified input table columns to specified data types to ensure that the data in the specified columns is in the correct format for efficient data processing. This function performs a conversion operation on the specified columns of an input table. This operation enables the user to convert the input table columns to the desired data types, allowing for consistency and compatibility with downstream operations.

TD\_ConvertTo follows this process:

1. Select the column for which you want to alter the data type.
2. Use TD\_ConvertTo to convert the data in the selected input table column into a specific data type.

### Function Information

- [TD\\_ConvertTo Syntax](#)
- [Required Syntax Elements for TD\\_ConvertTo](#)
- [Optional Syntax Elements for TD\\_ConvertTo](#)
- [TD\\_ConvertTo Input](#)
- [TD\\_ConvertTo Output](#)
- [TD\\_ConvertTo Usage Notes](#)
- [Example: How to Use TD\\_ConvertTo](#)

## TD\_ConvertTo Syntax

```
TD_ConvertTo (
    ON { table | view | (query) } AS InputTable
    USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        TargetDataType ('target_datatype' [,...])
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_ConvertTo

### ON clause

Accept the InputTable clause.

### TargetColumns

Specify the names of the InputTable columns to convert to another data type.

### TargetDataType

Specify either a single target data type for all target columns or a target data type for each target column. If you specify multiple target data types, the function assigns the *n*th target data type to the *n*th target column.

Allowed Values for the TargetDataType Element	Output Data Type
BYTEINT	BYTEINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	BIGINT
REAL	REAL
DECIMAL	DECIMAL ( <b>total_digits, precision</b> ) where you can specify the values up to 38 and 19 for the total_digits and precision parameters.
VARCHAR	<p>Depends on input data type:</p> <ul style="list-style-type: none"> <li>• Input Data Type: VARCHAR           <ul style="list-style-type: none"> <li>◦ Output Data Type: VARCHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.</li> </ul> </li> <li>• Input Data Type: CHAR           <ul style="list-style-type: none"> <li>◦ Output Data Type: VARCHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.</li> </ul> </li> <li>• Input Data Type: CLOB           <ul style="list-style-type: none"> <li>◦ Output Data Type: VARCHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC.</li> </ul> </li> <li>• Input Data Type: Other           <ul style="list-style-type: none"> <li>◦ Output Data Type: VARCHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.</li> </ul> </li> </ul>
VARCHAR(charlen= <i>len</i> , charset={LATIN   UNICODE}, casespecific={YES   NO})	VARCHAR( <i>len</i> ) with CHARACTER SET charset value, CASESPECIFIC case-specific value.

Allowed Values for the TargetDataType Element	Output Data Type
CHAR	<p>Depends on input data type:</p> <ul style="list-style-type: none"> <li>• Input Data Type: CHAR <ul style="list-style-type: none"> <li>◦ Output Data Type: CHAR with same CHARLEN, CHARACTER SET, and CASESPECIFIC values as input data type.</li> </ul> </li> <li>• Input Data Type: VARCHAR <ul style="list-style-type: none"> <li>◦ Output Data Type: CHAR(32000) with same CHARACTER SET and CASESPECIFIC values as input data type.</li> </ul> </li> <li>• Input Data Type: CLOB <ul style="list-style-type: none"> <li>◦ Output Data Type: CHAR(32000) with same CHARACTER SET as input data type, NOT CASESPECIFIC.</li> </ul> </li> <li>• Input Data Type: Other <ul style="list-style-type: none"> <li>◦ Output Data Type: CHAR(32000), CHARACTER SET UNICODE, NOT CASESPECIFIC.</li> </ul> </li> </ul>
CHAR(charlen= <i>len</i> , charset={LATIN   UNICODE}, casespecific={YES   NO})	CHAR( <i>len</i> ) with CHARACTER SET charset value, CASESPECIFIC case-specific value.
DATE	DATE FORMAT 'YYYY/MM/DD'
TIME	TIME(6)
TIMESTAMP	TIMESTAMP(6)
TIME WITH ZONE	TIME(6) WITH ZONE
TIMESTAMP WITH ZONE	TIMESTAMP(6) WITH ZONE
INTERVAL YEAR	INTERVAL YEAR(4)
INTERVAL MONTH	INTERVAL MONTH(4)
INTERVAL DAY	INTERVAL DAY(4)
INTERVAL HOUR	INTERVAL HOUR(4)
INTERVAL MINUTE	INTERVAL MINUTE(4)
INTERVAL SECOND	INTERVAL SECOND(4,6)
INTERVAL YEAR TO MONTH	INTERVAL YEAR(4) TO MONTH
INTERVAL DAY TO HOUR	INTERVAL DAY(4) TO HOUR
INTERVAL DAY TO MINUTE	INTERVAL DAY(4) TO MINUTE
INTERVAL DAY TO SECOND	INTERVAL DAY(4) TO SECOND(6)
INTERVAL HOUR TO MINUTE	INTERVAL HOUR(4) TO MINUTE
INTERVAL HOUR TO SECOND	INTERVAL HOUR(4) TO SECOND(6)

Allowed Values for the TargetDataType Element	Output Data Type
INTERVAL MINUTE TO SECOND	INTERVAL MINUTE(4) TO SECOND(6)
CLOB	Depends on input data type: <ul style="list-style-type: none"> <li>• Input Data Type: CLOB               <ul style="list-style-type: none"> <li>◦ Output Data Type: CLOB with same CHARLEN and CHARACTER SET value as input data type.</li> </ul> </li> <li>• Input Data Type: VARCHAR or CHAR               <ul style="list-style-type: none"> <li>◦ Output Data Type: CLOB(1048544000) with same CHARACTER SET as input data type.</li> </ul> </li> <li>• Input Data Type: Other               <ul style="list-style-type: none"> <li>◦ CLOB(1048544000), CHARACTER SET UNICODE.</li> </ul> </li> </ul>
CLOB(charlen= <i>len</i> , charset={LATIN   UNICODE})  <b>Note:</b> CLOB LATIN/UTF16 is only supported on the Block File System on the primary cluster. It is not available for the Object File System.	CLOB( <i>len</i> ) with CHARACTER SET charset value.
BYTE	BYTE(32000)
BYTE(charlen= <i>len</i> )	BYTE( <i>len</i> )
VARBYTE	VARBYTE(32000)
VARBYTE(charlen= <i>len</i> )	VARBYTE( <i>len</i> )
BLOB	BLOB(2097088000)
BLOB(charlen= <i>len</i> )	BLOB( <i>len</i> )
JSON	JSON(32000), CHARACTER SET UNICODE.
XML	XML(2097088000) INLINE LENGTH 4046

## Optional Syntax Elements for TD\_ConvertTo

### Accumulate

Specify the input table column names to copy to the output table.

## TD\_ConvertTo Input

### InputTable Schema

Column	Data Type	Description
<i>target_column</i>	Any	Column to convert to <i>target_datatype</i> .

## TD\_ConvertTo Output

### Output Table Schema

Column	Data Type	Description
<i>target_column</i>	<i>target_datatype</i>	Column converted to <i>target_datatype</i> .
<i>input_column</i>	Same as in InputTable	Column copied from InputTable.
AccumulateColumns	Any	The specified column names in the Accumulate element copied to the output table.

## TD\_ConvertTo Usage Notes

Data types define the kind of data that can be stored in a column, such as text, numbers, dates, or Boolean values. Converting a column's data type may be necessary to ensure that the data is in the appropriate format for analysis or modeling. This conversion is an important aspect of data cleaning that can improve the accuracy, consistency, and usability of the dataset. It also helps avoid errors and obtain more meaningful insights from the data.

For example, if a column contains dates stored as text, it may be necessary to convert the data type to a date format to perform date-based calculations or comparisons. Similarly, if a column contains numerical data stored as text, converting the data type to a numeric format can enable mathematical operations and aggregations.

Converting a column's data type can also help to detect and handle data inconsistencies or errors. For instance, if a column is expected to contain only numerical data, but some rows contain text values, converting the data type can identify these inconsistencies and either replace or remove the problematic values.

Conversion of data types may also allow for significant reduction in space taken up by the table (for example, an int column that contains 0 or 1 values, when converted to a byteint will save 15 bytes per row of storage space).

## Example: How to Use TD\_ConvertTo

### **TD\_ConvertTo InputTable: input\_table**

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare
cabin	embarked								
97	0	1	Goldschmidt; Mr. George B	male	71	0	0	PC 17754	
34.6542 A5	C								
488	0	1	Kent; Mr. Edward Austin	male	58	0	0	11771	29.7
B37	C								
505	1	1	Maioni; Miss. Roberta	female	16	0	0	110152	86.5
B79	S								
631	1	1	Barkworth; Mr. Algernon Henry Wilson	male	80	0	0	27042	30
A23	S								
873	0	1	Carlsson; Mr. Frans Olof	male	33	0	0	695	5
B51 B53 B55	S								

### **TD\_ConvertTo SQL Call**

```
SELECT * FROM TD_ConvertTo (
    ON input_table AS InputTable
    USING
        TargetColumns ('fare')
        TargetDataType ('integer')
) AS dt ORDER BY 1;
```

### **TD\_ConvertTo Output**

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare
cabin	embarked								
97	0	1	Goldschmidt; Mr. George B	male	71	0	0	PC 17754	34
A5	C								
488	0	1	Kent; Mr. Edward Austin	male	58	0	0	11771	29
B37	C								
505	1	1	Maioni; Miss. Roberta	female	16	0	0	110152	86
B79	S								
631	1	1	Barkworth; Mr. Algernon Henry Wilson	male	80	0	0	27042	30
A23	S								
873	0	1	Carlsson; Mr. Frans Olof	male	33	0	0	695	5
B51 B53 B55	S								

## **TD\_GetFutileColumns**

Data cleaning involves identifying and handling various types of issues in a dataset to ensure that it is accurate, complete, and consistent. One common issue is the presence of futile columns. These columns contain data that is not useful for the analysis or modeling process. This can include constant columns that contain the same value for all the rows in the dataset, unique identifier columns that do not provide any meaningful insights, redundant columns that provide the same information as other columns, or text columns that contain irrelevant or unstructured data. Removing these columns can help to simplify the analysis process, reduce the computational cost, and improve the accuracy of the analysis.

Removing futile columns is an important step in data cleaning. It helps to ensure the dataset only contains relevant and useful information. By removing these columns, analysts can:

- Focus their attention on the columns that contain the most useful information
- Avoid wasting time and resources on data that does not contribute to the analysis or modeling process
- Analyze and draw meaningful insights quicker

`TD_GetFutileColumns` function returns the futile column names if any of these conditions is met:

- If all values in the columns are unique
- If all the values in the columns are the same
- If the count of distinct values in the columns divided by the count of the total number of rows in the input table is greater than or equal to the specified threshold value

## Function Information

- [TD\\_GetFutileColumns Syntax](#)
- [Required Syntax Elements for TD\\_GetFutileColumns](#)
- [Optional Syntax Elements for TD\\_GetFutileColumns](#)
- [TD\\_GetFutileColumns Input](#)
- [TD\\_GetFutileColumns Output](#)
- [Example: How to Use TD\\_GetFutileColumns](#)

## TD\_GetFutileColumns Syntax

```
TD_GetFutileColumns(
    ON {table | view | (query)} AS InputTable PARTITION BY ANY
    ON {table | view | (query)} AS CategoryTable DIMENSION
    USING
        [ CategoricalSummaryColumn('target_column') ]
        [ ThresholdValue('threshold_value') ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_GetFutileColumns

### ON clause

Accepts the InputTable and CategoryTable clauses.

## Optional Syntax Elements for TD\_GetFutileColumns

### CategoricalSummaryColumn

Specifies the column name from the CategoricalSummaryTable generated using [TD\\_CategoricalSummary](#).

Default: ColumnName.

### ThresholdValue

Specifies the value up to which a particular column in input table is not considered futile. The value must be between 0 to 1.

Default: 0.95

## TD\_GetFutileColumns Input

### Input Table Schema

Column	Data Type	Description
Target_Column	VARCHAR	The input table columns from the Category Summary table.

### Categorical Summary Table Schema

Column	Data Type	Description
ColumnName	VARCHAR,CHARACTER SET UNICODE	The column name of the target column.
DistinctValue	VARCHAR,CHARACTER SET UNICODE	The distinct value in the target column.
DistinctValueCount	BIGINT	The count of distinct values in the target column.

## TD\_GetFutileColumns Output

### Output Table Schema

Column	Data Type	Description
FutileColumns	VARCHAR,CHARACTER SET UNICODE	The column names that are futile.

## Example: How to Use TD\_GetFutileColumns

### TD\_GetFutileColumns InputTable

```

passenger  gender ticket          cabin survived
-----  -----
 1 male    A/5 21171      C      0
 2 Female   PC 17599      C      1
 3 Female  STON/O2. 3101282 C      1
 4 male    113803       C      1
 5 Female  373450       C      0

```

### Creating Summary Table Using Columns

```

CREATE TABLE categorySummaryTable AS (
  SELECT * FROM TD_CATEGORICALSUMMARY (
    ON getFutileColumns_titanic AS InputTable
    USING
    TargetColumns('Cabin','gender','Ticket')
  ) AS dt)
WITH data;

```

### Output:

```

ColumnName DistinctValue  DistinctValueCount
-----  -----
cabin      C            5
gender     Female        3
gender     male          2
ticket    373450        1
ticket    A/5 21171      1
ticket    PC 17599        1
ticket    STON/O2. 3101282 1
ticket    113803        1

```

## Creating Summary Table Using Indexes

Another way to run the query shown in the preceding input table is to use indexing. Instead of writing column names, you can also use indexes. Indexes start from 0 (first column is indexed by 0) and then 1, and so on.

Running the following query gives you the stats for the first, second, and third columns:

```
CREATE TABLE categorySummaryTable AS (
    SELECT * FROM TD_CATEGORICALSUMMARY (
        ON getFutileColumns_titanic AS InputTable
        USING
        TargetColumns('[1:3]')
    ) AS dt)
WITH data;
```

### Output:

ColumnName	DistinctValue	DistinctValueCount
cabin	C	5
gender	Female	3
gender	male	2
ticket	373450	1
ticket	A/5 21171	1
ticket	PC 17599	1
ticket	STON/O2. 3101282	1
ticket	113803	1

### Note:

To include all columns, you can also use '[:]'. This method only works on categorical columns, so the dataset must have all categorical columns.

## TD\_GetFutileColumns SQL Call

```
SELECT * FROM TD_getFutileColumns(
    ON getFutileColumns_titanic AS InputTable PARTITION BY ANY
    ON categorySummaryTable AS categorytable DIMENSION
    USING
    CategoricalSummaryColumn('ColumnName')
    ThresholdValue(0.7)
)AS dt;
```

## TD\_GetFutileColumns Output Table

```
ColumnName
```

```
-----
```

```
ticket
```

```
cabin
```

## TD\_GetRowsWithoutMissingValues

TD\_GetRowsWithoutMissingValues displays the rows that have non-NULL values in specified input table columns.

A null value indicates the absence of information. This means that a real value is unknown or non-existent, such as no data is assigned to the column in that specific row. A null value is not a zero or a blank.

Null values can occur for reasons, such as incomplete data entry, data corruption, or errors in data processing. They can also occur when data is not collected or is not applicable for certain records or fields.

Null values can pose a challenge for data analysis and modeling because they can affect statistical calculations and analysis. There are advantages of removing null values from data, for example:

- Improved accuracy: Null values can skew analysis and modeling results, leading to inaccurate insights and decisions. By removing null values, you make sure your data is more accurate and reliable, which can lead to better business decisions.
- Enhanced efficiency: Null values can slow down data processing and analysis, leading to inefficiencies and longer processing times. By removing null values, you can streamline your data processing workflows and make your operations more efficient.
- Increased data quality: Null values can indicate missing or incomplete data, which can affect the overall quality of the data. By removing null values, you can improve the quality of your data, making it more useful and valuable for analysis and decision-making.
- Better customer insights: Null values in customer data can limit the ability to understand customer behavior and preferences. By removing null values, you can gain a more complete picture of your customers, enabling you to provide better products and services that meet your needs.

Therefore, it's important to identify and handle null values appropriately. Null values can be removed from a dataset using different techniques, depending on the nature and size of the dataset and the purpose of the analysis. Common techniques for removing null values include:

- Deleting rows with null values: This technique involves removing all rows that contain null values. While this approach is straightforward, this can result in a loss of data, especially if the null values are spread across multiple columns.
- Deleting columns with null values: This technique involves removing all columns that contain null values. While it retains the maximum number of rows, it can lead to a loss of information if the deleted columns are relevant to the analysis.

- Imputing null values: This technique involves filling in the null values with an appropriate value, such as the mean or median of the column. Imputing null values can help retain the maximum amount of data and reduce the bias introduced by deleting rows or columns.
- Using data modeling techniques: Advanced techniques such as regression, clustering, and decision trees can be used to predict the missing values based on other attributes in the dataset.

Overall, the best approach to remove null values depends on the specific needs and objectives of the analysis and the characteristics of the dataset itself.

#### **Important:**

Carefully consider the trade-offs between retaining maximum data, preserving data integrity, and minimizing the potential for bias or distortion.

#### **Note:**

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).  
For information about PTCs, see *International Character Set Support*, B035-1125.
- This function does not support KanjiSJIS or Graphic data types.
- SELECT TOP gives non-deterministic results. Therefore, identical queries including this instruction may produce different results.

## **Function Information**

- [TD\\_GetRowsWithoutMissingValues Syntax](#)
- [Required Syntax Elements for TD\\_GetRowsWithoutMissingValues](#)
- [Optional Syntax Elements for TD\\_GetRowsWithoutMissingValues](#)
- [TD\\_GetRowsWithoutMissingValues Input](#)
- [TD\\_GetRowsWithoutMissingValues Output](#)
- [Example: How to Use TD\\_GetRowsWithoutMissingValues](#)

#### **Related Information:**

[TD\\_GetRowsWithMissingValues](#)

## **TD\_GetRowsWithoutMissingValues Syntax**

```
TD_GetRowsWithoutMissingValues (
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY [ORDER BY
order_column ] ]
    [ USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
```

```
 ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_GetRowsWithoutMissingValues

**ON clause**

Specifies the table name, view name or query as an InputTable.

## Optional Syntax Elements for TD\_GetRowsWithoutMissingValues

**TargetColumns**

Specifies the target column names to check for non-null values.

Default: If omitted, the function considers all columns of the input table.

**Accumulate**

Specifies the input table column names to copy to the output table.

## TD\_GetRowsWithoutMissingValues Input

### **InputTable Schema**

Column	Data Type	Description
target_column	Any	Columns for which non-null values are checked.
accumulate_column	Any	The input table column names to copy to the output table.

## TD\_GetRowsWithoutMissingValues Output

### Output Table Schema

Column	Data Type	Description
target_column	Any	Columns for which non-null values are checked.
accumulate_column	Any	The input table column names to copy to the output table.

## Example: How to Use TD\_GetRowsWithoutMissingValues

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_GetRowsWithoutMissingValues InputTable: input\_table

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 23171	7.25	null	S
30	0	3	Todoroff, Mr. Lallio	male	null	0	0	349216	7.8958	null	S
505	1	1	Maiioni, Miss. Roberta	female	16	0	0	110152	86.5	B79	S
631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80	0	0	27042	30	A23	S
873	0	1	Carlsson, Mr. Frans Olof	male	33	0	0	695	5	B51 B53 B55	S

### Example: TD\_GetRowsWithoutMissingValues SQL Call

```
SELECT * FROM TD_getRowsWithoutMissingValues (
    ON input_table AS InputTable
    USING
    TargetColumns ('[name:cabin]')
) AS dt;
```

## TD\_GetRowsWithoutMissingValues Output Table

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
505	1	1	Maiioni, Miss. Roberta	female	16	0	0	110152	86.5	B79	S
631	1	1	Barkworth, Mr. Algernon Henry Wilson	male	80	0	0	27042	30	A23	S
873	0	1	Carlsson, Mr. Frans Olof	male	33	0	0	695	5	B51 B53 B55	S

## TD\_OutlierFilterFit

TD\_OutlierFilterFit function calculates the lower percentile, upper percentile, count of rows, and median for the specified input table columns. The calculated values for each column help the TD\_OutlierFilterTransform function detect outliers in the input table.

### Function Information

- [TD\\_OutlierFilterFit Syntax](#)
- [TD\\_OutlierFilterFit Syntax Elements](#)

- [TD\\_OutlierFilterFit Input](#)
- [TD\\_OutlierFilterFit Output](#)
- [Example: Using TD\\_OutlierFilterFit with Percentile Method](#)

## TD\_OutlierFilterFit Syntax

```
TD_OutlierFilterFit (
    ON { table | view | (query) } AS InputTable
    [OUT [ PERMANENT | VOLATILE ] TABLE OutputTable(output_table_name)]
    USING
        TargetColumns ({ 'target_column' | 'target_column_range' [, ...] })
        [GroupColumns ('group_column') ]
        [OutlierMethod ({ 'percentile' | 'tukey' | 'carling' })]
        [LowerPercentile (min_value)]
        [UpperPercentile (max_value)]
        [IQRMultiplier (k)]
        [ReplacementValue ({ 'delete' | 'null' | 'median' | replacement_value })]
        [RemoveTail ({ 'both' | 'upper' | 'lower' })]
        [PercentileMethod ({ 'PercentileCont' | 'PercentileDISC' })]
)

```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## TD\_OutlierFilterFit Syntax Elements

### TargetColumns

Specify the names of the numeric InputTable columns for which to compute metrics.

### GroupColumns

[Optional] Specify the name of the InputTable column by which to group the input data.

Default behavior: Function does not group input data.

### OutlierMethod

[Optional] Specify one of these methods for filtering outliers:

Method	Values Outside This Range Are Outliers
percentile (default method)	[ <i>min_value</i> , <i>max_value</i> ].
tukey	[Q1 - $k^*(Q3-Q1)$ , Q1 + $k^*(Q3-Q1)$ ] where: Q1 = 25th quartile of data Q3 = 75th quartile of data $k$ = interquartile range multiplier (see IQRMultiplier)
carling	$Q2 \pm c^*(Q3-Q1)$ where: Q2 = median of data Q1 = 25th quartile of data Q3 = 75th quartile of data $c = (17.63^*r - 23.64) / (7.74^*r - 3.71)$ $r$ = count of rows in <i>group_column</i> if you specify GroupColumns, otherwise count of rows in InputTable

### LowerPercentile

[Optional] Specify a lower range of percentile to use to detect whether the value is an outlier. Value 0 to 1 is supported. For Tukey and Carling, use 0.25 as the lower percentile. The default value is 0.05.

### UpperPercentile

[Optional] Specify a upper range of percentile to use to detect whether the value is an outlier. Value 0 to 1 is supported. For Tukey and Carling, use 0.75 as the upper percentile. The default value is 0.95.

### IQRMultiplier

[Optional] Specify interquartile range multiplier (IQR),  $k$ , for Tukey filtering.

The IQR is an estimate of the spread (dispersion) of the data in the target columns (IQR =  $|Q3-Q1|$ ).

Use  $k = 1.5$  for moderate outliers and  $k = 3.0$  for serious outliers.

Default: 1.5

### ReplacementValue

[Optional] Specify how to handle outliers:

Option	Description
delete (default value)	Do not copy row to output table.
null	Copy row to output table, replacing each outlier with NULL.
median	Copy row to output table, replacing each outlier with median value for its group.
<i>replacement_value</i> (Must be numeric.)	Copy row to output table, replacing each outlier with <i>replacement_value</i> .

**RemoveTail**

[Optional] Specify whether to remove the upper tail, the lower tail, or both.

Default: both

**PercentileMethod**

[Optional] Specify either the PercentileCont or the PercentileDISC method for calculating the upper and lower percentiles of the input data values. The default value is PercentileDISC.

**TD\_OutlierFilterFit Input****InputTable Schema**

Column	Data Type	Description
target_column	NUMERIC	The input table column names for computing metrics and filtering outliers using the <a href="#">TD_OutlierFilterTransform</a> function.
group_column	Any	[Optional] Column by which to group input data.

**TD\_OutlierFilterFit Output****FitTable (*fit\_table*) Schema**

Column	Data Type	Description
TD_OutlierMethod_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of OutlierMethod ('percentile', 'tukey', or 'carling').
group_column	Any	[Column appears only if you specify GroupColumns.] Column by which input data is grouped.
TD_IQRMultiplier_OFTFIT	NUMERIC	Value of IQRMultiplier ( <i>k</i> ).

Column	Data Type	Description
TD_RemoveTail_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of RemoveTail ('both', 'upper', or 'lower').
TD_ReplacementValue_ OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of ReplacementValue ('delete', 'null', 'median', or replacement_value).
TD_MinThreshold_OFTFIT	NUMERIC	Value of LowerPercentile (min_value).
TD_MaxThreshold_OFTFIT	NUMERIC	Value of UpperPercentile (max_value).
TD_AttributeValue_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	[Column appears once for each specified target_column.] 11/14/2023 What is the description for this entry?
TD_CountValue_OFTFIT	NUMERIC	Count of rows in group_column if you specify GroupColumns, otherwise count of rows in input table.
TD_MedianValue_OFTFIT	NUMERIC	Median values for target columns.
TD_LowerPercentile_OFTFIT	NUMERIC	Lower percentile of input data values, calculated by method specified by PercentileMethod.
TD_UpperPercentile_OFTFIT	NUMERIC	Upper percentile of input data values, calculated by method specified by PercentileMethod.

## Example: Using TD\_OutlierFilterFit with Percentile Method

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_OutlierFilterFit InputTable: titanic

passenger	pclass	fare	survived
1	3	7.250000000	0
2	1	71.283300000	1
3	3	7.925000000	1
4	1	53.100000000	1
5	3	8.050000000	0

## TD\_OutlierFilterFit SQL Call

```
CREATE TABLE outlier_fit AS (
    SELECT * FROM TD_OutlierFilterFit (
        ON titanic AS InputTable
        OUT TABLE OutputTable (outlier_fit)
        USING
            TargetColumns ('Fare')
            LowerPercentile (0.1)
            UpperPercentile (0.9)
            OutlierMethod ('Percentile')
            ReplacementValue ('median')
            PercentileMethod ('PercentileCont')
    ) AS dt
    WITH DATA;
```

## TD\_OutlierFilterFit Output

TD_OUTLIERMETHOD_OFTFIT	TD_IQRMULTIPLIER_OFTFIT	TD_REMOVETAIL_OFTFIT	TD_REPLACEMENTVALUE_OFTFIT	TD_MINTHRESHOLD_OFTFIT	TD_MAXTHRESHOLD_OFTFIT	TD_ATTRIBUTEVAL_OFTFIT
PERCENTILE	1.500000000	BOTH	MEDIAN	0.100000000	0.900000000	fare

## TD\_OutlierFilterTransform

TD\_OutlierFilterTransform filters outliers from the input table. The metrics for determining outliers come from [TD\\_OutlierFilterFit](#) output.

Outlier filtering is a technique used to identify and remove outliers from a dataset in machine learning pipelines. The simple method of filtering outliers would be to calculate the 25th and 75th percentiles of the data, and removing the lower and upper values from them respectively. The formulas are:

$$25\text{th percentile} = 0.25*(N+1)$$

$$75\text{th percentile} = 0.75*(N+1)$$

where

$N$  = number of data points

These measures provide information about the central tendency and spread of the data, respectively. While they can be useful in summarizing the distribution of the data, they do not provide information about extreme values that may be far from the median. Therefore, use more advanced methods like Tukey and Carling. Here is an example guide on how to apply these methods to filter outliers:

1. Calculate the Inter-quartile range (IQR) of the dataset: The IQR is calculated by subtracting the 25th percentile from the 75th percentile of the data.
2. Calculate the upper and lower bounds for outliers using the Tukey or Carling method:

- For the Tukey method, the upper bound is calculated by adding 1.5 times the IQR to the 75th percentile, while the lower bound is calculated by subtracting 1.5 times the IQR from the 25th percentile.
  - For the Carling method, the upper bound is calculated by adding 3 times the IQR to the 75th percentile, while the lower bound is calculated by subtracting 3 times the IQR from the 25th percentile.
3. Identify the outliers: Any data point that falls outside the upper and lower bounds is considered an outlier.
  4. Filter the outliers: Once the outliers have been identified, they can be removed from the dataset or replaced with another value using the methods discussed earlier.

Consider the following array of 10 data points:

[10, 12, 15, 17, 20, 22, 25, 30, 40, 100]

1. The 25th percentile is 13.5 and the 75th percentile is 27.5. Therefore, the IQR is 14.
2. The upper bound is 1.5 times the IQR added to the 75th percentile, which is  $27.5 + (1.5 * 14) = 48.5$ . The lower bound is 1.5 times the IQR subtracted from the 25th percentile, which is  $13.5 - (1.5 * 14) = -4.5$ .
3. Any data point that falls outside the upper and lower bounds is considered an outlier. In this case, the data point 100 falls outside the upper bound and is therefore an outlier.
4. The outlier can be removed from the dataset or replaced with mean, median, or other suitable values according to the type of data. In this case, remove the outlier to obtain the filtered dataset:

[10, 12, 15, 17, 20, 22, 25, 30, 40]

The Tukey or Carling methods depend on the specific characteristics of the dataset. The Tukey method is more conservative and may result in fewer outliers being identified, while the Carling method is more sensitive and may identify more outliers.

## Function Information

- [TD\\_OutlierFilterTransform Syntax](#)
- [Required Syntax Elements for TD\\_OutlierFilterTransform](#)
- [Optional Syntax Elements for TD\\_OutlierFilterTransform](#)
- [TD\\_OutlierFilterTransform Input](#)
- [TD\\_OutlierFilterTransform Output](#)
- [Example: How to Use TD\\_OutlierFilterTransform](#)

## TD\_OutlierFilterTransform Syntax

### TD\_OutlierFilterTransform Syntax Using Partition by Any

```
TD_OutlierFilterTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  ON { table | view | (query) } AS FitTable DIMENSION
)
```

## **TD\_OutlierFilterTransform Syntax Using Partition by Key**

```
TD_OutlierFilterTransform (
  ON { table | view | (query) } AS InputTable PARTITION BY group_column
  ON { table | view | (query) } AS FitTable PARTITION BY group_column
)
```

### **Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## **Required Syntax Elements for TD\_OutlierFilterTransform**

### **ON clause**

Specifies the table name, view name, or query as an InputTable and FitTable.

## **Optional Syntax Elements for TD\_OutlierFilterTransform**

All syntax elements are required.

## **TD\_OutlierFilterTransform Input**

### **InputTable Schema**

Column	Data Type	Description
target_column	NUMERIC	Input table column names for computing metrics and filtering outliers.
group_column	Any	[Optional] Column to group input data.

### **FitTable Schema**

Column	Data Type	Description
TD_OutlierMethod_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of OutlierMethod ('percentile', 'tukey', or 'carling').
group_column	Any	[Column appears only if you specify GroupColumns.] Column that input data is grouped.

Column	Data Type	Description
TD_IQRMultiplier_OFTFIT	NUMERIC	Value of IQRMultiplier ( $k$ ).
TD_RemoveTail_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of RemoveTail ('both', 'upper', or 'lower').
TD_ReplacementValue_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	Value of ReplacementValue ('delete', 'null', 'median', or replacement_value).
TD_MinThreshold_OFTFIT	NUMERIC	Value of LowerPercentile (min_value).
TD_MaxThreshold_OFTFIT	NUMERIC	Value of UpperPercentile (max_value).
TD_AttributeValue_OFTFIT	VARCHAR (CHARACTER SET UNICODE)	[Column appears once for each specified target_column.]
TD_CountValue_OFTFIT	NUMERIC	Count of rows in group_column if you specify GroupColumns, otherwise count of rows in input table.
TD_MedianValue_OFTFIT	NUMERIC	Median values for target columns.
TD_LowerPercentile_OFTFIT	NUMERIC	Lower percentile of input data values, calculated by method specified by PercentileMethod (PercentileCont or PercentileDISC).
TD_UpperPercentile_OFTFIT	NUMERIC	Upper percentile of input data values, calculated by method specified by PercentileMethod.

## TD\_OutlierFilterTransform Output

Column	Data Type	Description
target_column	NUMERIC	Column with the computed metrics.
OtherColumns	Any	The columns from the input table excluding the target columns are displayed.

## Example: How to Use TD\_OutlierFilterTransform

### TD\_OutlierFilterTransform Input

```

passenger  pclass fare          survived
----- -----
 1       3   7.250000000      0
 2       1  71.283300000      1

```

3	3	7.925000000	1
4	1	53.100000000	1
5	3	8.050000000	0

## TD\_OutlierFilterTransform SQL Call

```
SELECT * FROM TD_OutlierFilterTransform (
    ON titanic AS InputTable PARTITION BY ANY
    ON outlier_fit AS FitTable DIMENSION
) AS dt;
```

## TD\_OutlierFilterTransform Output

passenger	pclass	fare	survived
1	3	8.050000000	0
2	1	8.050000000	1
3	3	7.925000000	1
4	1	53.100000000	1
5	3	8.050000000	0

## TD\_SimpleImputeFit

TD\_SimpleImputeFit is a data cleaning function that assigns missing values in a dataset. When working with real-world data, it is not uncommon to have missing values in some of the variables. This can cause problems when trying to perform data analysis or modeling. In such cases, it may be necessary to impute the missing values with plausible estimates to ensure that the data is complete and suitable for further analysis.

The TD\_SimpleImputeFit function replaces the missing values with the mean, median, or most frequent value of the feature, depending on the imputation strategy selected. The function can also handle numeric and categoric features, and can be used with machine-learning models.

For example, you have a dataset of patient records, including variables such as age, gender, blood pressure, and cholesterol levels. However, due to some missing values in the dataset, some patients' records are incomplete. The blood pressure variable is missing for some patients, while the cholesterol levels variable is missing for others. This can cause problems when trying to perform data analysis or modeling on the dataset.

To address this issue, use the TD\_SimpleImputeFit function to assign the missing values with plausible estimates. You can impute missing values in the systolic blood pressure with the mean value of all patients in the dataset. Similarly, you can impute missing values in the cholesterol levels variable with the median value of the cholesterol levels of all patients in the dataset.

TD\_SimpleImputeFit outputs a table of values to substitute for missing values in the input table. The output table is input to [TD\\_SimpleImputeTransform](#), which makes the substitutions.

## Function Information

- [TD\\_SimpleImputeFit Syntax](#)
- [Required Syntax Elements for TD\\_SimpleImputeFit](#)
- [Optional Syntax Elements for TD\\_SimpleImputeFit](#)
- [TD\\_SimpleImputeFit Input](#)
- [TD\\_SimpleImputeFit Output](#)
- [Example: How to Use TD\\_SimpleImputeFit](#)

## TD\_SimpleImputeFit Syntax

```
TD_SimpleImputeFit (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
    USING
    { { literal_specification | stats_specification } |
        literal_specification
        stats_specification
    }
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_SimpleImputeFit

### ON clause

Accept the InputTable clause.

### *literal\_specification*

```
ColsForLiterals ({'literal_column' | literal_column_range } [, ...])
Literals ('literal' [, ...])
```

***stats\_specification***

```
    ColsForStats ({'stats_column' | stats_column_range } [, ...])
    Stats ('statistic' [, ...])
    [ PartitionColumn ('partition_column') ]
```

## Optional Syntax Elements for TD\_SimpleImputeFit

**OUT clause**

Accept the OutputTable clause.

**ColsForLiterals**

Specify the names of the InputTable columns that have missing values to replace with specified literal values.

**Literals**

Specify the literal values to substitute for missing values in the columns specified by ColsForLiterals. A literal must not exceed 128 characters.

The function maps each literal to the column in the same position in ColsForLiterals. For example, ColsForLiterals ('[1:5]', '-[2:4]', '[3]') specifies the column with index 3 last, so the function maps the last specified literal to it.

**ColsForStats**

Specify the names of the InputTable columns that have missing values to replace with specified statistics.

**Stats**

Specify the statistics to substitute for missing values in the columns specified by ColsForStats.

For numeric columns, the value of the Stats argument must be one of the following values:

- MIN
- MAX
- MEAN
- MEDIAN

For columns with the following data types, the value of the Stats argument can be MODE:

- CHARACTER
- VARCHAR
- BYTEINT

- SMALLINT
- INTEGER

CHARACTER and VARCHAR values must not exceed 128 characters.

The output of MODE is the value that appeared in the last according to the alphabetical order.

The function maps the value of each Stats argument to the column in the same position in ColsForStats. For example, ColsForStats ('[1:5]', '-[2:4]', '[3]') specifies the column with index 3 last, so the function maps the last specified Stats argument to it.

#### **PartitionColumn**

Specify the name of the InputTable column to partition the input.

Default behavior: The function treats all rows as a single partition.

## **TD\_SimpleImputeFit Input**

#### **InputTable Schema**

Column	Data Type	Description
target_column	CHAR, VARCHAR (with CHARACTER SET LATIN or UNICODE) or NUMERIC	Column to find missing values.

## **TD\_SimpleImputeFit Output**

#### **Output Table Schema**

Column	Data Type	Description
TD_INDEX_SIMFIT	INTEGER	Unique row identifier.
TD_TARGETCOLUMN_SIMFIT	VARCHAR (CHARACTER SET UNICODE)	Target column name (literal_column or stats_column).
TD_NUM_COLVAL_SIMFIT	NUMERIC	<ul style="list-style-type: none"> <li>If column is numeric, value substituted for missing value.</li> <li>Otherwise NULL.</li> </ul>
TD_STR_COLVAL_SIMFIT	VARCHAR (CHARACTER SET UNICODE)	<ul style="list-style-type: none"> <li>If column is nonnumeric, value substituted for missing value or MODE value of column.</li> <li>Otherwise NULL.</li> </ul>
TD_ISNUMERIC_SIMFIT	BYTEINT	<ul style="list-style-type: none"> <li>1 if column is numeric.</li> <li>Otherwise 0.</li> </ul>

Column	Data Type	Description
partition_column	Same as in InputTable	Column that input is partitioned.

## Example: How to Use TD\_SimpleImputeFit

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_SimpleImputeFit InputTable: simpleimputefit\_input

passenger	pclass	gender	fare	survived
1	3	male	725.320	0
2	1	female	712.250	1
3	null	female	null	1
4	1	null	531.780	1
5	3	male	805.210	0

### TD\_SimpleImputeFit SQL Call

```
CREATE TABLE fit_table AS (
  SELECT * FROM TD_SimpleImputeFit (
    ON simpleimputefit_input AS InputTable
    USING
      ColsForLiterals ('Pclass')
      Literals ('2')
      ColsForStats ('Gender', 'Fare')
      Stats ('mode', 'median')
  ) AS dt
) WITH DATA;
```

### TD\_SimpleImputeFit Output

TD_INDEX_SIMFIT	TD_TARGETCOLUMN_SIMFIT	TD_NUM_COLVAL_SIMFIT	TD_STR_COLVAL_SIMFIT	TD_ISNUMERIC_SIMFIT
1	pclass	2.000	null	1
2	gender	null	male	0
3	fare	718.785	null	1

## TD\_SimpleImputeTransform

TD\_SimpleImputeTransform substitutes specified values for missing values in the input table. The specified values come from [TD\\_SimpleImputeFit](#) output.

The TD\_SimpleImputeTransform function is a transformer for handling missing data in Teradata tables or views. TD\_SimpleImputerTransform can be used to impute missing values with reasonable estimates, which ensures the resulting analysis or modeling is more accurate and reliable. This function is used with output from the [TD\\_SimpleImputeFit](#) function.

It can be applied to columns in a table or view that have missing values. TD\_SimpleImputeTransform can replace these missing values with a variety of imputation strategies, including mean, median, most frequent, and constant values.

For example, suppose you have a dataset of customer information where each row represents a customer and columns represent various features such as age, income, gender, education level, etc. Some of the customers did not provide their income information during the data collection process, resulting in missing values in the income column. If you remove these rows with missing values, you will lose valuable information about these customers and potentially bias the analysis.

Use TD\_SimpleImputeTransform to impute the missing values in the income column. For instance, you can use the mean or median income of the non-missing values in the same column to fill in the missing values. This imputed data helps to provide a complete picture of the dataset to perform further analysis, such as identifying income distribution among different genders, education levels, or age groups.

## Function Information

- [TD\\_SimpleImputeTransform Syntax](#)
- [Required Syntax Elements for TD\\_SimpleImputeTransform](#)
- [Optional Syntax Elements for TD\\_SimpleImputeTransform](#)
- [TD\\_SimpleImputeTransform Input](#)
- [TD\\_SimpleImputeTransform Output](#)
- [Example: How to Use TD\\_SimpleImputeTransform](#)

## TD\_SimpleImputeTransform Syntax

### TD\_SimpleImputeTransform without partition\_column

```
TD_SimpleImputeTransform (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    ON { table | view | (query) } AS FitTable DIMENSION
)
```

### TD\_SimpleImputeTransform with partition\_column

```
TD_SimpleImputeTransform (
    ON { table | view | (query) } AS InputTable PARTITION BY partition_column
    ON { table | view | (query) } AS FitTable PARTITION BY partition_column
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_SimpleImputeTransform

### ON clause

Accept the InputTable and FitTable clauses.

## Optional Syntax Elements for TD\_SimpleImputeTransform

All syntax elements are required.

## TD\_SimpleImputeTransform Input

### InputTable Schema

Column	Data Type	Description
target_column	CHAR, VARCHAR, or (with CHARACTER SET LATIN or UNICODE) or NUMERIC	Column to find missing values. Column is in the output from <a href="#">TD_SimpleImputeFit</a> .

### FitTable Schema

Column	Data Type	Description
TD_INDEX_SIMFIT	INTEGER	Unique row identifier.
TD_TARGETCOLUMN_SIMFIT	VARCHAR (CHARACTER SET UNICODE)	Target column name (literal_column or stats_ column). Column is in the output from <a href="#">TD_SimpleImputeFit</a> .
TD_NUM_COLVAL_SIMFIT	NUMERIC	<ul style="list-style-type: none"> <li>If column is numeric, value substituted for missing value.</li> <li>Otherwise NULL.</li> </ul>
TD_STR_COLVAL_SIMFIT	VARCHAR (CHARACTER SET UNICODE)	<ul style="list-style-type: none"> <li>If column is nonnumeric, value substituted for missing value or MODE value of column.</li> <li>Otherwise NULL.</li> </ul>

Column	Data Type	Description
TD_ISNUMERIC_SIMFIT	BYTEINT	<ul style="list-style-type: none"> <li>• 1 if column is numeric.</li> <li>• Otherwise 0.</li> </ul>
partition_column	Same as in InputTable	Column on which input is partitioned.

## TD\_SimpleImputeTransform Output

### Output Table Schema

Column	Data Type	Description
target_column	Same as in InputTable	Column in which missing values have been replaced.

## Example: How to Use TD\_SimpleImputeTransform

### TD\_SimpleImputeTransform Input Table

passenger	pclass	gender	fare	survived
1	3	male	725.320000000	0
2	1	female	712.250000000	1
3	null	female	null	1
4	1	null	531.780000000	1
5	3	male	805.210000000	0

### Output Table from TD\_SimpleImputeFit Function

TD_INDEX_SIMFIT	TD_TARGETCOLUMN_SIMFIT	TD_NUM_COLVAL_SIMFIT	TD_STR_COLVAL_SIMFIT	TD_ISNUMERIC_SIMFIT
1	pclass	2.000000000	null	1
2	gender	null	male	0
3	fare	718.785000000	null	1

### Example: TD\_SimpleImputeTransform SQL Call

```
SELECT * FROM TD_SimpleImputeTransform (
    ON simpleimputefit_input AS InputTable
    ON fit_table AS FitTable DIMENSION
) AS dt;
```

### TD\_SimpleImputeTransform Output

passenger	pclass	gender	fare	survived
-----------	--------	--------	------	----------

5	3	male	805.210000000	0
4	1	male	531.780000000	1
3	2	female	718.785000000	1
1	3	male	725.320000000	0
2	1	female	712.250000000	1

## Unpack

The Unpack function unpacks data from a single packed column into multiple columns. The packed column is composed of multiple virtual columns, which become the output columns. To determine the virtual columns, the function must have either the delimiter that separates them in the packed column or their lengths.

Use the Unpack function to unpack data from a source column into multiple target columns. It is commonly used to separate a single column of data into multiple columns based on a specific delimiter or separator. The Unpack function complements the [Pack](#) function, but you can use it on any packed column that meets the input requirements.

For example, if you have a column containing a full name, you can use unpack function to parse the name into separate columns for first name and last name.

You can also use the Unpack function to extract specific characters or substrings from a column based on a particular position or length. This is useful when working with data that is stored in a fixed-width format. The Unpack function is a tool for manipulating and extracting data. Its flexibility and ease of use make it a valuable tool for data analysts and database administrators.

### Function Information

- [Unpack Syntax](#)
- [Required Syntax Elements for Unpack](#)
- [Optional Syntax Elements for Unpack](#)
- [Unpack Input](#)
- [Unpack Output](#)
- [Examples: How to Use Unpack](#)

## Unpack Syntax

```
Unpack (
    [ ON { table | view | (query) } AS InputTable ]
    USING
        TargetColumn ('target_column')
        OutputColumns ('output_column' [,...])
        OutputDataTypes ('datatype' [,...])
        [ Delimiter ('delimiter') ]
        [ ColumnLength ('column_length' [,...] ) ]
```

```
[ Regex ('regular_expression') ]
[ RegexSet ('group_number') ]
[ IgnoreInvalid ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for Unpack

### **TargetColumn**

Specifies the name of the input column that contains the packed data.

### **OutputColumns**

Specifies the name for the output columns, in the order in which the corresponding virtual columns appear in target column. The names must be valid object names, as defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

If you specify fewer output column names than there are virtual input columns, the function ignores the extra virtual input columns. That is, if the packed data contains  $x+y$  virtual columns and the OutputColumns syntax element specifies  $x$  output column names, the function assigns the names to the first  $x$  virtual columns and ignores the remaining  $y$  virtual columns.

### **OutputDataTypes**

Specifies the data types of the unpacked output columns. Supported data types are VARCHAR, INTEGER, DOUBLE PRECISION, TIME, DATE, and TIMESTAMP.

If OutputDataTypes specifies only one value and OutputColumns specifies multiple columns, the specified value applies to every output\_column.

If OutputDataTypes specifies multiple values, it must specify a value for each output\_column. The  $n$ th datatype corresponds to the  $n$ th output\_column.

The function can output only 16 VARCHAR columns.

## Optional Syntax Elements for Unpack

### ON clause

Accepts the InputTable clause.

### Delimiter

Specifies the delimiter, a single Unicode character in Normalization Form C (NFC), that separates the virtual columns in the packed data. The *delimiter* is case-sensitive.

---

**Note:**

Do not specify both this and ColumnLength. If the virtual columns are separated by a delimiter, specify the delimiter with this syntax element; otherwise, specify the ColumnLength syntax element.

---

Default: ',' (comma)

### ColumnLength

Specifies the length of the virtual columns; to use this syntax element, you must know the length of each virtual column.

If ColumnLength specifies only one value and OutputColumns specifies multiple columns, the specified value applies to every *output\_column*.

If ColumnLength specifies multiple values, it must specify a value for each *output\_column*. The *n<sup>th</sup> datatype* corresponds to the *n<sup>th</sup> output\_column*. However, the last *output\_column* can be an asterisk (\*), which represents a single virtual column that contains the remaining data. For example, if the first three virtual columns have the lengths 2, 1, and 3, and all remaining data belongs to the fourth virtual column, you can specify ColumnLength ('2', '1', '3', \*).

---

**Note:**

Do not specify both this and the Delimiter syntax.

### Regex

Specifies a regular expression that describes a row of packed data, enabling the function to find the data values.

A row of packed data contains a data value for each virtual column, but the row might also contain other information (such as the virtual column name). In the regular expression, each data value is enclosed in parentheses.

For example, a packed data has two virtual columns, age and gender, and that one row of packed data is age:34,gender:male. The regular expression that describes the row is

'.\*:(.\*)'. The .\*: matches the virtual column names, age and gender, and the (.\* ) matches the values, 34 and male.

To represent multiple data groups in regular expression, use multiple pairs of parentheses. Without parentheses, the last data group in regular expression represents the data value (other data groups are assumed to be virtual column names or unwanted data). If a different data group represents the data value, specify its group number with the RegexSet syntax element.

Default: '(.\* )', matches the whole string (between delimiters, if any). When applied to the preceding sample row, the default regular expression causes the function to return 'age:34' and 'gender:male' as data values.

### **RegexSet**

Specifies the ordinal number of the data group in regular expression that represents the data value in a virtual column.

Default behavior: The last data group in regular expression represents the data value. For example, a regular expression is '([a-zA-Z]\*):(.\* )'. If group number is '1', '([a-zA-Z]\*)' represents the data value. If group number is '2', '(.\* )' represents the data value.

Maximum: 30

### **IgnoreInvalid**

Specifies whether the function ignores rows that contain invalid data.

IgnoreInvalid may not behave as you expect if an item in a virtual column has trailing special characters. See [Example: Input Columns with Trailing Special Characters](#).

Default: 'false' (The function fails if it encounters a row with invalid data.)

### **Accumulate**

Specifies the input columns to copy to the output table.

## **Unpack Input**

### **Input Table Schema**

Column	Data Type	Description
target_column	CHARACTER, VARCHAR, or CLOB	Packed data.
accumulate_column or other_input_column	Any	[Column appears zero or more times.] Column to copy to output table. Typically, one such column contains row identifiers.

## Unpack Output

### Output Table Schema

Column	Data Type	Description
output_column	Specified by OutputDataTypes syntax element.	Unpacked column.
accumulate_column	Any	Column copied from input table.
other_input_column	Any	[Column does not appear when using Accumulate. Appears once for each input table column not specified by TargetColumn syntax element.] Column copied from input table.

## Examples: How to Use Unpack

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Example: Delimiter Separates Virtual Columns](#)
- [Example: No Delimiter Separates Virtual Columns](#)
- [Example: More Input Columns than Output Columns](#)
- [Example: Input Columns with Trailing Special Characters](#)

### Example: Delimiter Separates Virtual Columns

#### Unpack Input

The input table, ville\_tempdata, is a collection of temperature readings for two cities, Nashville and Knoxville, in the state of Tennessee. In the column of packed data, the delimiter comma (,) separates the virtual columns. The last row contains invalid data.

**ville\_tempdata**

sn	packed_temp_data
10	Nashville,Tennessee,35.1
11	Nashville,Tennessee,36.2
12	Nashville,Tennessee,34.5
13	Nashville,Tennessee,33.6
14	Nashville,Tennessee,33.1

sn	packed_temp_data
15	Nashville,Tennessee,33.2
16	Nashville,Tennessee,32.8
17	Nashville,Tennessee,32.4
18	Nashville,Tennessee,32.2
19	Nashville,Tennessee,32.4
20	Thisisbaddata

## Unpack SQL Call with Delimiter Separates Virtual Columns

Because comma is the default delimiter, the Delimiter syntax element is optional.

```
SELECT * FROM Unpack (
    ON ville_tempdata AS InputTable
    USING
        TargetColumn ('packed_temp_data')
        OutputColumns ('city', 'state', 'temp_f')
        OutputDataTypes ('varchar', 'varchar', 'real')
        Delimiter (',')
        Regex ('(.*)')
        RegexSet (1)
        IgnoreInvalid ('true')
        Accumulate ('sn')
) AS dt ORDER BY sn;
```

## Unpack Output with Delimiter Separates Virtual Columns

Because of IgnoreInvalid ('true'), the function did not fail when it encountered the row with invalid data, but it did not output that row.

city	state	temp_f	sn
Nashville	Tennessee	3.51000000000000E 001	10
Nashville	Tennessee	3.62000000000000E 001	11
Nashville	Tennessee	3.45000000000000E 001	12
Nashville	Tennessee	3.36000000000000E 001	13
Nashville	Tennessee	3.31000000000000E 001	14
Nashville	Tennessee	3.32000000000000E 001	15
Nashville	Tennessee	3.28000000000000E 001	16

city	state	temp_f	sn
Nashville	Tennessee	3.24000000000000E 001	17
Nashville	Tennessee	3.22000000000000E 001	18
Nashville	Tennessee	3.24000000000000E 001	19

## Example: No Delimiter Separates Virtual Columns

### Unpack Input

The input table is ville\_tempdata1 with no delimiter that separates the virtual columns in the packed data. To enable the function to determine the virtual columns, the function call specifies the column lengths.

ville_tempdata1	
sn	packed_temp_data
10	NashvilleTennessee35.1
11	NashvilleTennessee36.2
12	NashvilleTennessee34.5
13	NashvilleTennessee33.6
14	NashvilleTennessee33.1
15	NashvilleTennessee33.2
16	NashvilleTennessee32.8
17	NashvilleTennessee32.4
18	NashvilleTennessee32.2
19	NashvilleTennessee32.4
20	Thisisbaddata

### Unpack SQL Call with No Delimiter Separates Virtual Columns

```
SELECT * FROM Unpack (
  ON ville_tempdata1 AS InputTable
  USING
    TargetColumn ('packed_temp_data')
    OutputColumns ('city', 'state', 'temp_f')
    OutputDataTypes ('varchar', 'varchar', 'real')
    ColumnLength ('9', '9', '4')
```

```

Regex ('(.*)')
RegexSet (1)
IgnoreInvalid ('true')
) AS dt ORDER BY sn;

```

## Unpack Output with No Delimiter Separates Virtual Columns

city	state	temp_f	sn
Nashville	Tennessee	3.51000000000000E 001	10
Nashville	Tennessee	3.62000000000000E 001	11
Nashville	Tennessee	3.45000000000000E 001	12
Nashville	Tennessee	3.36000000000000E 001	13
Nashville	Tennessee	3.31000000000000E 001	14
Nashville	Tennessee	3.32000000000000E 001	15
Nashville	Tennessee	3.28000000000000E 001	16
Nashville	Tennessee	3.24000000000000E 001	17
Nashville	Tennessee	3.22000000000000E 001	18
Nashville	Tennessee	3.24000000000000E 001	19

## Example: More Input Columns than Output Columns

### Unpack Input

The input table is ville\_tempdata1 with no delimiter that separates virtual columns in the packed data. Its packed\_temp\_data column has three virtual columns.

#### ville\_tempdata1

sn	packed_temp_data
10	NashvilleTennessee35.1
11	NashvilleTennessee36.2
12	NashvilleTennessee34.5
13	NashvilleTennessee33.6
14	NashvilleTennessee33.1
15	NashvilleTennessee33.2
16	NashvilleTennessee32.8

sn	packed_temp_data
17	NashvilleTennessee32.4
18	NashvilleTennessee32.2
19	NashvilleTennessee32.4
20	Thisisbaddata

## Unpack SQL Call with More Input Columns than Output Columns

The OutputColumns syntax element specifies only two output column names.

```
SELECT * FROM Unpack (
    ON ville_tempdata1 AS InputTable
    USING
        TargetColumn ('packed_temp_data')
        OutputColumns ('city', 'state')
        OutputDataTypes ('varchar', 'varchar')
        ColumnLength ('9', '9')
        Regex ('(.*)')
        RegexSet (1)
        IgnoreInvalid ('true')
) AS dt ORDER BY sn;
```

## Unpack Output with More Input Columns than Output Columns

The output table has columns for the first two virtual input columns, but not for the third.

city	state	sn
Nashville	Tennessee	10
Nashville	Tennessee	11
Nashville	Tennessee	12
Nashville	Tennessee	13
Nashville	Tennessee	14
Nashville	Tennessee	15
Nashville	Tennessee	16
Nashville	Tennessee	17
Nashville	Tennessee	18
Nashville	Tennessee	19

## Example: Input Columns with Trailing Special Characters

In this example, the items in the first virtual input column have trailing special characters. No delimiter separates the virtual columns. ColumnLength is 2. The call to Unpack includes IgnoreInvalid ('true'), but the output is unexpected.

### Unpack Input

t2	c1
	1,1919-04-05
	1.1919-04-05
	5,.1919-04-05
	2,2019/04/05
	4.,1919-04-05
	32019/04/05

### Unpack SQL Call with IgnoreInvalid ('true') with Trailing Special Characters

```
SELECT * FROM Unpack (
    ON t2 AS InputTable
    USING
        TargetColumn ('c1')
        OutputColumns ('a','b')
        OutputDataTypes ('int','date')
        ColumnLength ('2','*')
        IgnoreInvalid ('True')
) AS dt;
```

### Unpack Output with IgnoreInvalid ('true') with Trailing Special Characters

a	b
1	19/04/05
1	19/04/05
2	19/04/05

The reason for the unexpected output is the behavior of an internal library that Unpack uses, which is as follows:

Input Row	Behavior
1,1919-04-05	Library prunes trailing comma, converts "1" to integer and "1919-04-05" to date. Output row 1.
1.1919-04-05	Library prunes trailing period, converts "1" to integer and "1919-04-05" to date. Output row 2.
5,.1919-04-05	Library prunes trailing comma, converts 5 to integer, but cannot convert ".1919-04-05" to date. (No output row.) With ColumnLength ('3','*'), library prunes trailing comma and period, converts "5" to integer and "1919-04-05" to date, and outputs a row for this input row.
2,2019/04/05	Library prunes trailing comma, converts "2" to integer and "1919/04/05" to date. Output row 3.
4., 1919-04-05	Library converts "4." to integer, but cannot convert ",1919-04-05" to date. No output row.
32019/04/05	Library converts "32" to integer, but cannot convert "019/04/05" to date. No output row.

# Data Exploration Functions

Data exploration functions help you learn about the variables (columns) of the input data set.

- [MovingAverage](#)
- [TD\\_CategoricalSummary](#)
- [TD\\_ColumnSummary](#)
- [TD\\_GetRowsWithMissingValues](#)
- [TD\\_Histogram](#)
- [TD\\_QQNorm](#)
- [TD\\_UnivariateStatistics](#)
- [TD\\_WhichMax](#)
- [TD\\_WhichMin](#)

## MovingAverage

The MovingAverage function computes average values in a series, using the specified moving average type.

### [Weighted Moving Average](#)

Computes average of points in series, applying weights to older values. Weights for older values decrease arithmetically.

### [Triangular Moving Average](#)

Computes double-smoothed average of points in series.

### [Simple Moving Average](#)

Computes unweighted mean of previous  $n$  data points.

### [Modified Moving Average](#)

Computes first value as simple moving average. Computes subsequent values by adding new value and subtracting last average from resulting sum.

### [Exponential Moving Average](#)

Computes average of points in series, applying damping factor that exponentially decreases weights of older values.

### [Cumulative Moving Average](#)

Computes cumulative moving average of value from beginning of series.

Moving Average (MA) is a widely used technical analysis indicator that is used to smooth out fluctuations in a data series and to identify trends. It is calculated by taking the average of a predetermined number of periods or time intervals.

Moving averages can be used to identify trends, support and resistance levels, and potential buy or sell signals. Traders often use multiple moving averages with different time periods to get a clearer picture of the market trends.

Other common uses include:

- Weather forecasting: Meteorologists use moving averages to smooth out variations in temperature, humidity, or wind speed over time. This helps them to identify trends and patterns that may indicate changes in weather patterns.
- Traffic analysis: Transportation engineers use moving averages to analyze traffic flow and congestion patterns. By calculating the average speed or volume of traffic over a specific time period, they can identify peak travel times and plan for infrastructure improvements.
- Manufacturing: Production managers use moving averages to monitor quality control and production efficiency. By tracking production metrics over time, they can identify areas of improvement and optimize their manufacturing processes.
- Health care: Medical researchers use moving averages to analyze trends in disease outbreaks, patient outcomes, and other health-related data. By tracking changes in health indicators over time, they can identify risk factors and develop prevention strategies.

Overall, moving averages can be a useful tool for analyzing any time series data where fluctuations and trends need to be identified and analyzed over time.

## Function Information

- [MovingAverage Syntax](#)
- [Required Syntax Elements for MovingAverage](#)
- [Optional Syntax Elements for MovingAverage](#)
- [MovingAverage Input](#)
- [MovingAverage Output](#)
- [Examples: How to Use MovingAverage](#)

## Weighted Moving Average

Weighted moving average is a type of moving average that assigns different weights to each data point in the time series. This means that the most recent data points have a greater impact on the moving average calculation than older data points. Weighted moving averages are commonly used in finance and economics to smooth out fluctuations in a time series data and identify trends over time.

In an n-point weighted moving average, the most recent data point has weight n, the second most recent data point has weight (n - 1), and so on, until the weight is zero.

With MAvgType ('W'), the MovingAverage function uses this formula:

$$\text{WMA}_M = (nV_M + (n-1)V_{M-1} + \dots + 2V_{(M-n+2)} + V_{(M-n+1)}) / (n + (n-1) + \dots + 2 + 1)$$

$V_M$  is the target column value at index M in the window under consideration.

The value  $n$ , the number of old values to use when calculating the new weighted moving average, is specified by the WindowSize syntax element.

## Triangular Moving Average

The triangular moving average (TMA) differs from the simple moving average in that it is double-smoothed; that is, averaged twice. It takes an average over Simple Moving Averages. Double-smoothing keeps the triangular moving average from responding to new data points as fast as the simple moving average. If you want an average that responds quickly to new data points, use the simple moving average or exponential moving average.

With MAvgType ('T'), the MovingAverage function uses this procedure:

1. Compute the window size, N:

$$N = \text{ceil}(window\_size + 1)/2$$

2. Compute the simple moving average of each target column, using this formula:

$$\text{SMA}_i = (V_1 + V_2 + \dots + V_N)/N$$

The function calculates  $\text{SMA}_i$  on the  $i$ th window of the target column from the start of the row.

$V_i$  is the value of the target column at index  $i$  in the window.

3. Compute the triangular moving average by computing the simple moving average with window size N on the values obtained in step 2, using this formula:

$$\text{TMA} = (\text{SMA}_1 + \text{SMA}_2 + \dots + \text{SMA}_N)/N$$

The function writes the cumulative moving average values computed for the first  $n$  rows, where  $n$  is less than N, to the output table.

## Simple Moving Average

The simple moving average (SMA) is the most basic type of moving average. It is calculated by taking the sum of a set of data points and dividing by the number of data points in the set. The formula for an SMA is:

$$\text{SMA} = (\text{Sum of data over a specified number of periods}) / (\text{Number of periods})$$

For example, if you wanted to calculate a 10-day moving average of a stock's price, you would add up the closing prices of the stock over the past 10 days and divide by 10.

## Modified Moving Average

Modified moving average (MMA) makes averages more susceptible to recent shifts. The Modified moving average is a special case of the Exponential moving average, for which the smoothing constant is equal to the reciprocal of the smoothing interval.

The first point of the modified moving average is calculated like the first point of the simple moving average. Each subsequent point is calculated by adding the new value to the most recently calculated modified moving average value and then, from that sum, subtracting the last average value. The difference is the new modified moving average value.

With MAvgType ('M'), the MovingAverage function uses this procedure:

- Compute the arithmetic average of the first *window\_size* rows.

For each subsequent row, compute the new modified moving average value with this formula:

$$\text{MMA}_M = \text{MMA}_{M-1} + (1/\text{window\_size}) (\text{V}_M - \text{MMA}_{M-1})$$

$\text{V}_M$  is the current value. M is the current index.

## Exponential Moving Average

Exponential moving average (EMA), or exponentially weighted moving average (EWMA), applies a damping factor, *alpha*, that exponentially decreases the weights of older values. This technique gives much more weight to recent observations, while retaining older observations.

With MAvgType ('E'), the MovingAverage function uses this procedure:

1. Compute the arithmetic average of the first *n* rows.
2. For each subsequent row, compute the new exponential moving average value with this formula:

$$\text{EMA}_M = \text{alpha} * \text{V} + (1 - \text{alpha}) * \text{EMA}_{M-1}$$

## Cumulative Moving Average

The cumulative moving average, also known as the running average, is calculated by adding all of the data points up to a certain point in time and dividing by the number of data points. The formula for calculating a cumulative moving average is:

$$\text{CMA} = (\text{Sum of data points up to a specific point in time}) / (\text{Number of data points up to that point in time})$$

For example, the cumulative moving average of a stock's closing price on day 20 would be calculated by adding the closing prices of days 1 through 20 and dividing by 20.

With MAvgType ('C'), the MovingAverage function computes the arithmetic average of all the rows from the beginning of the series with this formula:

$$\text{CMA} = (\text{V}_1 + \text{V}_2 + \dots + \text{V}_N)/N$$

$\text{V}_i$  is a value. N is the number of rows from the beginning of the data set.

## MovingAverage Syntax

```
MovingAverage (
    ON { table | view | (query) }
    [ PARTITION BY partition_column [, ...] ]
```

```

[ ORDER BY order_column [,...] ]
[ USING
  [ MAvgType ({'C' | 'E' | 'M' | 'S' | 'T' | 'W'}) ]
  [ TargetColumns ({'target_column' | 'target_column_range'}[,...]) ]
  [ Alpha (alpha) ]
  [ StartRows (n) ]
  [ IncludeFirst ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
  [ WindowSize (window_size) ]
]
)

```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for MovingAverage

### ON clause

Accepts the PARTITION BY and ORDER BY clauses.

**Note:**

- If the ON clause does not include the PARTITION BY and ORDER BY clauses, results are nondeterministic.
- The ORDER BY clause supports only ASCII collation.
- The PARTITION BY clause assumes column names are in Normalization Form C (NFC).

## Optional Syntax Elements for MovingAverage

### MAvgType

Specify one of the following moving average types:

Type	Description
'C' (Default)	Cumulative moving average.
'E'	Exponential moving average.

Type	Description
'M'	Modified moving average.
'S'	Simple moving average.
'T'	Triangular moving average.
'W'	Weighted moving average.

**TargetColumns**

Specify the input column names for which to compute the moving average.

Default behavior: The function copies every input column to the output table but does not compute any moving averages.

**Alpha**

[Optional with MAvgType E, otherwise ignored.] Specify the damping factor, a value in the range [0, 1], which represents a percentage in the range [0, 100]. For example, if *alpha* is 0.2, the damping factor is 20%. A higher *alpha* discounts older observations faster.

Default: 0.1

**StartRows**

[Optional with MAvgType E, otherwise ignored.] Specify the number of rows to skip before calculating the exponential moving average. The function uses the arithmetic average of these rows as the initial value of the exponential moving average. The value must be an integer.

Default: 2

**IncludeFirst**

[Ignored with MAvgType C, otherwise optional.] Specify whether to include the starting rows in the output table. If you specify 'true', the output columns for the starting rows contain NULL, because their moving average is undefined.

Default: 'false'

**WindowSize**

[Optional with MAvgType M, S, T, and W; otherwise ignored.] Specify the number of previous values to consider when computing the new moving average. The data type of *window\_size* must be BYTEINT, SMALLINT, or INTEGER.

Minimum value: 3

Default: '10'

## MovingAverage Input

### Input Table Schema

Column	Data Type	Description
<i>partition_column</i>	Any	Column by which input data is partitioned. This column must contain all rows of an entity. For example, if function is to compute moving average of a particular stock share price, all transactions of that stock must be in one partition. PARTITION BY clause assumes column names are in Normalization Form C (NFC).
<i>order_column</i>	Any	Column by which input table is ordered. ORDER BY clause supports only ASCII collation.
<i>target_column</i>	INTEGER, SMALLINT, BIGINT, NUMERIC, NUMBER, or DOUBLE PRECISION	Values to average.

## MovingAverage Output

### Output Table Schema

Column	Data Type	Description
<i>partition_column</i>	Same as in input table	Column by which input data is partitioned.
<i>order_column</i>	Same as in input table	Column by which input table is ordered.
<i>target_column</i>	Same as in input table	Column copied from input table.
<i>target_column_typeavg</i>	DOUBLE PRECISION	Moving average of <i>target_column</i> values when row was added to data set, where <i>type</i> is value of MAvgType syntax element.

## Examples: How to Use MovingAverage

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [MovingAverage Examples Input](#)
- [Example: Simple Moving Average](#)

- [Example: Weighted Moving Average](#)
- [Example: Exponential Moving Average](#)
- [Example: Modified Moving Average](#)
- [Example: Cumulative Moving Average](#)
- [Example: Triangular Moving Average](#)

## MovingAverage Examples Input

The input table, company1\_stock, contains observations of common stock closing prices.

**company1\_stock**

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>
1	Company1	1961-05-17 00:00:00.000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000
4	Company1	1961-05-22 00:00:00.000000	459.000000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000
8	Company1	1961-05-26 00:00:00.000000	479.000000000000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000
10	Company1	1961-05-31 00:00:00.000000	490.000000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>
22	Company1	1961-06-16 00:00:00.000000	482.000000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000
24	Company1	1961-06-20 00:00:00.000000	478.000000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000

## Example: Simple Moving Average

This example computes a simple moving average for the price of stock.

### MovingAverage SQL Call for Simple Moving Average

```
SELECT * FROM MovingAverage (
    ON company1_stock PARTITION BY name ORDER BY period
    USING
        MAvgType ('S')
        TargetColumns ('stockprice')
        WindowSize (10)
        IncludeFirst ('true')
) AS dt ORDER BY id;
```

### MovingAverage Output for Simple Moving Average

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_smavg</b>
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	?
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	?
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	?
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	?
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	?
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	?
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	?
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	?
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	?
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	470.600000000000

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_smavg</b>
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	474.700000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	479.400000000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	483.200000000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	486.600000000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	489.700000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	492.300000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	492.200000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	491.600000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	491.700000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	491.200000000000
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	489.600000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	487.600000000000
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	485.700000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	484.000000000000

## Example: Weighted Moving Average

This example computes the weighted moving average for the price of stock.

### MovingAverage SQL Call for Weighted Moving Average

```
SELECT * FROM MovingAverage (
    ON company1_stock PARTITION BY name ORDER BY period
    USING
        MAvgType ('W')
        TargetColumns ('stockprice')
        WindowSize (10)
        IncludeFirst ('true')
) AS dt ORDER BY id;
```

### MovingAverage Output for Weighted Moving Average

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_wmavg</b>
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	?

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_wmavg</b>
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	?
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	?
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	?
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	?
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	?
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	?
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	?
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	?
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	473.454545454545
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	477.927272727273
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	482.909090909091
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	487.327272727273
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	490.527272727273
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	492.854545454545
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	493.472727272727
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	493.345454545455
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	490.745454545455
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	489.800000000000
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	489.690909090909
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	488.836363636364
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	487.163636363636
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	485.236363636364
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	483.490909090909
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	482.272727272727

## Example: Exponential Moving Average

This example computes an exponential moving average for the price of stock.

## MovingAverage SQL Call for Exponential Moving Average

```
SELECT * FROM MovingAverage (
    ON company1_stock PARTITION BY name ORDER BY period
    USING
        MAvgType ('E')
        TargetColumns ('stockprice')
        StartRows (10)
        Alpha (0.1)
        IncludeFirst ('true')
) AS dt ORDER BY id;
```

## MovingAverage Output for Exponential Moving Average

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_emavg</b>
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	?
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	?
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	?
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	?
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	?
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	?
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	?
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	?
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	?
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.860000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.674000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	475.306600000000
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	477.475940000000
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	479.328346000000
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	480.395511400000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	481.255960260000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	480.930364234000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	481.537327810600

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_emavg</b>
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	482.483595029540
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	482.935235526586
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	482.841711973927
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	482.457540776535
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	482.011786698881
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	481.710608028993

## Example: Modified Moving Average

This example computes the modified moving average for the price of stock.

### MovingAverage SQL Call for Modified Moving Average

```
SELECT * FROM MovingAverage (
    ON company1_stock PARTITION BY name ORDER BY period
    USING
        MAvgType ('M')
        TargetColumns ('stockprice')
        WindowSize (10)
        IncludeFirst ('true')
) AS dt ORDER BY id;
```

### MovingAverage Output for Modified Moving Average

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_mmavg</b>
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	459.700000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	458.930000000000
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	458.937000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	459.243300000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	459.218970000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	459.597073000000
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	461.537365700000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	464.683629130000

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_mmavg</b>
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.215266217000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.693739595300
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.524365635770
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	475.171929072193
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	477.354736164974
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	479.219262548476
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	480.297336293629
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	481.167602664266
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	480.850842397839
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	481.465758158055
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	482.419182342250
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	482.877264108025
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	482.789537697222
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	482.410583927500
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	481.969525534750
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	481.672572981275

## Example: Cumulative Moving Average

This example computes a cumulative moving average for the price of stock.

### MovingAverage SQL Call for Cumulative Moving Average

```
SELECT * FROM MovingAverage (
    ON company1_stock PARTITION BY name ORDER BY period
    USING
        MAVgType ('C')
        TargetColumns ('stockprice')
) AS dt ORDER BY id;
```

### MovingAverage Output for Cumulative Moving Average

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_cmavg</b>
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_cmavg</b>
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	458.500000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	456.333333333333
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	457.000000000000
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	458.000000000000
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	458.166666666667
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	458.857142857143
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	461.375000000000
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	464.888888888889
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	467.400000000000
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	469.636363636364
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	472.000000000000
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	474.076923076923
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	475.714285714286
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	477.066666666667
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	477.875000000000
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	478.529411764706
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	478.500000000000
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	478.947368421053
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	479.550000000000
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	479.904761904762
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	480.000000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	479.956521739130
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	479.875000000000
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	479.840000000000

## Example: Triangular Moving Average

This example computes the triangular moving average for the price of stock.

## MovingAverage SQL Call for Triangular Moving Average

```
SELECT * FROM MovingAverage (
    ON company1_stock PARTITION BY name ORDER BY period
    USING
        MAvgType ('T')
        TargetColumns ('stockprice')
        WindowSize (10)
        IncludeFirst ('true')
) AS dt ORDER BY id;
```

## MovingAverage Output for Triangular Moving Average

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_tmavg</b>
1	Company1	1961-05-17 00:00:00.000000	460.000000000000	460.000000000000
2	Company1	1961-05-18 00:00:00.000000	457.000000000000	459.250000000000
3	Company1	1961-05-19 00:00:00.000000	452.000000000000	458.277777777778
4	Company1	1961-05-22 00:00:00.000000	459.000000000000	457.958333333333
5	Company1	1961-05-23 00:00:00.000000	462.000000000000	457.966666666667
6	Company1	1961-05-24 00:00:00.000000	459.000000000000	458.000000000000
7	Company1	1961-05-25 00:00:00.000000	463.000000000000	457.777777777778
8	Company1	1961-05-26 00:00:00.000000	479.000000000000	458.416666666667
9	Company1	1961-05-29 00:00:00.000000	493.000000000000	460.555555555556
10	Company1	1961-05-31 00:00:00.000000	490.000000000000	463.444444444444
11	Company1	1961-06-01 00:00:00.000000	492.000000000000	467.000000000000
12	Company1	1961-06-02 00:00:00.000000	498.000000000000	471.611111111111
13	Company1	1961-06-05 00:00:00.000000	499.000000000000	477.138888888889
14	Company1	1961-06-06 00:00:00.000000	497.000000000000	482.555555555556
15	Company1	1961-06-07 00:00:00.000000	496.000000000000	486.916666666667
16	Company1	1961-06-08 00:00:00.000000	490.000000000000	490.416666666667
17	Company1	1961-06-09 00:00:00.000000	489.000000000000	493.000000000000
18	Company1	1961-06-12 00:00:00.000000	478.000000000000	493.944444444444
19	Company1	1961-06-13 00:00:00.000000	487.000000000000	493.555555555556
20	Company1	1961-06-14 00:00:00.000000	491.000000000000	492.500000000000

<b>id</b>	<b>name</b>	<b>period</b>	<b>stockprice</b>	<b>stockprice_tmavg</b>
21	Company1	1961-06-15 00:00:00.000000	487.000000000000	491.111111111111
22	Company1	1961-06-16 00:00:00.000000	482.000000000000	489.500000000000
23	Company1	1961-06-19 00:00:00.000000	479.000000000000	487.694444444444
24	Company1	1961-06-20 00:00:00.000000	478.000000000000	486.444444444444
25	Company1	1961-06-21 00:00:00.000000	479.000000000000	485.305555555556

## TD\_CategoricalSummary

TD\_CategoricalSummary displays the distinct values and their counts for each specified input table column.

A categorical summary refers to a summary of data that is organized into distinct categories or groups. This type of summary is commonly used when dealing with data that is nominal or categorical in nature, such as demographic information, survey responses, or the type of products purchased by customers.

TD\_CategoricalSummary can provide information about the number of observations in each category. This information can be presented in tables, charts, histograms, or pie charts, depending on the type of data and the research question.

Categorical summaries can be useful in several stages of the machine learning process, such as data exploration, feature engineering, and model evaluation. They can help identify patterns, relationships, and outliers in the data, as well as guide the selection and transformation of features for the predictive model. They can also be used to assess the performance of the model and compare it with other models or benchmarks.

### Function Information

- [TD\\_CategoricalSummary Syntax](#)
- [Required Syntax Elements for TD\\_CategoricalSummary](#)
- [Optional Syntax Elements for TD\\_CategoricalSummary](#)
- [TD\\_CategoricalSummary Input](#)
- [TD\\_CategoricalSummary Output](#)
- [Example: How to Use TD\\_CategoricalSummary](#)

## TD\_CategoricalSummary Syntax

```
TD_CategoricalSummary (
    ON { table | view | (query) } AS InputTable
    USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_CategoricalSummary

### ON clause

Accepts the InputTable clause.

### TargetColumns

Specify the names of the InputTable columns for which to display the distinct values and their counts.

## Optional Syntax Elements for TD\_CategoricalSummary

All syntax elements are required.

## TD\_CategoricalSummary Input

### InputTable Schema

Column	Data Type	Description
target_column	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE)	Column to display distinct values and their counts.

## TD\_CategoricalSummary Output

### Output Table Schema

Column	Data Type	Description
ColumnName	VARCHAR (CHARACTER SET UNICODE)	Name of target_column.
DistinctValue	VARCHAR (CHARACTER SET LATIN or UNICODE depending on input table)	Name of distinct value in target_column. Table has one row for each distinct value.
DistinctValueCount	BIGINT	Count of distinct value in target_column. Table has one row for each distinct value.

## Example: How to Use TD\_CategoricalSummary

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_CategoricalSummary InputTable: cat\_titanic\_train

	passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin
	embarked										
A5	97	0	1	Goldschmidt; Mr. George B	male	71	0	0	PC 17754	34.6542	
B37	488	C	0	Kent; Mr. Edward Austin	male	58	0	0	11771	29.7	
B79	505	C	1	Maioni; Miss. Roberta	female	16	0	0	110152	86.5	
A23	631	S	1	Barkworth; Mr. Algernon Henry Wilson	male	80	0	0	27042	30	
B55	873	S	0	Carlsson; Mr. Frans Olof	male	33	0	0	695	5	B51 B53

### Example: TD\_CategoricalSummary Call for Single Column

```
SELECT * FROM TD_CategoricalSummary (
    ON cat_titanic_train AS InputTable
    USING
        TargetColumns ('gender')
) AS dt;
```

### TD\_CategoricalSummary Output

ColumnName	DistinctValue	DistinctValueCount
gender	female	1
gender	male	4

### Example: Calculating Statistics Using Multiple Columns

You can input multiple columns in the TargetColumns parameter to have these statistics calculated for all columns at the same time. embarked has been added.

```
SELECT * FROM TD_CategoricalSummary (
    ON cat_titanic_train AS InputTable
    USING
        TargetColumns ('embarked', 'gender')
) AS dt;
```

## TD\_Categorical Columns Output

ColumnName	DistinctValue	DistinctValueCount
embarked	S	3
gender	female	1
embarked	C	2
gender	male	4

## Example: Calculating Statistics Using Indexing

Another way to run the query is to use indexing. Instead of writing column names, you can also use indexes. Indexes start from 0 (first column is indexed by 0) and then 1, and so on.

**Input data for this (passengers\_2):**

name	gender	ticket	cabin	embarked
Harris	male	PC 17754	A3	C
Jim	male	PC 1754	A5	B
Taha	male	PC 1984	A8	C
Ahmed	male	PC 17754	A5	C
John	male	PC 1772	A8	A

## Indexing Input Query

The reason to use this data is because TD\_CategoricalSummary only works on categorical columns and this dataset only has categorical columns. The previous dataset had some integer and decimal datatype columns.

Running the following query gives the stats for the zeroth, first, and second columns:

```
SELECT * FROM TD_CategoricalSummary (
    ON passengers_2 AS InputTable
    USING
    TargetColumns ('[0:2]')
) AS dt;
```

## Indexing Output

ColumnName	DistinctValue	DistinctValueCount
name	Ahmed	1
name	Jim	1
name	Harris	1

name	John	1
ticket	PC 1772	1
ticket	PC 17754	2
gender	male	5
name	Taha	1
ticket	PC 1984	1
ticket	PC 1754	1

## Statistics for All Columns Query

```
SELECT * FROM TD_CategoricalSummary (
    ON passengers_2 AS InputTable
    USING
    TargetColumns ('[:]')
) AS dt;
```

## All Columns Output

ColumnName	DistinctValue	DistinctValueCount
cabin	A3	1
name	Jim	1
cabin	A8	2
cabin	A5	2
name	Ahmed	1
ticket	PC 17754	2
embarked	B	1
embarked	A	1
ticket	PC 1772	1
name	Harris	1
embarked	C	3
ticket	PC 1984	1
gender	male	5
name	John	1
name	Taha	1
ticket	PC 1754	1

Using indexes in the TargetColumns parameter requires entering square brackets in quotes, and the entered column indexes need to be of the CHAR or VARCHAR datatype. If they are not, the query gives an error. You select all columns by entering the following [:] in quotes on the TargetColumns parameter.

## TD\_ColumnSummary

TD\_ColumnSummary is a function where the contents of a column are grouped and presented with respect to certain standard values. Summarizing a column in terms of positive, negative, null, and other similar values can be useful in several ways:

- Data Cleaning: Summarizing the column in terms of null and not null values can identify missing data, which is important for data cleaning and analysis. Depending on the nature of the analysis, missing data can either be removed or imputed.
- Data Exploration: Summarizing the column in terms of positive and negative values can identify trends or patterns in the data. For example, if the column represents the profit of a business, a large number of negative values may indicate that the business is not performing well.
- Data Validation: Summarizing the column in terms of positive and negative values can help identify errors or anomalies in the data. For example, if the column represents the height of people, a negative value is impossible and would indicate an error in the data.
- Reporting: Summarizing the column in terms of positive, negative, null, and not null values can provide a quick summary of the data that can be included in reports or presentations.

In general, summarizing a column in terms of positive, negative, null, not null, zeros, blanks and their respective percentages can help provide a quick overview of the data and identify potential issues or trends.

A column can be summarized with respect to the values it contains. To summarize a given column as blank values, zeros, null, not null, positive, negative and their respective percentages, follow the steps:

1. Count the total number of values (n) in the column.
2. Count the number of blank values (b) in the column.
3. Count the number of zero values (z) in the column.
4. Count the number of null values (null) in the column.
5. Count the number of not null values (n - null) in the column.
6. Count the number of positive values (pos) in the column.
7. Count the number of negative values (neg) in the column.
8. Calculate the percentage of blank values ( $b/n * 100$ ) in the column.
9. Calculate the percentage of zero values ( $z/n * 100$ ) in the column.
10. Calculate the percentage of null values ( $null/n * 100$ ) in the column.
11. Calculate the percentage of positive values ( $pos/n * 100$ ) in the column.
12. Calculate the percentage of negative values ( $neg/n * 100$ ) in the column.

Such a summary is used for exploratory data analysis of each column in a given database and its respective table. A correlation of the columns in terms of their summary can direct us towards their dependency on each other.

TD\_ColumnSummary displays the following for each specified input table column:

- Column name
- Column data type
- Count of these values:

- Non-NULL
- NULL
- Blank (all space characters) (NULL for numeric data type)
- Zero (NULL for nonnumeric data type)
- Positive (NULL for nonnumeric data type)
- Negative (NULL for nonnumeric data type)
- Percentage of NULL values
- Percentage of non-NULL values

## Function Information

- [TD\\_ColumnSummary Syntax](#)
- [Required Syntax Elements for TD\\_ColumnSummary](#)
- [Optional Syntax Elements for TD\\_ColumnSummary](#)
- [TD\\_ColumnSummary Input](#)
- [TD\\_ColumnSummary Output](#)
- [Example: How to Use TD\\_ColumnSummary](#)

## TD\_ColumnSummary Syntax

```
TD_ColumnSummary (
    ON { table | view | (query) } AS InputTable
    USING
    TargetColumns ({ 'target_column' | target_column_range }[,...])
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

---

## Required Syntax Elements for TD\_ColumnSummary

### ON clause

Accept the InputTable clause.

### TargetColumns

Specify the names of the InputTable columns to summarize.

## Optional Syntax Elements for TD\_ColumnSummary

All syntax elements are required.

### TD\_ColumnSummary Input

#### InputTable Schema

Column	Data Type	Description
target_column	Any	Column to display summary.

### TD\_ColumnSummary Output

#### Output Table Schema

Column	Data Type	Description
ColumnName	VARCHAR (CHARACTER SET UNICODE)	Name of target column.
DataType	VARCHAR (CHARACTER SET LATIN)	Data type of target column.
NonNullCount	BIGINT	Count of non-NULL values in target column.
NullCount	BIGINT	Count of NULL values in target column.
BlankCount	BIGINT	Count of blank values (values with all space characters) in target column when data type is CHAR or VARCHAR; otherwise NULL.
ZeroCount	BIGINT	Count of zero values in target column when data type is NUMERIC; otherwise NULL.
PositiveCount	BIGINT	Count of positive values in target column when data type is NUMERIC; otherwise NULL.
NegativeCount	BIGINT	Count of negative values in target column when data type is NUMERIC; otherwise NULL.
NullPercentage	DOUBLE PRECISION	Percentage of NULL values in target column.
NonNullPercentage	DOUBLE PRECISION	Percentage of non-NULL values in target column.

## Example: How to Use TD\_ColumnSummary

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_ColumnSummary InputTable: col\_titanic\_train

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin
embarked										
C	49	0	3 Samaan; Mr. Youssef	male	null	2	0	2662	21.679	null
S	78	0	3 Moutal; Mr. Rahamin Haim	male	null	0	0	374746	8.05	null
S	505	1	1 Maioni; Miss. Roberta	female	16	0	0	110152	8.65	B79
S	631	1	1 Barkworth; Mr. Algernon Henry Wilson	male	80	0	0	27042	30	A23
B55 S	873	0	1 Carlsson; Mr. Frans Olof	male	33	0	0	695	5	B51 B53

### Example: Using Column Names in TD\_ColumnSummary Call

```
SELECT * FROM TD_ColumnSummary (
    ON col_titanic_train AS InputTable
    USING
        TargetColumns ('age','pclass','embarked','cabin')
) AS dt;
```

### Example: Using Column Range in TD\_ColumnSummary Call

```
SELECT * FROM TD_ColumnSummary (
    ON col_titanic_train AS InputTable
    USING
        TargetColumns ('[10:11]')
) AS dt;
```

### TD\_ColumnSummary Output

ColumnName	Datatype	NonNullCount	NullCount	BlankCount
ZeroCount	PositiveCount	NegativeCount	NullPercentage	NonNullPercentage
age	INTEGER	3	2	
null	0	0	4.00E+001	6.00E+001
cabin	VARCHAR(20) CHARACTER SET LATIN	3	2	
0	null	null	4.00E+001	6.00E+001
embarked	VARCHAR(20) CHARACTER SET LATIN	5	0	

0	null	null	null	0.00E000	1.00E+002
pclass	INTEGER			5	0
null	0	5	0	0.00E000	1.00E+002

## TD\_GetRowsWithMissingValues

TD\_GetRowsWithMissingValues is a function or method that retrieves all rows or records from a dataset that contain one or more missing or null values.

This function is often used in data cleaning or preprocessing tasks to identify and handle missing values in datasets. Finding rows with missing values is an essential step in data cleaning and preprocessing for analytics. By identifying and handling missing values, data analysts and scientists can ensure that their analyses and models are based on complete and accurate data, which can improve the quality and reliability of their results.

Missing values can lead to biased or inaccurate analysis results if not appropriately addressed. Here are some common use cases for finding rows with missing values in analytics:

- **Imputation:** One approach for handling missing values is to impute them, which means to update them with an estimated value. Before imputing missing values, you must identify which rows contain them to ensure that imputed values are accurate and appropriate.
- **Outlier detection:** Missing values can sometimes be an indication of outliers or extreme values. By identifying and investigating the rows with missing values, you can determine whether there are outliers or other anomalies in the data that need to be addressed.
- **Data quality assessment:** Missing values can also be a sign of poor data quality. By identifying the rows with missing values, you can assess the overall quality of the data and determine whether it is suitable for analysis.
- **Feature engineering:** In machine learning, missing values can be problematic because many algorithms cannot handle them. Therefore, you must identify the rows with missing values and decide whether to impute them or drop them. Additionally, missing values can be indicative of a particular pattern in the data that can be used to engineer new features that improve model performance.

Identifying rows with missing values enables data analysts to make informed decisions about how to handle missing values and ensures that their analysis results are based on complete and accurate data.

### Function Information

- [TD\\_GetRowsWithMissingValues Syntax](#)
- [Required Syntax Elements for TD\\_GetRowsWithMissingValues](#)
- [Optional Syntax Elements for TD\\_GetRowsWithMissingValues](#)
- [TD\\_GetRowsWithMissingValues Input](#)
- [TD\\_GetRowsWithMissingValues Output](#)

- [Example: Using TD\\_GetRowsWithMissingValues to Find NULL Values](#)

**Related Information:**[TD\\_GetRowsWithoutMissingValues](#)

## TD\_GetRowsWithMissingValues Syntax

```
TD_GetRowsWithMissingValues (
    ON { table | view | (query) } AS InputTable
        [ PARTITION BY ANY [ORDER BY order_column ] ]
        [ USING
            TargetColumns ({ 'target_column' | target_column_range }[,...])
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
        ]
    )
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_GetRowsWithMissingValues

**ON clause**

Accept the InputTable clause.

## Optional Syntax Elements for TD\_GetRowsWithMissingValues

**TargetColumns**

Specify the target column names to check for NULL values.

Default: If omitted, the function considers all columns of the InputTable.

**Accumulate**

Specify the input table column names to copy to the output table.

## TD\_GetRowsWithMissingValues Input

### InputTable Schema

Column	Data Type	Description
target_column	Any	Columns for which NULL values are checked.
accumulate_column	Any	The input table column names to copy to the output table.

## TD\_GetRowsWithMissingValues Output

### Output Table Schema

Column	Data Type	Description
target_column	Any	Columns for which NULL values are checked.
accumulate_column	Any	The input table column names to copy to the output table.

## Example: Using TD\_GetRowsWithMissingValues to Find NULL Values

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_GetRowsWithMissingValues InputTable: input\_table

passenger survived pclass name cabin embarked				gender	age	sibsp	parch	ticket	fare
----- -----				-----	-----	-----	-----	-----	-----
null	1	0	3 Braund; Mr. Owen Harris		male	22	1	0 A/5 21171	7.25
null	30	0	3 Todoroff; Mr. Lallio		male	null	0	0 349216	7.8958
B79	505	1	1 Maioni; Miss. Roberta		female	16	0	0 110152	86.5
A23	631	1	1 Barkworth; Mr. Algernon Henry Wilson		male	80	0	0 27042	30
B55	873	0	1 Carlsson; Mr. Frans Olof		male	33	0	0 695	5 B51 B53

### TD\_GetRowsWithMissingValues SQL Call

```
SELECT * FROM TD_getRowsWithMissingValues (
    ON input_table AS InputTable
    USING
        TargetColumns ('[name:cabin]')
) AS dt;
```

## TD\_GetRowsWithMissingValues Output

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25	null	S
30	0	3	Todoroff, Mr. Lalio	male	null	0	0	349216	7.8958	null	S

## TD\_Histogram

TD\_Histogram calculates the frequency distribution of a dataset using one of the following methods:

- Sturges
- Scott
- Variable-width
- Equal-width

A histogram is a graphical representation of the distribution of numerical data. It consists of a series of rectangles, or bars, which represent the frequency of occurrences of data values within certain intervals, or "bins." The x-axis of a histogram represents the range of values in the data set being analyzed, while the y-axis represents the frequency of occurrence of values within each bin. Each bar in the histogram represents a bin, and the height of the bar corresponds to the number of data points in that bin.

Several types of histograms can be used, depending on the characteristics of the data being analyzed:

- Continuous Histograms: Use for data that is continuous, such as height or weight. The bins are typically defined as intervals of equal width, and the area of each bar represents the proportion of data points that fall within that interval.
- Discrete Histograms: Use for data that is discrete, such as the number of cars in a parking lot. The bins are typically defined as integer values, and the height of each bar represents the frequency of occurrence of each integer value.
- Frequency Polygon: Replace the bars by points and the lines are drawn connecting these points. This type of histogram is useful for comparing multiple datasets on the same graph.
- Cumulative Histogram: Show the cumulative frequency of the data as the bars are stacked on top of each other. This type of histogram is useful for identifying percentiles and cumulative proportions.
- 2D Histogram: Represent the joint distribution of two variables. The x-axis and y-axis represent the two variables, and the height of each bar represents the frequency of occurrence of each combination of values.
- Kernel Density Estimation Histogram: Smooth version of a histogram that estimates the probability density function of the data. This type of histogram is useful for identifying the shape of the distribution and identifying any peaks or outliers.

A histogram offers several advantages for analyzing and visualizing data:

- Provides a clear visual representation: Histograms are an effective way to represent large sets of data in a clear and concise manner. They allow you to quickly visualize the distribution of the data and identify patterns or outliers.

- Easy to understand: Histograms are relatively simple to understand and do not require advanced statistical knowledge. They are a useful tool for communicating data to a wide range of audiences, including those without a background in statistics.
- Allows for easy comparisons: Histograms make it easy to compare data sets and identify differences or similarities between them. Multiple histograms can be plotted on the same graph, making it easy to compare the distribution of different variables.
- Provides information on central tendency and variability: The shape of the histogram can indicate whether the data is skewed or symmetrical, while the spread of the data can be estimated from the range of the bins.
- Identifies outliers: Histograms can be used to identify outliers or unusual data points that fall outside the typical range of values. This can be particularly useful for detecting errors or anomalies in a dataset.

## Function Information

- [TD\\_Histogram Syntax](#)
- [Required Syntax Elements for TD\\_Histogram](#)
- [Optional Syntax Elements for TD\\_Histogram](#)
- [TD\\_Histogram Input](#)
- [TD\\_Histogram Output](#)
- [Examples: How to Use TD\\_Histogram](#)

## TD\_Histogram Syntax

```
TD_Histogram (
    ON { table | view | (query) } AS InputTable
    [ ON { table | view | (query) } AS MinMax DIMENSION ]
    USING
        MethodType ({ 'Sturges' | 'Scott' | 'Variable-Width' | 'Equal-Width' })
        TargetColumn ({'target_column' | 'target_column_range'[,...] })
        [ NBins ({'nbins' | 'nbins_i',... } ) ]
        [ Inclusion ({ 'inclusion_val' | 'inclusion_val_i',... } ) ]
        [ GroupbyColumns ({ 'group_column_i',... } ) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_Histogram

### ON clause for InputTable

Accept the InputTable clause.

### MethodType

Specify the method for calculating the frequency distribution of the dataset:

Available Methods	Description
Sturges	<p>Sturges algorithm performs best if data is normally distributed and <math>n</math> is at least 30.</p> <p>Algorithm for calculating bin width:</p> $w = r/(1 + \log_2 n)$ <p>where:</p> <ul style="list-style-type: none"> <li><math>w</math> = bin width</li> <li><math>r</math> = data value range</li> <li><math>n</math> = number of elements in dataset</li> </ul>
Scott	<p>Scott algorithm performs best on normally distributed data.</p> <p>Algorithm for calculating bin width:</p> $w = 3.49s/(n^{1/3})$ <p>where:</p> <ul style="list-style-type: none"> <li><math>w</math> = bin width</li> <li><math>s</math> = standard deviation of data values</li> <li><math>n</math> = number of elements in dataset</li> <li><math>r</math> = data value range</li> </ul> <p>Number of bins: <math>r/w</math></p>
Variable-Width	<ul style="list-style-type: none"> <li>• Requires MinMax table, which specifies the minimum value and the maximum value of the bin.</li> <li>• If one target column is specified, specify the minimum value in column1, maximum value in column2, and label of the bin in column3.</li> <li>• If more than one target column is specified, specify ColumnName in column1, minimum value in column2, maximum value in column3, and the label of the bin in column4.</li> </ul> <p><b>Note:</b></p> <p>The maximum number of bins cannot exceed 10000 per column.</p>
Equal-Width	<p>Algorithm for calculating bin width:</p> $w = (max - min)/k$ <p>where:</p> <ul style="list-style-type: none"> <li><math>min</math> = minimum value of the bins</li> <li><math>max</math> = maximum value of the bins</li> <li><math>k</math> = number of intervals into which algorithm divides dataset</li> </ul> <p>Interval boundaries: <math>min+w, min+2w, \dots, min+(k-1)w</math></p>

Available Methods	Description
	<ul style="list-style-type: none"> <li>• Optional MinMax table.</li> <li>• If MinMax table is omitted, the TD_Histogram function internally computes the min value and max value from the input data for the target columns.</li> <li>• If MinMax table is specified, the user can specify in the following manner: <ul style="list-style-type: none"> <li>◦ If one target column is specified, specify min value in column1 and max value in column2.</li> <li>◦ If more than one target column is specified, specify ColumnName in column1, min value in column2, and max value in column3.</li> </ul> </li> </ul>

**TargetColumn**

Specify the InputTable columns for which the histogram is to be calculated.

**NBins**

[Required with methods Equal-Width and Variable-Width] Specify the integer value that specifies the number of ranges or bins.

If only one value is specified, it is applied to all the target columns. Otherwise, the number of NBins values must be equal to the number of target columns.

**Note:**

The maximum NBins value is 10000.

**Inclusion**

Specify whether to include points on bin boundaries, in the bin to the left of the boundary or the bin to the right of the boundary.

If only one value is specified, it is applied to all the target columns. Otherwise, the number of Inclusion values must be equal to the number of target columns.

Default: left

**GroupByColumns**

Specify the names of the InputTable columns that contain the group values for binning.

This argument must not have columns that are already specified in TargetColumn.

This argument does not support range.

**Note:**

The maximum number of unique columns in the GroupByColumns argument is 2042.

## Optional Syntax Elements for TD\_Histogram

### ON clause for Dimension

Accept the MinMax table clause.

### NBins

[Required with methods Equal-Width and Variable-Width] Specify the integer value which will specify the number of ranges or bins.

If only one value is specified, it is applied to all the target columns. Otherwise, the number of NBins values must be equal to the number of target columns.

---

#### Note:

The maximum NBins value is 10000.

---

### Inclusion

Specify whether to include points on bin boundaries, in the bin to the left of the boundary or the bin to the right of the boundary.

If only one value is specified, it is applied to all the target columns. Otherwise, the number of Inclusion values must be equal to the number of target columns.

Default: left

### GroupByColumns

Specify the names of the InputTable columns that contain the group values for binning.

This argument must not have columns that are already specified in TargetColumn.

This argument does not support range.

---

#### Note:

The maximum number of unique columns in the GroupByColumns argument is 2042.

---

## TD\_Histogram Input

### InputTable Schema

Column	Data Type	Description
target_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Indicates the columns to use with the function.

Column	Data Type	Description
group_column	BYTEINT, SMALLINT, INTEGER, BIGINT, CHAR, VARCHAR Character types: Character set can be LATIN or UNICODE.	<p>Determines how input data is grouped. The function produces a separate histogram for each group.</p> <p><b>Note:</b> The column appears once for each specified group_column.</p>

### MinMaxTable Schema: MethodType - Equal-Width

Column	Data Type	Description
ColumnName	CHAR or VARCHAR Character types: Character set can be LATIN or UNICODE.	Target column names. It is required only when more than one target column is specified.
MinValue	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Overall minimum value of the bins.
MaxValue	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Overall maximum value of the bins.

### MinMaxTable Schema: MethodType - Variable-Width

Column	Data Type	Description
ColumnName	CHAR or VARCHAR Character types: Character set can be LATIN or UNICODE.	Target column names. It is required only when more than one target column is specified.
MinValue	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Minimum value of the bin.
MaxValue	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Maximum value of the bin.
Label	CHAR, VARCHAR, BYTEINT, SMALLINT, INTEGER, or BIGINT Character types: Character set can be LATIN or UNICODE.	Corresponding labels for bins or data value ranges. Maximum label length: 128 UNICODE characters.

## TD\_Histogram Output

### Output Table Schema

Column	Data Type	Description
group_column_1...n	BYTEINT, SMALLINT, INTEGER, BIGINT, CHAR, VARCHAR Character types: Character set can be LATIN or UNICODE.	Determines how InputTable data is grouped. <b>Note:</b> The column appears once for each specified group_column.
ColumnName	VARCHAR (UNICODE)	Target column names.
Label	VARCHAR (UNICODE) or BIGINT	Labels for bins or data value ranges.
MinValue	DOUBLE PRECISION	Minimum values for bins or data value ranges.
MaxValue	DOUBLE PRECISION	Maximum values for bins or data value ranges.
CountOfValues	BIGINT	Counts of values in bins or data value ranges.
Bin_percent	DOUBLE PRECISION	Percentage of InputTable rows in bins or data value ranges.

## Examples: How to Use TD\_Histogram

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

---

### Note:

Multiple column support for TD\_OrdinalEncoding, TD\_OneHotEncoding, and TD\_Histogram is available in release 17.20.03.07 and later. If you are using an older version, the TargetColumn argument accepts only one column.

---

### Input Table cars\_hist

sn	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4	21.0	6	160.0	110.0	3.9	2.62	16.46	S	manual	4	4
2	Mazda RX4 Wag	21.0	6	160.0	110.0	3.9	2.875	17.02	S	manual	4	4
3	Datsun 710	22.8	4	108.0	93.0	3.85	2.32	18.61	V	manual	4	1
4	Hornet 4 Drive	21.4	6	258.0	110.0	3.08	3.215	19.44	V	automatic	3	1
5	Hornet Sportabout	18.7	8	360.0	175.0	3.15	3.44	17.02	S	automatic	3	2
6	Plymouth Valiant	18.1	6	225.0	105.0	2.76	3.46	20.22	V	automatic	3	1

sn	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
7	Duster 360	14.3	8	360.0	245.0	3.21	3.57	15.84	S	automatic	3	4
8	Merc 240D	24.4	4	146.7	62.0	3.69	3.19	20.0	V	automatic	4	2
9	Merc 230	22.8	4	140.8	95.0	3.92	3.15	22.9	V	automatic	4	2
10	Merc 280	19.2	6	167.6	123.0	3.92	3.44	18.3	V	automatic	4	4
11	Merc 280C	17.8	6	167.6	123.0	3.92	3.44	18.9	V	automatic	4	4
12	Merc 450SE	16.4	8	275.8	180.0	3.07	4.07	17.4	S	automatic	3	3
13	Merc 450SL	17.3	8	275.8	180.0	3.07	3.73	17.6	S	automatic	3	3
14	Merc 450SLC	15.2	8	275.8	180.0	3.07	3.78	18.0	S	automatic	3	3
15	Cadillac Fleetwood	10.4	8	472.0	205.0	2.93	5.25	17.98	S	automatic	3	4
16	Lincoln Continental	10.4	8	460.0	215.0	3.0	5.424	17.82	S	automatic	3	4
17	Chrysler Imperial	14.7	8	440.0	230.0	3.23	5.345	17.42	S	automatic	3	4
18	Fiat 128	32.4	4	78.7	66.0	4.08	2.2	19.47	V	manual	4	1
19	Honda Civic	30.4	4	75.7	52.0	4.93	1.615	18.52	V	manual	4	2
20	Toyota Corolla	33.9	4	71.1	65.0	4.22	1.835	19.9	V	manual	4	1
21	Toyota Corona	21.5	4	120.1	97.0	3.7	2.465	20.01	V	automatic	3	1
22	Dodge Challenger	15.5	8	318.0	150.0	2.76	3.52	16.87	S	automatic	3	2
23	AMC Javelin	15.2	8	304.0	150.0	3.15	3.435	17.3	S	automatic	3	2
24	Camaro Z28	13.3	8	350.0	245.0	3.73	3.84	15.41	S	automatic	3	4
25	Pontiac Firebird	19.2	8	400.0	175.0	3.08	3.845	17.05	S	automatic	3	2
26	Fiat X1-9	27.3	4	79.0	66.0	4.08	1.935	18.9	V	manual	4	1
27	Porsche 914-2	26.0	4	120.3	91.0	4.43	2.14	16.7	S	manual	5	2
28	Lotus Europa	30.4	4	95.1	113.0	3.77	1.513	16.9	V	manual	5	2
29	Ford Pantera L	15.8	8	351.0	264.0	4.22	3.17	14.5	S	manual	5	4
30	Ferrari Dino	19.7	6	145.0	175.0	3.62	2.77	15.5	S	manual	5	6
31	Maserati Bora	15.0	8	301.0	335.0	3.54	3.57	14.6	S	manual	5	8
32	Volvo 142E	21.4	4	121.0	109.0	4.11	2.78	18.6	V	manual	4	2

### Example: TD\_Histogram Call Using Sturges Method Type

```
SELECT * FROM TD_Histogram(
ON cars_hist as InputTable
USING
TargetColumn('hp','disp')
MethodType('STURGES')
) as dt ORDER BY 1,2,3,4,5,6;
```

**TD\_Histogram Output for Sturges Method Type**

ColumnName	Label	MinValue	MaxValue	CountOfValues	Bin_Percent
disp	0	0.0	100.0	5	15.625
disp	1	100.0	200.0	11	34.375
disp	2	200.00	300.00	5	15.625
disp	3	300.00	400.00	7	21.875
disp	4	400.00	500.00	4	12.500
hp	0	50.00	100.00	9	28.125
hp	1	100.00	150.00	8	25.000
hp	2	150.00	200.00	8	25.000
hp	3	200.00	250.00	5	15.625
hp	4	250.00	300.00	1	3.125
hp	5	300.00	350.00	1	3.125

**Example: TD\_Histogram Call Using Scott Method Type**

```
SELECT * FROM TD_Histogram(
ON cars_hist as InputTable
USING
TargetColumn('hp','disp')
MethodType('SCOTT')
) as dt ORDER BY 1,2,3,4,5,6;
```

**TD\_Histogram Output for Scott Method Type**

ColumnName	Label	MinValue	MaxValue	CountOfValues	Bin_Percent
disp	0	0.0	100.0	5	15.625
disp	1	100.0	200.0	11	34.375
disp	2	200.00	300.00	5	15.625
disp	3	300.00	400.00	7	21.875
disp	4	400.00	500.00	4	12.500
hp	0	0.00	100.00	9	28.125
hp	1	100.00	200.00	16	50.000
hp	2	200.00	300.00	6	25.000
hp	3	300.00	400.00	1	3.125

**Example: TD\_Histogram Call Using EqualWidth Method Type**

```
SELECT * FROM TD_Histogram(
ON cars_hist as InputTable
USING
TargetColumn('hp','disp')
MethodType('Equal-Width')
```

```
Nbins(3, 4)
) as dt ORDER BY 1,2,3,4,5,6;
```

### **TD\_Histogram Output for Equal-Width Method Type**

ColumnName	Label	MinValue	MaxValue	CountOfValues	Bin_Percent
disp	0	71.100000	171.325000	16	50.000
disp	1	171.325000	271.550000	2	6.250
disp	2	271.550000	371.775000	10	31.250
disp	3	371.775000	472.000000	4	12.500
hp	0	52.000000	146.333333	17	53.125
hp	1	146.333333	240.666667	11	34.375
hp	2	240.666667	335.000000	4	12.500

### **Example: TD\_Histogram Call Using Variable-Width Method Type**

```
SELECT * FROM TD_Histogram(
ON cars_hist as InputTable
ON minmaxTable as MinMax Dimension
USING
TargetColumn('hp','disp')
MethodType('VARIABLE-WIDTH')
Nbins(3, 4)
) as dt ORDER BY 1,2,3,4,5,6;
```

### **TD\_Histogram Output for Variable-Width Method Type**

ColumnName	Label	MinValue	MaxValue	CountOfValues	Bin_Percent
disp	label_0	71.100	171.325	16	50.000
disp	label_1	171.325	271.550	2	6.250
disp	label_2	271.550	371.775	10	31.250
disp	label_3	371.775	472.000	4	12.500
hp	label_0	52.000	146.333	17	53.125
hp	label_1	146.333	240.666	11	34.375
hp	label_2	240.666	335.000	4	12.500

## TD\_QQNorm

The TD\_QQNorm function is a Q-Q (quantile-quantile) norm method that compares the distribution of a data set to a normal distribution. TD\_QQNorm checks whether the values in an input table columns are normally distributed. The function returns the quantiles of the column values and corresponding theoretical quantile values from a normal distribution. If the column values are normally distributed, then the quantiles of column values and normal quantile values appear in a straight line with a slope of 1, when plotted on a graph.

The data is first sorted in ascending order, and then the corresponding quantiles are calculated. Next, the expected quantiles are calculated based on the theoretical distribution being compared to. For a normal distribution, the expected quantiles are calculated based on the mean and standard deviation of the data.

When plotted on a graph, the function output displays the quantiles of the dataset against the expected quantiles of the theoretical distribution, usually on a scatter plot. Deviations from a straight line indicate deviations from normality.

The TYD\_QQNorm function is commonly used in statistics and data analysis to check the assumptions of statistical models that rely on normality, such as linear regression.

Quartile normalization is a data preprocessing technique commonly used in bioinformatics and statistics to normalize gene expression data. The method aims to remove technical variation that can occur between samples due to differences in data acquisition or processing. It is often used to ensure that data from different samples or platforms are comparable, by adjusting for systematic differences in the data that may arise due to different experimental conditions.

In quartile normalization, the data is sorted in ascending order, and then divided into four equal-sized groups, or quartiles. The median value of each quartile is then computed, and these medians are used to adjust the data values so that the medians across all samples are the same. This equalizes the distribution of the data across samples, making it easier to compare gene expression levels between samples.

Quartile normalization is often used as a preprocessing step for other analyses, such as differential gene expression analysis or clustering.

To use quartile normalization to formulate QQ norm data, follow these steps:

1. For each element in your dataset, rank the expression values from smallest to largest. This creates a new dataset where each value is represented by a rank.
2. Calculate the average rank for each value across all samples. This gives you a new dataset where each element is represented by a single value, which is the average rank across all samples.
3. Sort the average ranks from step 2 in ascending order. This creates a new dataset that represents the order of the values from lowest to highest average rank.
4. Calculate the quantiles of the sorted average ranks, using the bins or quantiles. For example, if we want to use 100 quantiles, we would divide the sorted average ranks into 100 equal-sized bins and calculate the quantiles for each bin.
5. For each sample in your dataset, map the original expression values to their corresponding quantiles from step 4. This ensures that the distribution of expression values within each sample is the same as the distribution of average ranks across all samples.

Once you have the normalized data using quantile normalization, you can create a QQ norm table by tabulating the observed quantiles of your dataset against the expected quantiles from a theoretical distribution, such as the normal distribution. If your data is normally distributed, the points on the QQ plot fall along a straight line. If there are deviations from normality, such as skewness or heavy-tailedness, the points on the QQ plot deviate from the straight line.

## Function Information

- [TD\\_QQNorm Syntax](#)
- [Required Syntax Elements for TD\\_QQNorm](#)
- [Optional Syntax Elements for TD\\_QQNorm](#)
- [TD\\_QQNorm Input](#)
- [TD\\_QQNorm Output](#)
- [Example: How to Use TD\\_QQNorm](#)

## TD\_QQNorm Syntax

```
TD_QQNorm (
    ON { table | view | (query) } AS InputTable
        [ PARTITION BY ANY [ ORDER BY order_column ] ]
    USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        RankColumns ({ 'rank_column' | rank_column_range }[,...])
        [ OutputColumns ('output_column' [,...]) ]
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_QQNorm

### ON clause

Accept the InputTable clause.

### TargetColumns

Specify the names of the numeric InputTable columns to check for normal distribution.

**RankColumns**

Specify the names of the InputTable columns that contain the ranks for the target columns.

**Optional Syntax Elements for TD\_QQNorm****OutputColumns**

Specify names for the output table columns that contain the theoretical quantiles of the target columns.

Default: target\_column\_theoretical\_quantiles

**Accumulate**

Specify the names of the InputTable columns to copy to the output table.

**TD\_QQNorm Input****InputTable Schema**

Column	Data Type	Description
target_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Column to check for normal distribution.
rank_column	BYTEINT, SMALLINT, INTEGER, or BIGINT	Ranks for target column.
accumulate_column	Any	Column to copy to output table.

**TD\_QQNorm Output****Output Table Schema**

Column	Data Type	Description
accumulate_column	Any	Column copied from InputTable.
target_column	DOUBLE PRECISION	Column checked for normal distribution.
output_column if specified, otherwise target_column_theoretical_quantiles	DOUBLE PRECISION	Theoretical quantile values for target column.

## Example: How to Use TD\_QQNorm

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_QQNorm Input

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare
cabin		embarked							
34.6542	97	0	1 Goldschmidt; Mr. George B C	male	71	0	0	PC 17754	
A5									
29.7	488	0	1 Kent; Mr. Edward Austin C	male	58	0	0	11771	
B79	505	1	1 Maioni; Miss. Roberta	female	16	0	0	110152	86.5
A23	631	1	1 Barkworth; Mr. Algernon Henry Wilson	male	80	0	0	27042	30
B51	873	S	1 Carlsson; Mr. Frans Olof	male	33	0	0	695	5
B53		B55							

From input table, create RankTable with this statement:

```
CREATE TABLE RankTable AS (
  SELECT age, fare,
  CAST (ROW_NUMBER() OVER (ORDER BY age ASC NULLS LAST) AS BIGINT)
  AS rank_age,
  CAST (ROW_NUMBER() OVER (ORDER BY fare ASC NULLS LAST) AS BIGINT)
  AS rank_fare
  FROM input_table AS dt
) WITH DATA;
```

age	fare	rank_age	rank_fare
16	86.5	1	5
33	5	2	1
58	29.7	3	2
71	34.6542	4	4
80	30	5	3

### TD\_QQNorm SQL Call Using Column Numbers

```
SELECT * FROM TD_QQNorm (
  ON RankTable AS InputTable
  USING
  TargetColumns ('[0:1]')
  RankColumns ('[2:3]')
) AS dt;
```

## TD\_QQNorm SQL Call Using Column Names

```
SELECT * FROM TD_QQNorm (
    ON RankTable AS InputTable
    USING
        TargetColumns ('age', 'fare')
        RankColumns ('rank_age', 'rank_fare')
) AS dt;
```

## TD\_QQNorm SQL Call Using PARTITION BY ANY Clause

```
SELECT * FROM TD_QQNorm (
    ON RankTable AS InputTable
    PARTITION BY ANY
    USING
        TargetColumns ('age', 'fare')
        RankColumns ('rank_age', 'rank_fare')
) AS dt;
```

## TD\_QQNorm Output

age	age_theoretical_quantiles	fare	fare_theoretical_quantiles
16	-1.17986882170049	86.5	1.17986882170049
33	-0.496788749686441	5	-1.17986882170049
58	-0.000000101006675468085	29.7	-0.496788749686441
71	0.496788749686441	34.6542	0.496788749686441
80	1.17986882170049	30	-0.000000101006675468085

## TD\_UnivariateStatistics

TD\_UnivariateStatistics displays descriptive statistics for each specified numeric input table column.

### Note:

There may be different statistics values on different runs due to the precision difference for decimal and number columns.

### Function Information

- [TD\\_UnivariateStatistics Syntax](#)
- [TD\\_UnivariateStatistics Syntax Elements](#)
- [TD\\_UnivariateStatistics Input](#)

- [TD\\_UnivariateStatistics Output](#)
- [Example: Using TD\\_UnivariateStatistics to Determine Mean, Median, and Mode](#)

## TD\_UnivariateStatistics Syntax

```
TD_UnivariateStatistics (
    ON { table | view | (query) }
        AS InputTable [ PARTITION BY ANY ]
    USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        [ PartitionColumns ('partition_column' [,...]) ]
        [ Stats ('statistic' [,...]) ]
        [ Centiles ('percentile' [,...]) ]
        [ TrimPercentile ('trimmed_percentile') ]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## TD\_UnivariateStatistics Syntax Elements

### TargetColumns

Specify the names of the numeric InputTable columns for which to compute statistics.

### PartitionColumns

[Optional] Specify the names of the InputTable columns on which to partition the input. The function copies these columns to the output table.

Default behavior: The function treats all rows as a single partition.

### Stats

[Optional] Specify the statistics to calculate. *statistic* is one of these:

- SUM
- COUNT or CNT
- MAXIMUM or MAX
- MINIMUM or MIN

- MEAN
- UNCORRECTED SUM OF SQUARES or USS
- NULL COUNT or NLC
- POSITIVE VALUES COUNT or PVC
- NEGATIVE VALUES COUNT or NVC
- ZERO VALUES COUNT or ZVC
- TOP5 or TOP
- BOTTOM5 or BTM
- RANGE or RNG
- GEOMETRIC MEAN or GM
- HARMONIC MEAN or HM
- VARIANCE or VAR
- STANDARD DEVIATION or STD
- STANDARD ERROR or SE
- SKEWNESS or SKW
- KURTOSIS or KUR
- COEFFICIENT OF VARIATION or CV
- CORRECTED SUM OF SQUARES or CSS
- MODE
- MEDIAN or MED
- UNIQUE ENTITY COUNT or UEC
- INTERQUARTILE RANGE or IQR
- TRIMMED MEAN or TM
- PERCENTILES or PRC
- ALL

Default: ALL

## Centiles

[Optional] Specify the centile to calculate. *percentile* is an INTEGER in the range [1, 100].

The function ignores Centiles unless Stats specifies PERCENTILES, PRC, or ALL.

Default: 1, 5, 10, 25, 50, 75, 90, 95, 99

## TrimPercentile

[Optional] Specify the trimmed lower percentile, an integer value in the range [1, 50].

The function calculates the mean of the values between the trimmed lower percentile (*trimmed\_percentile*) and trimmed upper percentile (1-*trimmed\_percentile*).

The function ignores TrimPercentile unless Stats specifies TRIMMED MEAN, TM, or ALL.

Default: 20

## TD\_UnivariateStatistics Input

### InputTable Schema

Column	Data Type	Description
target_column	NUMERIC	Column to calculate statistics.
partition_column	Any	Partition for statistics calculation.

## TD\_UnivariateStatistics Output

### Output Table Schema

Column	Data Type	Description
partition_column	Any	Column copied from input table. Defines a partition for statistics calculation.
Attribute	VARCHAR	Target column for function calculated statistics.
StatsName	VARCHAR	[Column appears once for each specified <i>statistic</i> .] Statistic.
StatsValue	DOUBLE PRECISION	[Column appears once for each specified <i>statistic</i> .] Statistic value.

## Example: Using TD\_UnivariateStatistics to Determine Mean, Median, and Mode

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_UnivariateStatistics InputTable: titanic\_train

passenger	survived	name	gender	age	fare
97	0	Goldschmidt; Mr. George B	male	71	34.6542
488	0	Kent; Mr. Edward Austin	male	58	29.7
505	1	Maioni; Miss. Roberta	female	16	86.5
631	1	Barkworth; Mr. Algernon Henry Wilson	male	80	30
873	0	Carlsson; Mr. Frans Olof	male	33	5

## TD\_UnivariateStatistics SQL Call

```
SELECT * FROM TD_UnivariateStatistics (
    ON titanic_train AS InputTable
    USING
        TargetColumns ('age', 'fare')
        Stats ('MEAN', 'MEDIAN', 'MODE')
) AS dt;
```

## TD\_UnivariateStatistics Output

ATTRIBUTE	StatName	StatValue
age	MEAN	5.16000000000000E 001
age	MEDIAN	5.80000000000000E 001
age	MODE	1.60000000000000E 001
fare	MEAN	3.71708400000000E 001
fare	MEDIAN	3.00000000000000E 001
fare	MODE	5.00000000000000E 000

## TD\_WichMax

TD\_WichMax displays all rows that have the maximum value in a specified input table column. The maximum value represents the upper limit of a quantity or set of values. For example, if we have a set of numbers such as {3, 5, 1, 7, 2}, the maximum value would be 7 since it is the largest number in the set.

Finding maximum values in a dataset is important for the following reasons:

- Identification of outliers: The maximum values in a dataset may indicate the presence of outliers, which are data points that are significantly different from the rest of the data. Outliers can skew statistical analyses and make it difficult to draw meaningful conclusions from the data. Identifying these outliers can help researchers to decide whether to remove them from the dataset or take other corrective actions.
- Understanding the range of values: Knowing the maximum value in a dataset can help researchers to understand the range of values that are present in the data. This can be useful for determining the accuracy and precision of measurements, identifying potential errors or biases in the data collection process, and determining the sensitivity of statistical analyses to small changes in the data.
- Establishing performance benchmarks: In certain contexts, such as sports or financial markets, the maximum values can be used to establish performance benchmarks. For example, the fastest time in a race or the highest stock price can serve as a benchmark for future performance. This can help individuals and organizations to set goals and measure their progress towards achieving them.
- Safety considerations: In some cases, the maximum values of certain parameters can be used to establish safety standards. For example, in engineering, the maximum load that a bridge can support

or the maximum wind speed that a building can withstand can be used to establish safety guidelines and ensure that structures are built to withstand expected conditions.

Finding maximum values in a dataset can help researchers to better understand the data, identify potential issues, and establish benchmarks and safety standards in a variety of contexts.

## Function Information

- [TD\\_WichMax Syntax](#)
- [Required Syntax Elements for TD\\_WichMax](#)
- [Optional Syntax Elements for TD\\_WichMax](#)
- [TD\\_WichMax Input](#)
- [TD\\_WichMax Output](#)
- [TD\\_WichMax Example](#)

## TD\_WichMax Syntax

```
TD_WichMax (
    ON { table | view | (query) } AS InputTable
        [ PARTITION BY ANY [ORDER BY order_by_column ] ]
    USING
        TargetColumn ('target_column')
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_WichMax

### ON clause

Accept the InputTable clause.

### TargetColumn

Specify the target column names to determine maximum values.

## Optional Syntax Elements for TD\_WichMax

### PARTITION BY ANY

Accept the order\_by\_column clause.

## TD\_WichMax Input

### InputTable Schema

Column	Data Type	Description
target_column	Any except BLOB and CLOB and UDT.	Columns for which maximum values are checked.

## TD\_WichMax Output

### Output Table Schema

Column	Data Type	Description
target_column	Any except BLOB and CLOB and UDT.	Columns for which maximum values are checked.

## TD\_WichMax Example

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_WichMax InputTable: titanic\_dataset

passenger	survived	pclass	gender	age	sibsp	parch	fare	cabin	embarked
1	0	3	male	22	1	0	7.25	null	S
2	1	1	female	38	1	0	71.28	C85	C
3	1	3	female	26	0	0	7.93	null	S
4	1	1	female	35	1	0	53.10	C123	S
5	0	3	male	35	0	0	8.05	null	S

### Example: TD\_WichMax SQL Call for Maximum Fare

```
SELECT * FROM TD_WichMax (
    ON titanic_dataset AS InputTable
```

```
USING
TargetColumn ('fare')
) AS dt;
```

## **TD\_WichMax Output for Maximum Fare**

passenger	survived	pclass	gender	age	sibsp	parch	fare	cabin	embarked
2	1	1	female	38	1	0	71.28	C85	C

## **TD\_WichMin**

TD\_WichMin displays all rows that have the minimum value in specified input table column. Minimum value represents the lower limit of a quantity or set of values. For example, if we have a set of numbers such as {3, 5, 1, 7, 2}, the minimum value would be 1 since it is the smallest number in the set.

Minimum values are often used in optimization problems to find the smallest possible value of a function within a given domain. This can be useful in various fields such as engineering, economics, and physics.

Finding the minimum value in a dataset is important for the following reasons:

- Quality Control: Identify potential errors in the data. For example, if you have a dataset of temperatures that range from -20°C to 40°C, but there is a minimum value of -100°C, then it is likely that there is an error in the data.
- Data Normalization: Normalize data by providing a baseline for comparison. By setting the minimum value to zero, you can create a normalized scale that allows for easy comparison between different datasets.
- Analysis: Identify outliers or unusual data points in the dataset. Outliers can have a significant impact on statistical analysis and can skew the results, so it's important to identify them.
- Machine Learning: Scale data when using machine learning. Scaling data involves transforming it to fit within a specific range, which often means setting the minimum value to zero.

Finding the minimum value in a dataset is an important step in data analysis, data normalization, and machine learning. It helps to ensure data quality and accuracy and provides a baseline for comparison and analysis.

### **Function Information**

- [TD\\_WichMin Syntax](#)
- [Required Syntax Elements for TD\\_WichMin](#)
- [Optional Syntax Elements for TD\\_WichMin](#)
- [TD\\_WichMin Input](#)
- [TD\\_WichMin Output](#)
- [TD\\_WichMin Example](#)

## TD\_WichMin Syntax

```
TD_WichMin (
    ON { table | view | (query) } AS InputTable
        [ PARTITION BY ANY [ORDER BY order_by_column] ]
    USING
        TargetColumn ('target_column')
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_WichMin

**ON clause**

Accept the InputTable clause.

**TargetColumn**

Specify the target column names to check for minimum values.

## Optional Syntax Elements for TD\_WichMin

**PARTITION BY ANY**

Accept the *order\_by\_column* clause.

## TD\_WichMin Input

**InputTable Schema**

Column	Data Type	Description
<i>target_column</i>	Any except BLOB and CLOB and UDT.	Columns for which minimum values are checked.

## TD\_WichMin Output

### Output Table Schema

Column	Data Type	Description
target_column	Any except BLOB and CLOB and UDT.	Columns for which minimum values are checked.

## TD\_WichMin Example

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_WichMin InputTable: titanic\_dataset

passenger	survived	pclass	gender	age	sibsp	parch	fare	cabin	embarked
1	0	3	male	22	1	0	7.25	null	S
2	1	1	female	38	1	0	71.28	C85	C
3	1	3	female	26	0	0	7.93	null	S
4	1	1	female	35	1	0	53.10	C123	S
5	0	3	male	35	0	0	8.05	null	S

### Example: TD\_WichMin SQL Call for Minimum Fare

```
SELECT * FROM TD_WichMin (
  ON titanic_dataset AS InputTable
  USING
    TargetColumn ('fare')
) AS dt;
```

### TD\_WichMin Output for Minimum Fare

passenger	survived	pclass	gender	age	sibsp	parch	fare	cabin	embarked
1	0	3	male	22	1	0	7.25	null	S

The output shows the row that has the lowest value in the fare column.

# Feature Engineering Transform Functions

## Antiselect

AntiSelect returns all columns except those specified.

## TD\_BinCodeFit

Converts numeric data to categorical data by binning the numeric data into multiple numeric bins (intervals).

## TD\_BinCodeTransform

Transforms input table columns from the BinCodeFit function output.

## TD\_ColumnTransformer

Transforms the input table columns in a single operation.

## TD\_FunctionFit

Determines whether specified numeric transformations can be applied to specified input columns.

## TD\_FunctionTransform

Applies numeric transformations to input columns.

## TD\_NonLinearCombineFit

Returns the target columns and a specified formula which uses the non-linear combination of existing features.

## TD\_NonLinearCombineTransform

Generates the values of the new feature using the specified formula from the TD\_NonLinearCombineFit function output.

## TD\_OneHotEncodingFit

Outputs a table of attributes and categorical values to the TD\_OneHotEncodingTransform function.

## TD\_OneHotEncodingTransform

Encodes specified attributes and categorical values as one-hot numeric vectors using the output from the TD\_OneHotEncodingFit function.

## TD\_OrdinalEncodingFit

Encodes specified attributes and categorical values as one-hot numeric vectors using the output from the TD\_OneHotEncodingFit function.

**TD\_OrdinalEncodingTransform**

Maps the categorical value to a specified ordinal value using the TD\_OrdinalEncodingFit output.

**TD\_Pivoting**

Pivots the data, that is, changes the data from sparse to dense format.

**TD\_PolynomialFeaturesFit**

Stores all the specified values in the argument in a tabular format.

**TD\_PolynomialFeaturesTransform**

Extracts values of arguments from the output of the TD\_PolynomialFeaturesFit function and generates a feature matrix of all polynomial combinations of the features.

**TD\_RandomProjectionFit**

Returns a random projection matrix based on the specified arguments.

**TD\_RandomProjectionMinComponents**

Calculates the minimum number of components required for applying RandomProjection on the given dataset for the specified epsilon(distortion) parameter value.

**TD\_RandomProjectionTransform**

Converts the high-dimensional input data to a lower-dimensional space using the TD\_RandomProjectionFit function output.

**TD\_RowNormalizeFit**

Outputs a table of parameters and specified input columns to TD\_RowNormalizeTransform which normalizes the input columns row-wise.

**TD\_RowNormalizeTransform**

Normalizes the input columns row-wise using the output of the TD\_RowNormalizeFit function.

**TD\_ScaleFit**

Outputs a table of statistics to the TD\_ScaleTransform function.

**TD\_ScaleTransform**

Scales the specified input table columns using the output of the TD\_ScaleFit function.

**TD\_TargetEncodingFit**

Takes the InputTable and a CategoricalTable as input and generates the required hyperparameters to be used by the TD\_TargetEncodingTransform function for encoding the categorical values.

**TD\_TargetEncodingTransform**

Takes the InputTable and a FitTable generated by TD\_TargetEncodingFit for encoding the categorical values.

**TD\_Unpivoting**

Unpivots the data, that is, changes the data from dense format to sparse format.

## Antiselect

Antiselect returns all columns *except* those specified in the Exclude syntax element.

**Note:**

- This function requires the UTF8 client character set for UNICODE data.

### Function Information

- [Antiselect Syntax](#)
- [Antiselect Syntax Elements](#)
- [Antiselect Input](#)
- [Antiselect Output](#)
- [Antiselect Examples](#)

## Antiselect Syntax

```
Antiselect (
    ON { table | view | (query) }
    USING
    Exclude ({ 'exclude_column' | exclude_column_range }[,...])
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Antiselect Syntax Elements

### Exclude

Specify the names of the input table columns to exclude from the output table. Column names must be valid object names, which are defined in *Teradata Vantage™ - SQL Fundamentals*, B035-1141.

The *exclude\_column* is a column name. This is the syntax of *exclude\_column\_range*:

```
'start_column:end_column' [, '-exclude_in-range_column' ]
```

The range includes its endpoints.

The *start\_column* and *end\_column* can be:

- Column names (for example, 'column1:column2')

Column names must contain only letters in the English alphabet, digits, and special characters. If a column name includes any special characters, surround the column name with double quotation marks. For example, if the column name is a\*b, specify it as "a\*b". A column name cannot contain a double quotation mark.

- Nonnegative integers that represent the indexes of columns in the table (for example, '[0:4]')

The first column has index 0; therefore, '[0:4]' specifies the first five columns in the table.

- Empty. For example:

- '[ :4 ]' specifies all columns up to and including the column with index 4.
- '[ 4: ]' specifies the column with index 4 and all columns after it.
- '[ : ]' specifies all columns in the table.

The *exclude\_in-range\_column* is a column in the specified range, represented by either its name or its index (for example, '[0:99]', '-[50]', '-column10' specifies the columns with indexes 0 through 99, except the column with index 50 and column10).

Column ranges cannot overlap, and cannot include any specified *exclude\_column*.

## Antiselect Input

The input table can have any schema.

## Antiselect Output

The output table has all input table columns except those specified by the Exclude syntax element.

## Antiselect Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Antiselect Example: No Column Ranges](#)
- [Antiselect Example: Column Range](#)

### Antiselect Example: No Column Ranges

#### Antiselect Input with No Column Ranges

The input table, antiselect\_test, is a sample set of sales data containing 13 columns.

antiselect_test												
sno	id	orderdate	priority	qty	sales	disct	dmode	custname	province	region	cus	
1	3	2010-10-13 00:00:00	Low	6	261.54	0.04	Regular Air	Muhammed MacIntyre	Nunavut	Nunavut	Smart Bus	
49	293	2012-10-01 00:00:00	High	49	10123	0.07	Delivery Truck	Barry French	Nunavut	Nunavut	Con	
50	293	2012-10-01 00:00:00	High	27	244.57	0.01	Regular Air	Barry French	Nunavut	Nunavut	Con	
80	483	2011-07-10 00:00:00	High	30	4965.76	0.08	Regular Air	Clay Rozendal	Nunavut	Nunavut	Cor	
85	515	2010-08-28 00:00:00	Not specified	19	394.27	0.08	Regular Air	Carlos Soltero	Nunavut	Nunavut	Con	
86	515	2010-08-28 00:00:00	Not specified	21	146.69	0.05	Regular Air	Carlos Soltero	Nunavut	Nunavut	Con	
97	613	2011-06-17 00:00:00	High	12	93.54	0.03	Regular Air	Carl Jackson	Nunavut	Nunavut	Cor	

#### Antiselect SQL Call with No Column Ranges

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
  Exclude ('id', 'orderdate', 'disct', 'province', 'custsegment')
) AS dt ORDER BY 1, 4;
```

## AntiselectOutput with No Column Ranges

sno	priority	qty	sales	dmode	custname	region	prodcat
1	Low	6	2.61540000000000E002	Regular Air	Muhammed MacIntyre	Nunavut	Office Supplies
49	High	49	1.01230000000000E004	Delivery Truck	Barry French	Nunavut	Office Supplies
50	High	27	2.44570000000000E002	Regular Air	Barry French	Nunavut	Office Supplies
80	High	30	4.96576000000000E003	Regular Air	Clay Rozendal	Nunavut	Technology
85	Not specified	19	3.94270000000000E002	Regular Air	Carlos Soltero	Nunavut	Office Supplies
86	Not specified	21	1.46690000000000E002	Regular Air	Carlos Soltero	Nunavut	Furniture
97	High	12	9.35400000000000E001	Regular Air	Carl Jackson	Nunavut	Office Supplies

## Antiselect Example: Column Range

### Antiselect Input

The input table is antiselect\_test, as in [Antiselect Example: No Column Ranges](#).

### Antiselect SQL Call with Column Range

```
SELECT * FROM Antiselect (
  ON antiselect_test
  USING
  Exclude ('id', '[2:3]', 'custname:prodcat')
) AS dt ORDER BY 1, 4;
```

## Antiselect Output with Column Range

sno	qty	sales	disct	dmode
1	6	2.61540000000000E 002	0.04	Regular Air

sno	qty	sales	disct	dmode
49	49	1.01230000000000E 004	0.07	Delivery Truck
50	27	2.44570000000000E 002	0.01	Regular Air
80	30	4.96576000000000E 003	0.08	Regular Air
85	19	3.94270000000000E 002	0.08	Regular Air
86	21	1.46690000000000E 002	0.05	Regular Air
97	12	9.35400000000000E 001	0.03	Regular Air

## TD\_BinCodeFit

TD\_BinCodeFit outputs a table that you can use with [TD\\_BinCodeTransform](#), which converts the specified input table columns.

Bin Coding, also known as binning or bucketing, is a data preprocessing technique used in statistics and machine learning to transform continuous data into categorical or discrete data. In binning, a range of continuous values is divided into smaller intervals or bins, and the data values are assigned to the appropriate bin. This allows the data to be analyzed more easily by grouping similar values into categories.

Binning is used to:

- Reduce noise in the data by aggregating values into groups.
- Identify trends and patterns in the data by grouping similar values together.
- Simplify complex data by reducing the number of unique values.

There are several methods of binning data such as:

- Equal-width binning: In this method, the range of values is divided into a fixed number of intervals of equal width.
- Equal-frequency binning: In this method, the range of values is divided into a fixed number of intervals containing an equal number of data points. This is useful when the distribution of data is uneven. For example, if you have data ranging from 0 to 100 and you want to divide it into 5 bins, you would group the values so that each bin contains the same number of data points.
- Manual binning: In this method, you manually define the intervals or bins based on domain knowledge or specific requirements. This method provides more control over the grouping of values, but can be more subjective.

### Equal-width Bin Coding

The width of each bin is given by:

$$\text{bin width} = (\text{max value} - \text{min value}) / \text{number of bins}$$

where max value and min value are the maximum and minimum values of the data, respectively, and the number of bins is the desired number of intervals.

Example:

Suppose you have the following dataset of 20 values:

5, 7, 10, 13, 18, 19, 23, 25, 26, 28, 32, 34, 35, 38, 40, 41, 45, 47, 49, 52

Divide this data into five bins of equal width. First, find the minimum value = 5 and maximum value = 52.

Calculate the bin width using the formula:

bin width = (max value - min value) / number of bins

(52 - 5) divided by 5 results in 9.4.

Since you cannot have decimal intervals, you can round up to 10. Therefore, the bin width is 10. You can create 5 bins of width 10 as follows:

- Bin 1: 5-14
- Bin 2: 15-24
- Bin 3: 25-34
- Bin 4: 35-44
- Bin 5: 45-54

Assign each data point to the appropriate bin. For example, the value 18 falls into the second bin (15-24), while the value 41 falls into the fifth bin (45-54). This process results in the data being transformed into five discrete categories, which can be used for further analysis or visualization.

## **Manual-width Bin Coding**

Manual bin coding involves defining the intervals or bins based on domain knowledge or specific requirements. The mathematical formulation of manual binning has you define the intervals or bins, and then assign the data values to the appropriate bin.

For example, you have the following dataset of 20 values:

5, 7, 10, 13, 18, 19, 23, 25, 26, 28, 32, 34, 35, 38, 40, 41, 45, 47, 49, 52

Divide this data into four bins based on domain knowledge, and bin the data as follows:

- Bin 1: 0-10
- Bin 2: 11-25
- Bin 3: 26-40
- Bin 4: 41-52

Assign each data point to the appropriate bin. For example, the value 18 falls into the second bin (11-25), while the value 41 falls into the fourth bin (41-52). This process results in the data being transformed into four discrete categories based on our domain knowledge.

When binning data, it's important to consider the appropriate number of bins and the width of each bin. If the bins are too narrow, the data may appear noisy and difficult to interpret. On the other hand, if the bins are too wide, important information may be lost. Binning is a useful technique for simplifying and summarizing complex data, but it must be used with care to ensure that the resulting data is still meaningful and informative.

Bin Coding can help reduce noise, simplify complex data, and identify patterns, but it requires careful consideration of the number of bins and the width of each bin to ensure meaningful and informative results.

## Function Information

- [TD\\_BinCodeFit Syntax](#)
- [Required Syntax Elements for TD\\_BinCodeFit](#)
- [Optional Syntax Elements for TD\\_BinCodeFit](#)
- [TD\\_BinCodeFit Input](#)
- [TD\\_BinCodeFit Output](#)
- [Example: How To Use TD\\_BinCodeFit](#)

## TD\_BinCodeFit Syntax

### For Equal-Width Bins with Generated Labels

```
TD_BincodeFit (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
    USING
        TargetColumns ({ 'target_column' | 'target_column_range' }[,...])
        MethodType ('equal-width')
        NBins ('single bin value' | 'separate bin values')
        [ LabelPrefix ('single prefix' | 'separate prefix values') ]
)
```

### For Variable-Width Bins with Specified Labels

```
TD_BincodeFit (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS FitInput DIMENSION
    USING
        TargetColumns ({ 'target_column' | 'target_column_range' }[,...])
        MethodType ('variable-width')
        [ MinValueColumn ('minvalue_column') ]
        [ MaxValueColumn ('maxvalue_column') ]
        [ LabelColumn ('label_column') ]
        [ TargetColNames ('target_names_column') ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_BinCodeFit

### ON clause

- Accept the InputTable clause for equal-width bins with generated labels.
- Accept the InputTable and FitInput clauses for variable-width bins with specified labels.

### TargetColumns

Specify the names of the InputTable columns to bin-code.

The maximum number of target columns is 2018.

### MethodType

Specify the bin-coding method:

Method Type	Description
equal-width	Bins have fixed width with auto-generated labels. Function finds minimum and maximum values of target columns ( <i>min</i> and <i>max</i> ) and computes bin width, <i>w</i> , with this formula: $w = (max - min)/k$ <i>k</i> , the number of bins, is determined by NBins.
variable-width	Bins have specified variable widths and specified labels, provided by FitInput table. Maximum number of bins is 3000.

### NBins

[MethodType ('equal-width') only.] Specify either a single bin value for all the target columns or separate bin values for each of the target columns.

## Optional Syntax Elements for TD\_BinCodeFit

### OUT clause

Accept the OutputTable clause for equal-width bins with generated labels.

### OutputTable

[MethodType ('equal-width') only.] Specify a name for the secondary output table which contains the actual bins used by the [TD\\_BinCodeTransform](#) function.

### LabelPrefix

[MethodType ('equal-width') only.] Specify either a prefix for all the target columns or a separate prefix for each of the target columns.

Lower Bin Boundary	Upper Bin Boundary	<i>prefix</i> -Generated Bin Label	<i>prefix_count</i> -Generated Bin Label
<i>min</i>	<i>min + w</i>	<i>prefix_1</i>	<i>target_column_1</i>
<i>min + w</i>	<i>min + 2w</i>	<i>prefix_2</i>	<i>target_column_2</i>
...	...	...	...
<i>min + kw</i>	<i>min + (k - 1)w</i>	<i>prefix_k</i>	<i>target_column_k</i>

Default: Target column name is used as the prefix

### MinValueColumn

[MethodType ('variable-width') only.] Specify the name of the FitInput column that has the minimum value of the bin (lower bin boundaries).

Default: MinValue

### MaxValueColumn

[MethodType ('variable-width') only.] Specify the name of the FitInput column that has the maximum value of the bin (upper bin boundaries).

Default: MaxValue

### LabelColumn

[MethodType ('variable-width') only.] Specify the name of the FitInput column that has the bin labels.

Default: Label

**TargetColNames**

[MethodType ('variable-width') only.] Specify the name of the FitInput column that has the target column names.

**Note:**

Column range is not supported

**TD\_BinCodeFit Input****InputTable Schema**

Column	Data Type	Description
target_column	BYTEINT,SMALLINT,INTEGER,BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Column to bin-code.

**FitTable Schema**

Required with specify MethodType ('variable-width'), ignored otherwise.

Column	Data Type	Description
target_names_column	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE)	Bin column name.
minvalue_column	BYTEINT,SMALLINT,INTEGER,BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Minimum value of the bin.
maxvalue_column	BYTEINT,SMALLINT,INTEGER,BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Maximum value of the bin.
label_column	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE)	Bin labels.

**TD\_BinCodeFit Output****Output Table Schema**

Column	Data Type	Description
TD_ColumnName_BINFIT	VARCHAR (CHARACTER SET UNICODE)	Bin column name.
TD_MinValue_BINFIT	DOUBLE PRECISION	Minimum value of the bin.

Column	Data Type	Description
TD_MaxValue_BINFIT	DOUBLE PRECISION	Maximum value of the bin.
TD_LabelPrefix_BINFIT	VARCHAR (CHARACTER SET UNICODE)	[Column appears only with MethodType ('equal-width')] Label prefix.
TD_Label_BINFIT	VARCHAR (CHARACTER SET UNICODE)	[Column appears only with MethodType ('variable-width')] Bin label.
TD_Bins_BINFIT	INTEGER	Bin count.
TD_IndexValue_BINFIT	SMALLINT	Index value.
TD_MaxLenLabel_BINFIT	SMALLINT	Maximum bin label length.
target_column	Same as in Input table	Target column for TD_BinCodeTransform.

## OutputTable Schema (Secondary Output Table)

The function outputs this secondary output table only if you specify MethodType ('equal-width').

Column	Data Type	Description
ColumnName	VARCHAR (CHARACTER SET UNICODE)	Column name of the bin.
MinValue	DOUBLE PRECISION	Minimum value of the bin.
MaxValue	DOUBLE PRECISION	Maximum value of the bin.
label	VARCHAR (CHARACTER SET UNICODE)	Label of the bin.

## Example: How To Use TD\_BinCodeFit

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_BinCodeFit InputTable: bin\_titanic\_train

passenger	survived	pclass	name	gender	age
sibsp	parch	ticket	fare	cabin	embarked
-----	-----	-----	-----	-----	-----
97	0	1	Goldschmidt; Mr. George B		male 71

0	0	PC	17754	34.654200000	A5	C		
	488		0	1	Kent; Mr. Edward Austin		male	58
0	0	11771	29.700000000	B37	C			
	505		1	1	Maioni; Miss. Roberta		female	16
0	0	110152	86.500000000	B79	S			
	631		1	1	Barkworth; Mr. Algernon Henry Wilson	male	80	
0	0	27042	30.000000000	A23	S			
	873		0	1	Carlsson; Mr. Frans Olof	male	33	
0	0	695	5.000000000	B51	B53	B55	S	

**TD\_BinCodeFit Input: FitInput table**

ColumnName	MinValue	MaxValue	Label
age	0.00	20.00	Young Age
age	21.00	45.00	Middle Age
age	46.00	90.00	Old Age

**Example: Using Age Column with TD\_BinCodeFit SQL Call**

```
CREATE TABLE FitOutputTable AS (
  SELECT * FROM TD_BincodeFit (
    ON bin_titanic_train AS InputTable
    ON FitInputTable AS FitInput DIMENSION
    USING
    TargetColumns ('age')
    MethodType ('Variable-Width')
    MinValueColumn ('MinValue')
    MaxValueColumn ('MaxValue')
    LabelColumn ('Label')
    TargetColNames ('ColumnName')
  ) AS dt
) WITH DATA;
```

**Example: Using Variable Width for Target Columns with TD\_BinCodeFit SQL Call**

```
CREATE TABLE FitOutputTable AS (
  SELECT * FROM TD_BincodeFit (
    ON bin_titanic_train AS InputTable
    ON FitInputTable AS FitInput DIMENSION
    USING
    TargetColumns ('[5]')
    MethodType ('Variable-Width')
```

```

    MinValueColumn ('MinValue')
    MaxValueColumn ('MaxValue')
    LabelColumn ('Label')
    TargetColNames ('ColumnName')
) AS dt
) WITH DATA;

```

## TD\_BinCodeFit Output

	TD_ColumnName_BINFIT	TD_MinValue_BINFIT	TD_MaxValue_BINFIT	TD_Label_BINFIT	TD_Bins_BINFIT	TD_IndexValue_BINFIT
	TD_MaxLenLabel_BINFIT	age				
0	age	10 null	46.000000000	90.000000000 Old Age		3
0	age	10 null	21.000000000	45.000000000 Middle Age		3
0	age	10 null	0.000000000	20.000000000 Young Age		3

## TD\_BinCodeTransform

TD\_BinCodeTransform is a data transformation technique used to convert continuous numerical data into categorical data. The process involves dividing the numerical data into multiple intervals or bins and assigning categorical labels to each bin. You can use the resulting categorical data for further analysis or modeling, and can help you to simplify and reduce the complexity of the data.

TD\_BinCodeTransform follows this process:

1. Determines the number and size of the bins.
2. Assigns the data to the bin based on its value. This process of assigning data values to bins is known as binning.
3. Assigns a categorical label to each bin, which the function can auto-generate based on the bin boundaries or your inputs.

See [TD\\_BinCodeTransform Usage Notes](#).

TD\_BinCodeTransform bin-codes input table columns, using [TD\\_BinCodeFit](#) output.

## Function Information

- [TD\\_BinCodeTransform Syntax](#)
- [Required Syntax Elements for TD\\_BinCodeTransform](#)
- [Optional Syntax Elements for TD\\_BinCodeTransform](#)
- [TD\\_BinCodeTransform Input](#)
- [TD\\_BinCodeTransform Output](#)
- [TD\\_BinCodeTransform Usage Notes](#)
- [Example: How to Use TD\\_BinCodeTransform](#)

## TD\_BinCodeTransform Syntax

```
TD_BinCodeTransform (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS FitTable DIMENSION
    USING
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_BinCodeTransform

**ON clause**

Accepts the InputTable and FitTable ON clauses.

## Optional Syntax Elements for TD\_BinCodeTransform

**Accumulate**

Names of InputTable columns to copy to the output table.

## TD\_BinCodeTransform Input

**InputTable Schema**

See [TD\\_BinCodeFit Input](#).

**FitTable Schema**

See [TD\\_BinCodeFit Output](#).

## TD\_BinCodeTransform Output

### Output Table Schema

Column	Data Type	Description
<i>accumulate_column</i>	Same as in InputTable.	Column copied from InputTable.
<i>target_column</i>	VARCHAR (CHARACTER SET UNICODE)	[Column appears once for each specified <i>target_column</i> .] Bin labels.

## TD\_BinCodeTransform Usage Notes

The first step in TD\_BinCodeTransform is to determine the number and size of the bins. The bins can be of equal width or variable width, depending on the nature of the data. For example, if you have a dataset of ages ranging from 0 to 100, you can create bins of width 10 units each, resulting in 10 bins.

After determining the bin size, the TD\_BinCodeTransform assigns the data to the appropriate bin based on its value. For example, TD\_BinCodeTransform assigns an age value of 37 to the 4th bin in the age range of 30-39. This process of assigning data values to bins is also known as binning.

After binning the data, the TD\_BinCodeTransform transforms the numerical data into categorical data. The TD\_BinCodeTransform assigns each bin a categorical label, which the TD\_BinCodeTransform can auto-generate based on the bin boundaries or may be specified by the user. For example, if TD\_BinCodeTransform binned the ages into 10-year intervals, the categorical labels can be "0-9 years", "10-19 years", and so on.

You can use the categorical data for further analysis or modeling. For example, you can generate a frequency distribution or histogram of the binned data to visualize the distribution of values. In machine learning, you can use the categorical data as input to models that require categorical data, such as decision trees or random forests.

Binning has advantages and may make the data easier to analyze, particularly when dealing with large datasets. Binning can also help address issues of overfitting by reducing the number of unique values in the data and smoothing out variations in the data.

However, you need to carefully choose the bin sizes and boundaries, as they can impact the accuracy and interpretability of the results. If the bins are too wide, you can lose valuable information, while if the bins are too narrow, your data can become too sparse, leading to overfitting.

## Example: How to Use TD\_BinCodeTransform

- InputTable: bin\_titanic\_train, as in [Example: How To Use TD\\_BinCodeFit](#)
- FitTable: FitOutputTable, created by [Example: How To Use TD\\_BinCodeFit](#)

## TD\_BinCodeTransform SQL Call

```
SELECT * FROM TD_BinCodeTransform (
    ON bin_titanic_train AS InputTable
    ON FitOutputTable AS FitTable DIMENSION
    USING
        Accumulate ('passenger')
) AS dt;
```

## TD\_BinCodeTransform Output

```
passenger age
-----
505 Young Age
631 Old Age
97 Old Age
488 Old Age
873 Middle Age
```

## TD\_ColumnTransformer

The TD\_ColumnTransformer function transforms the input table columns in a single operation. You only need to provide the FIT tables to the function, and the function runs all transformations that you require in a single operation.

The function performs the following transformations:

- TD\_Scale Transform
- TD\_Bincode Transform
- TD\_Function Transform
- TD\_NonLinearCombine Transform
- TD\_OutlierFilter Transform
- TD\_PolynomialFeatures Transform
- TD\_RowNormalize Transform
- TD\_OrdinalEncoding Transform
- TD\_OneHotEncoding Transform
- TD\_SimpleImpute Transform

TD\_ColumnTransformer supports multiple instances of same FitTable for NonLinearCombineFitTable. For all other types, only single instance is allowed.

You must create the FIT tables before using the function and you must provide the FIT tables in the same order as in the training data sequence to transform the dataset. The FIT tables must have a maximum of 128 columns.

**Note:**

The TD\_BincodeFit function has a maximum of 5 columns when using the variable-width method.

**Function Information**

- [TD\\_ColumnTransformer Syntax](#)
- [TD\\_ColumnTransformer Syntax Elements](#)
- [TD\\_ColumnTransformer Input](#)
- [TD\\_ColumnTransformer Output](#)
- [TD\\_ColumnTransformer Example](#)

**TD\_ColumnTransformer Syntax**

```
TD_ColumnTransformer (
  ON { table | view | (query) } AS InputTable
  [ ON { table | view | (query) } AS BincodeFitTable DIMENSION ]
  [ ON { table | view | (query) } AS FunctionFitTable DIMENSION ]
  [ ON { table | view | (query) } AS NonLinearCombineFitTable DIMENSION ]
  [ ON { table | view | (query) } AS OneHotEncodingFitTable DIMENSION ]
  [ ON { table | view | (query) } AS OrdinalEncodingFitTable DIMENSION ]
  [ ON { table | view | (query) } AS OutlierFilterFitTable DIMENSION ]
  [ ON { table | view | (query) } AS PolynomialFeaturesFitTable DIMENSION ]
  [ ON { table | view | (query) } AS RowNormalizeFitTable DIMENSION ]
  [ ON { table | view | (query) } AS ScaleFitTable DIMENSION ]
  [ ON { table | view | (query) } AS SimpleImputeFitTable DIMENSION ]
  USING
  [FillRowIDColumnName('output_column_name')]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## TD\_ColumnTransformer Syntax Elements

### FillRowIdColumnName

[Optional] Name for the output column in which unique identifiers for each row are populated.  
It adds a column to the data.

## TD\_ColumnTransformer Input

Column	Data Type	Description
TargetColumn	<ul style="list-style-type: none"> <li>CHAR or VARCHAR for categorical columns</li> <li>INTEGER, REAL, DECIMAL, or NUMBER for numeric columns</li> </ul>	<p>The input table columns that requires transformation based on the FIT table.</p> <p>Functions with categorical columns:</p> <ul style="list-style-type: none"> <li>TD_OrdinalEncoding Fit</li> <li>TD_OneHotEncoding Fit</li> </ul> <p>Functions with numeric columns:</p> <ul style="list-style-type: none"> <li>TD_Scale Fit</li> <li>TD_Bincode Fit</li> <li>TD_Function Fit</li> <li>TD_NonLinearCombine Fit</li> <li>TD_OutlierFilter Fit</li> <li>TD_PolynomialFeatures Fit</li> <li>TD_RowNormalize Fit</li> <li>TD_SimpleImpute Fit</li> </ul>

## TD\_ColumnTransformer Output

### TD\_ColumnTransformer Output

Column	Data Type	Description
TargetColumn	<ul style="list-style-type: none"> <li>CHAR or VARCHAR for categorical columns</li> <li>INTEGER, REAL, DECIMAL, or NUMBER for numeric columns</li> </ul>	The columns transformed using the ColumnTransformer function.
otherColumns	<ul style="list-style-type: none"> <li>CHAR or VARCHAR for categorical columns</li> <li>INTEGER, REAL, DECIMAL, or NUMBER for numeric columns</li> </ul>	The default columns from input to output.
NewColumns	<ul style="list-style-type: none"> <li>CHAR or VARCHAR for categorical columns</li> <li>INTEGER, REAL, DECIMAL, or NUMBER for numeric columns</li> </ul>	<p>The following FIT functions produce new columns:</p> <ul style="list-style-type: none"> <li>TD_Polynomial FeaturesFit</li> <li>TD_NonLinearCombineFit</li> </ul>

Column	Data Type	Description
		<ul style="list-style-type: none"> <li>TD_OneHotEncodingFit</li> </ul> <p>The following functions change the schema of the target column:</p> <ul style="list-style-type: none"> <li>TD_BincodeFit</li> <li>TD_OrdinalEncodingFit</li> </ul>

## TD\_ColumnTransformer Example

### TD\_ColumnTransformer Input Table: titanic\_train

PassengerID	Pclass	Name	Gender	Age	SibSp	Parch	Fare	Cabin	Embarked
149	2	Navratil, Michael	M	36	0	2	26.0	B21	S
152	1	Pearson, Mrs. Thomas	F	Null	1	0	66.6	C2	S
581	2	Christian, Miss Juliana	F	25	1	1	30.0	Null	S
663	1	Collier, Dr. Edwin	M	47	0	0	25.70	A23	S
704	3	Gavin, Mr. Herbert	M	25	0	0	7.74	Null	Q

### Create getCabin Input table

```

drop table getSubtitles;
create multiset table getSubtitles as (
select * from Unpack(
on titanic_train
Using
TargetColumn('Name')
OutputColumns('NTtitle')
OutputDatatypes('Varchar')
Delimiter('$')
Regex('([A-Za-z]+)\.\.')
)as dt)with data;

drop table getCabin;
create multiset table getCabin as (

```

```

SELECT * FROM TD_strApply (
ON getSubtitles as inputtable
USING
TargetColumns ('cabin')
StringOperation('getNchars')
StringLength(1)
Accumulate('[::]', '-cabin')
InPlace('True')
) as dt)with data;

```

## TD\_ColumnTransformer SQL Call

```

SELECT * FROM TD_ColumnTransformer(
ON getCabin AS inputtable
ON imputeFit AS SimpleImputeFitTable dimension
ON NonLinearCombineFit AS NonLinearCombineFitTable dimension
ON ordinalFit AS OrdinalEncodingFitTable dimension
ON onehotfittable AS OneHotEncodingFitTable dimension
ON ScaleFit AS ScaleFitTable dimension
)AS dt ORDER BY 1,2,3,4,5,6,7;

```

## TD\_ColumnTransformer Output

NTitle	passenger	survived	fare	pclass	embarked	gender	age	FamilySize	sibsp	parch	cabin_A	cabin_B
ticket												
cabin_C	cabin_other											
4.11356604308324E-002	1	-1	8	0	2	?	3	5.00000000000000E 000	1	2	0	3
5.6848213999047E-002	1	-1	17	0	1	?	3	6.00000000000000E 000	1	2	0	4
5.85561002574126E-002	0	....	....	....	....	....	....	....	....	....	1	349909
6607	1	5	888	1	2	B	1	1.00000000000000E 000	2	19	0	0
0	0	4.57713517012109E-002	0	3	2	?	2	4.00000000000000E 000	28	0	1	0
												W./C.

Comparison of serial processing of the functions to TD\_ColumnTransformer function based on size of data set:

Data Set	Serial Processing in Seconds	TD_ColumnTransformer Processing in Seconds
10M	89	29
20M	167	49
30M	332	98

## TD\_FunctionFit

TD\_FunctionFit determines whether specified numeric transformations can be applied to specified input columns and outputs a table to use as input to [TD\\_FunctionTransform](#), which does the transformations.

### Function Information

- [TD\\_FunctionFit Syntax](#)
- [TD\\_FunctionFit Input](#)
- [TD\\_FunctionFit Output](#)
- [TD\\_FunctionFit Example](#)

#### Related Information:

[TD\\_NumApply](#)

## TD\_FunctionFit Syntax

```
CREATE TABLE output_table AS (
    TD_FunctionFit (
        ON { table | view | (query) } AS InputTable
        ON { table | view | (query) } AS TransformationTable DIMENSION
    )
) WITH DATA;
```

---

#### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## TD\_FunctionFit Input

### InputTable Schema

Column	Data Type	Description
<i>input_column</i>	VARCHAR (CHARACTER SET LATIN or UNICODE) or NUMERIC	Column whose name can appear as TargetColumn in TransformationTable.

## TransformationTable Schema

Column	Data Type	Description
TargetColumn	VARCHAR (CHARACTER SET LATIN or UNICODE)	Name of InputTable column to transform.
Transformation	VARCHAR (CHARACTER SET LATIN or UNICODE)	Transformation to apply to TargetColumn—for allowed transformations, see following table.
Parameters	VARCHAR (CHARACTER SET LATIN or UNICODE)	[Optional] Transformation parameters in JSON format. If this column is absent and transformation has parameter, function uses default value in Parameters column of the following table.
DefaultValue	NUMERIC	[Optional] Default value for transformed value if TargetColumn is nonnumeric or NULL. If this column is absent, function uses default value 0.

## Allowed Transformations

If a transformation requires parameters, they are specified in JSON format as mentioned for log and power in the following table.

Transformation	Parameter	Operation on TargetColumn Value x
ABS	None	$ x $
CEIL	None	CEIL (x) (Least integer $\geq x$ )
EXP	None	$e^x$ ( $e = 2.718$ )
FLOOR	None	FLOOR (x) (Greatest integer $\leq x$ )
LOG	[Optional] {"base": base} Default: e	$\text{LOG}_{\text{base}}(x)$
POW	[Optional] {"exponent": exponent} Default: 1	$x^{\text{exponent}}$
SIGMOID	None	$1 / (1 + e^{-x})$
TANH	None	$(e^x - e^{-x}) / (e^x + e^{-x})$

## TD\_FunctionFit Output

### Output Table Schema

Same as TransformationTable schema (see [TD\\_FunctionFit Input](#)).

## TD\_FunctionFit Example

### TD\_FunctionFit Input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

InputTable: function\_input\_table

passenger	survived	pclass	name	cabin	embarked	gender	age	sibsp	parch
ticket									
21171	1	0	3 Braund; Mr. Owen Harris		S	male	22	1	0 A/5
17599	2	1	1 Cumings; Mrs. John Bradley (Florence Briggs Thayer)	C85	C	female	38	1	0 PC
3101282	3	1	3 Heikkinen; Miss. Laina		S	female	26	0	0 STON/O2.
113803	4	1	1 Futrelle; Mrs. Jacques Heath (Lily May Peel)		S	female	35	1	0
373450	5	0	3 Allen; Mr. William Henry	C123	S	male	35	0	0
		8.05000000			S				

### TD\_FunctionFit SQL Call

```
CREATE TABLE fit_out AS (
  SELECT * FROM TD_FunctionFit (
    ON function_input_table AS InputTable
    ON transformations AS TransformationTable DIMENSION
  ) AS dt
) WITH DATA;
```

### TD\_FunctionFit Output

TargetColumn	Transformation	Parameters	Defaultvalue
age	LOG	{"base":2}	0.000000000
fare	POW	{"exponent": 2}	10.000000000

## TD\_FunctionTransform

Function transformations play a critical role in the machine learning pipeline. These transformations involve applying mathematical functions to columns of data to create new variables that can help improve the accuracy and robustness of machine learning models.

Some of the most commonly used transformations include absolute, log, exponential, ceil, floor, sigmoid, and tanh transformations. These transformations serve a variety of purposes, such as:

- Data normalization: Transformations can normalize the data by transforming it into a more standardized form. For example, taking the log of a variable can scale down larger values while keeping smaller values relatively unchanged. Additionally, sigmoid and tanh functions be used to map the data into a range between 0 and 1 or -1 and 1, respectively. This reduces the impact of outliers and makes the data more consistent.
- Handling non-linear relationships: In some cases, the relationship between a variable and the target variable may not be linear. Transforming the variable using a non-linear transformation (such as the log or exponential functions) can capture these non-linear relationships and improve the accuracy of the model.
- Mitigating skewness: Skewed data can cause issues for machine learning algorithms, particularly those that assume a normal distribution. Log, power, sigmoid or tanh transformations can balance imbalanced data by compressing the values of the majority class towards the center of the distribution and expanding the values of the minority class towards the tails of the distribution. This can improve the performance of machine learning algorithms that are sensitive to skewed data.
- Addressing heteroscedasticity: In some cases, the variance of a variable may not be constant across different levels of the variable. Transforming the variable using log function can sometimes address this heteroscedasticity and improve the accuracy of the model.

By transforming columns of data into new variables that capture important information, machine learning models can make better predictions and achieve better performance across a wide range of applications.

`TD_FunctionTransform` applies numeric transformations to input columns, using [TD\\_FunctionFit](#) output.

## Function Information

- [TD\\_FunctionTransform Syntax](#)
- [TD\\_FunctionTransform Syntax Elements](#)
- [TD\\_FunctionTransform Input](#)
- [TD\\_FunctionTransform Output](#)
- [TD\\_FunctionTransform Example](#)

## TD\_FunctionTransform Syntax

```
TD_FunctionTransform (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS FitTable DIMENSION
    USING
    [ IDColumns ({ 'id_column' | id_column_range }[,...]) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## TD\_FunctionTransform Syntax Elements

### IDColumns

[Optional] Specify the names of the InputTable columns with NUMERIC datatypes to exclude from transformations. The columns with VARCHAR datatypes are automatically excluded.

No *id\_column* can be a *target\_column* in the TransformationTable for the TD\_FunctionFit call that output FitTable.

## TD\_FunctionTransform Input

### InputTable Schema

See [TD\\_FunctionFit Input](#).

### FitTable Schema

See [TD\\_FunctionFit Output](#).

## TD\_FunctionTransform Output

### Output Table Schema

Column	Data Type	Description
<i>input_column</i>	If NUMERIC in TD_FunctionFit InputTable: DOUBLE PRECISION Otherwise: Same as in TD_FunctionFit InputTable	Transformed values.

## TD\_FunctionTransform Example

### TD\_FunctionTransform Input

- InputTable: titanic\_data, as in [TD\\_FunctionFit Example](#)
- FitTable: fit\_out, created by [TD\\_FunctionFit Example](#)

## TD\_FunctionTransform SQL Call

```
SELECT * FROM TD_FunctionTransform (
    ON function_input_table AS InputTable
    ON fit_out AS FitTable DIMENSION
    USING
    IDColumns ('[0:2]', '[6:7]')
) AS dt ORDER BY Passenger;
```

## TD\_FunctionTransform Output

passenger name gender cabin	survived	pclass	embarked	age	sibsp	parch	ticket	fare
Harris 000	1	0		3	Braund; Mr. Owen			
							male	4.45943161863730E
							S	
Thayer) 17599	2	1	A/5 21171	5.25625000000000E 001	1 Cumings; Mrs. John Bradley (Florence Briggs			
					female	5.24792751344359E 000		
					C		1	0 PC
Laina 000	3	1	5.08130885889000E 003	C85	Heikkinen; Miss.			
						female	4.70043971814109E	
Peel) 113803	4	0	STON/O2. 3101282	6.28056250000000E 001	1 Futrelle; Mrs. Jacques Heath (Lily May			
					female	5.12928301694497E 000		
					S		1	0
Henry 000	5	0	2.81961000000000E 003	C123	Allen; Mr. William			
						male	5.12928301694497E	
						S		
		0	373450				6.48025000000000E 001	

## TD\_NonLinearCombineFit

TD\_NonLinearCombineFit function returns the target columns and a specified formula which uses the non-linear combination of existing features.

Feature engineering is the process of creating features from existing data to improve the performance of a machine learning model. One approach to feature engineering is to use a technique that combines existing features in a non-linear way to create ones that capture more complex relationships between the original features and the target.

By combining features in a non-linear way, this technique can create features that are not explicitly present in the original dataset. These features can help to simplify the model, reduce overfitting, and improve the interpretability of the model. Moreover, this technique can help to identify the most relevant features by evaluating the performance of the model with different combinations of the original features and their non-linear combinations.

In summary, feature engineering with non-linear combinations of features is an important technique for improving the performance and interpretability of machine learning models. This can help to create features, simplify the model, reduce overfitting, and identify the most relevant features for the target.

### Function Information

- [TD\\_NonLinearCombineFit Syntax](#)
- [Required Syntax Elements for TD\\_NonLinearCombineFit](#)

- [Optional Syntax Elements for TD\\_NonLinearCombineFit](#)
- [TD\\_NonLinearCombineFit Input](#)
- [TD\\_NonLinearCombineFit Output](#)
- [Example: How to Use TD\\_NonLinearCombineFit](#)

## TD\_NonLinearCombineFit Syntax

```
TD_NonLinearCombineFit (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable(output_table_name) ]
    USING
    TargetColumns ({'target_column' | 'target_column_range'}[,...])
    Formula ('Y = expression')
    [ ResultColumn ('result_column') ]
)
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_NonLinearCombineFit

**ON clause**

Specifies the table name, view name or query as an InputTable.

**TargetColumns**

Specifies the input table column names to run the non-linear combination.

**Formula**

Specifies the formula. See "Arithmetic, Trigonometric, Hyperbolic Operators/Functions" in *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145.

## Optional Syntax Elements for TD\_NonLinearCombineFit

**OUT clause**

Accepts the OutputTable clause.

**ResultColumn**

Specifies the new feature column name generated by the Transform function. The Fit function saves the specified formula in this column.

Default: TD\_CombinedValue.

**TD\_NonLinearCombineFit Input****Input Table Schema**

Column	Data Type	Description
TargetColumns	BYTEINT,SMALLINT,INTEGER,BIGINT,DECIMAL,NUMERIC,FLOAT,REAL, DOUBLE PRECISION	The input table column names to use in the non-linear combination.

**TD\_NonLinearCombineFit Output****Output Table Schema**

Column	Data Type	Description
ResultColumn	VARCHAR CHARACTER SET UNICODE	The Formula element displays in this column.
TargetColumns	BYTEINT,SMALLINT,INTEGER,BIGINT,DECIMAL,NUMERIC,FLOAT,REAL, DOUBLE PRECISION	The specified target columns displayed as NULL values.

**Example: How to Use TD\_NonLinearCombineFit****TD\_NonLinearCombineFit Input table**

	passenger	survived	pclass	gender	age	sibsp	parch	fare	cabin	embarked
null	1	0	General	male	22	1	0	7.250000000		
C85	2	S	Deluxe	female	38	1	1	71.280000000		
	3	C	General	female	26	0	0	7.930000000		
null	4	S	Deluxe	female	35	1	0	53.100000000		
C123	5	S	General	male	35	0	1	8.050000000	null	S

**Example: Determining Total Cost Using TD\_NonLinearCombineFit**

```
SELECT * FROM TD_NonLinearCombineFit (
  ON nonLinearCombineFit_input AS InputTable
  OUT TABLE FitTable (nonLinearCombineFit_output)
  USING
```

```

TargetColumns ('SibSp', 'Parch', 'Fare')
Formula ('Y=(X0+X1+1)*X2')
ResultColumn ('TotalCost')
) as dt order by 1;

```

## TD\_NonLinearCombineFit Output Table

total_cost	sibsp	parch	fare
-----	-----	-----	-----
Y=(X0+X1+1)*X2	null	null	null

## TD\_NonLinearCombineTransform

TD\_NonLinearCombineTransform generates the values of a feature using the specified formula from the TD\_NonLinearCombineFit function output.

A non-linear combination transform is a mathematical function that takes one or more input variables and combines them using a non-linear mathematical formula to generate an output variable. The relationship between the input variables and the output variable is not a linear relationship.

In contrast to linear transformations, where the output variable is a linear combination of the input variables, non-linear transformations can capture more complex relationships between variables, such as curves, bends, and interactions. Non-linear transformations are commonly used in machine learning and data analysis to capture non-linear patterns in data.

An example of a non-linear combination transform is the polynomial regression model, which uses a polynomial function to fit a non-linear relationship between the input variables and the output variable. Another example is the sigmoid function, which is often used in neural networks to introduce non-linearities into the model.

### Function Information

- [TD\\_NonLinearCombineTransform Syntax](#)
- [Required Syntax Elements for TD\\_NonLinearCombineTransform](#)
- [Optional Syntax Elements for TD\\_NonLinearCombineTransform](#)
- [TD\\_NonLinearCombineTransform Input](#)
- [TD\\_NonLinearCombineTransform Output](#)
- [Example: How to Use TD\\_NonLinearCombineTransform](#)

## TD\_NonLinearCombineTransform Syntax

```

TD_NonLinearCombineTransform (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS FitTable DIMENSION
    USING

```

```
[ Accumulate ( {'accumulate_column' | 'accumulate_column_range'}[,...] ) ]
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_NonLinearCombineTransform

### ON clause

Accept the Input Table and FitTable clauses.

## Optional Syntax Elements for TD\_NonLinearCombineTransform

### Accumulate

Specify the input table column names to copy to the output table.

## TD\_NonLinearCombineTransform Input

### Input Table Schema

Column	Data Type	Description
TargetColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL /NUMERIC, FLOAT, REAL, DOUBLE PRECISION	The input table column names to use in the non-linear combination.
AccumulateColumns	ANY	The input table column names to copy to the output table.

## TD\_NonLinearCombineTransform Output

### Output Table Schema

Column	Data Type	Description
AccumulateColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL /NUMERIC, FLOAT, REAL, DOUBLE PRECISION	The specified column names in the Accumulate element copied to the output table.

Column	Data Type	Description
ResultColumn	REAL	The values calculated using the specified formula are displayed.

## Example: How to Use TD\_NonLinearCombineTransform

### TD\_NonLinearCombineFit InputTable

```

passenger survived pclass gender age sibsp parch fare      cabin embarked
-----
1      0 General male   22    1    0  7.250000000 null  S
2      1 Deluxe  female 38    1    1  71.280000000 C85  C
3      1 General female 26    0    0  7.930000000 null  S
4      1 Deluxe  female 35    1    0  53.100000000 C123 S
5      0 General male   35    0    1  8.050000000 null  S

```

### TD\_NonLinearCombineFit Output

```

total_cost      sibsp  parch  fare
-----  -----  -----  -----
Y=(X0+X1+1)*X2 null    null    null

```

### TD\_NonLinearCombineTransform SQL Call

```

SELECT * FROM TD_NonLinearCombineTransform (
ON nonLinearCombineFit_input AS InputTable
ON nonLinearCombineFit_output AS FitTable DIMENSION
USING
Accumulate('Passenger')
) as dt order by 1;

```

### TD\_NonLinearCombineTransform Output Table

```

passenger TotalCost
-----
1 14.50000
2 213.84000
3 7.93000
4 106.20000
5 16.10000

```

## TD\_OneHotEncodingFit

TD\_OneHotEncodingFit outputs a table of attributes and categorical values to input to [TD\\_OneHotEncodingTransform](#), which encodes them as one-hot numeric vectors, that is, a vector that

contains 0 or 1, where it is set to 1 if a specific categorical value is true (for example, M in a gender column would be 1, all other values are 0).

TD\_OneHotEncodingFit follows this process:

1. Select a table of attributes and categorical values.
2. Use TD\_OneHotEncodingFit to output the attributes and categorical values to TD\_OneHotEncodingTransform.

## Function Information

- [TD\\_OneHotEncodingFit Syntax](#)
- [Required Syntax Elements for TD\\_OneHotEncodingFit](#)
- [Optional Syntax Elements for TD\\_OneHotEncodingFit](#)
- [TD\\_OneHotEncodingFit Input](#)
- [TD\\_OneHotEncodingFit Output](#)
- [TD\\_OneHotEncodingFit Usage Notes](#)
- [Example: How to Use TD\\_OneHotEncodingFit](#)

## TD\_OneHotEncodingFit Syntax

### TD\_OneHotEncodingFit Syntax for Dense Input

```
TD_OneHotEncodingFit (
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY ]
    [ ON { table | view | (query) } AS CategoryTable Dimension ]
    USING
    IsInputDense ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})
    TargetColumn ({'target_column' | 'target_column_range'}[,,...])
{
    Approach ('LIST')
    {
        CategoricalValues ('category_i'[,...])
    |
        TargetColumnNames ('targetcolnames_column')
        CategoriesColumn ('category_column')
    }
    |
    Approach ('AUTO')
}
[ OtherColumnName ('other_column_name' | 'other_column_name_i', ...)]
[ CategoryCounts (category_count | category_count_i, ...)]
)
```

## **TD\_OneHotEncodingFit Syntax for Sparse Input**

```
TD_OneHotEncodingFit (
  ON { table | view | (query) } AS InputTable PARTITION BY attribute_column
  USING
  IsInputDense ({{'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}})
  TargetAttributes ('attribute_name'[,...])
  [ OtherAttributesNames ('other_attribute_name'[,...]) ]
  AttributeColumn ('attribute_column')
  ValueColumn ('value_column_name')
)
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## **Required Syntax Elements for TD\_OneHotEncodingFit**

### **ON clause**

Accept the InputTable clause.

---

**Note:**

Applicable for dense and sparse input format.

---

### **IsInputDense**

Specify whether the input is dense or sparse.

### **TargetColumn**

[Required with IsInputDense ('true'), disallowed otherwise.] Specify the InputTable categorical columns to be encoded. The maximum number of unique columns in the TargetColumn argument is 2018.

### **CategoryCounts**

[Required with AutoApproach] Specify the category counts for each of the TargetColumns. The number of values in CategoryCounts must be the same as the number of TargetColumns.

**CategoricalValues**

[Required with Approach as LIST and a single target column] Specify the list of categories that need to be encoded in the desired order.

If only one target column is provided, category values are read from this argument. Otherwise, they will be read from the category table. This argument is supported to keep the backward compatibility.

- The number of characters in TargetColumnName plus the number of characters in the category specified in the CategoricalValues argument must be less than 128 characters.
- The maximum number of categories in the CategoricalValues argument is 2018.

**AttributeColumn**

[Required with IsInputDense ('false'), disallowed otherwise.] Specify the name of the InputTable column of attributes.

**ValueColumn**

[Required with IsInputDense ('false'), disallowed otherwise.] Specify the name of the InputTable column of attribute values.

**TargetAttributes**

[Required with IsInputDense ('false'), disallowed otherwise.] Specify one or more attributes to encode in one-hot form. Every *target\_attribute* must be in *attribute\_column*.

**Note:**

The number of characters in values specified in the TargetAttributes argument plus the number of characters in categories specified in the value column must be less than 128 characters.

**Optional Syntax Elements for TD\_OneHotEncodingFit****ON clause**

Accept the CategoryTable clause.

**Note:**

Applicable for dense input format only.

**IsInputDense**

Specify whether the input is dense or sparse.

---

**Note:**

CategoryTable is meant for dense input format and not for sparse format.

---

**TargetColumnNames**

Specify the CategoryTable column which contains the name of the target columns.

**CategoriesColumn**

Specify the CategoryTable column which contains the category values.

**CategoryCounts**

Specify the category counts for each of the TargetColumns. The number of values in CategoryCounts must be the same as the number of TargetColumns.

**Approach**

Specify whether to determine categories automatically from the input table data (AUTO approach) or the user-provided list (LIST approach).

Default: LIST.

**OtherColumnName**

[Optional with IsInputDense ('true'), disallowed otherwise.] Specify the column name for the column representing one-hot encoding for values other than the ones specified in the CategoricalValues argument or CategoryTable or categories found through the Auto approach.

Default: 'other'

- The number of characters in TargetColumnNames plus the number of characters in other category value must be less than 128 characters.
- The maximum number of other category value in the OtherColumnName argument is 2018.

**OtherAttributeNames**

[Optional with IsInputDense ('false'), disallowed otherwise.] For each *target\_attribute*, specify a category name (*other\_attribute*) for attributes that TargetAttributes does not specify. The *n*th *other\_attribute* corresponds to the *n*th *target\_attribute*.

- The number of characters in values specified in the TargetAttributes argument plus the number of characters in values specified in the OtherAttributeNames argument must be less than 128 characters.

- The number of values passed to the TargetAttributes argument and OtherAttributeNames argument must be equal.

## TD\_OneHotEncodingFit Input

### InputTable Schema for Dense Input

Column	Data Type	Description
<i>target_column</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Columns from the InputTable to be encoded.

### InputTable Schema for Sparse Input

Column	Data Type	Description
<i>attribute_column</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Column from the InputTable which contains the attributes.
<i>value_column</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Column from the InputTable which contains the attribute values.

### CategoryTable Schema

Column	Data Type	Description
<i>ColumnName</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Column from the CategoryTable which contains the name of the target columns.
<i>CategoryValue</i>	CHAR or VARCHAR (CHARACTER SET LATIN or UNICODE)	Column from the CategoryTable which contains the category values. The maximum category length is 128 characters.

## TD\_OneHotEncodingFit Output

### Output Table Schema for Dense Input

Column	Data Type	Description
<i>targetColumn_name</i>	INTEGER	Used to identify TargetColumns in the Transform function. Contains TargetColumn names.
<i>&lt;targetColumn&gt;_&lt;cat_value&gt;</i>	VARCHAR (CHARACTER SET UNICODE)	Preserves the column definition for the Transform function. Contains NULL values.

Column	Data Type	Description
<targetColumn>_<other>	VARCHAR (CHARACTER SET UNICODE)	Preserves the column definition for the Transform function. Contains NULL values.

## Output Table Schema for Sparse Input

Column	Data Type	Description
TD_VALUE_TYPE_OHEFIT	INTEGER	1 if row has <i>attribute_column-target_attribute</i> pair. 0 if row has <i>attribute_column-other_attribute</i> pair.
<i>attribute_column</i>	VARCHAR (CHARACTER SET UNICODE)	Preserves attribute column name for TD_OneHotEncodingTransform. Contains only NULL values.
<i>value_column</i>	VARCHAR (CHARACTER SET UNICODE)	Preserves value column name for TD_OneHotEncodingTransform. Contains only NULL values.

## TD\_OneHotEncodingFit Usage Notes

One hot encoding is a technique used to represent categorical data as numerical data. It involves creating a binary vector for each category or level of a categorical variable, with each vector having a length equal to the number of possible categories.

In this technique, a value of 1 is assigned to the corresponding category for a particular observation and a value of 0 is assigned to all other categories. This results in a matrix of 1's and 0's, where each row represents a single observation and each column represents a category.

For example, if we have a categorical variable "Color" with three possible values - red, blue, and green - we would create three binary vectors, one for each color. If an observation is red, then the corresponding binary vector would be [1,0,0]. If it is blue, the vector would be [0,1,0]. And if it is green, the vector would be [0,0,1].

One hot encoding is commonly used in machine learning algorithms, such as logistic regression and neural networks, as these algorithms typically require numerical data as input. It can also help in reducing the potential bias that can result from using ordinal encoding (where values are assigned numerical codes based on their order or rank) or label encoding (where values are assigned numerical codes based on their frequency of occurrence).

For example, we have the following dataset:

Fruit	Categorical Value pf Fruit	Price
apple	1	5
mango	2	10

Fruit	Categorical Value pf Fruit	Price
apple	1	15
orange	3	20

Assuming we want to one hot encode the column Fruit, how do we go about it? We have three different values in this column: apple, mango, and orange. This means that three new columns will be introduced as a result of our one hot encoding. Following is the result of running this encoding on the dataset:

apple	mango	orange	price
1	0	0	5
0	1	0	10
1	0	0	15
0	0	1	20

The column Categorical Value of Fruit is omitted from the result shown. This does not necessarily need to be the case and it is for our ease of understanding only. All rows that had apple in the original dataset now have a 1 in the column apple and a 0 otherwise. Similarly, all rows that had mango now have a 1 in the mango column and 0 otherwise. And so on, for all other fruits in our fruit column.

## Example: How to Use TD\_OneHotEncodingFit

### Note:

Multiple column support for TD\_OrdinalEncoding, TD\_OneHotEncoding, and TD\_Histogram is available in release 17.20.03.07 and later. If you are using an older version, the TargetColumn argument accepts only one column.

### TD\_OneHotEncodingFit Input

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment 0 in the left sidebar.

### TD\_OneHotEncodingFit Dense InputTable: onehot\_titanic\_dataset

Input Data:

Passenger_id	Survived	Pclass	Name	Age	Gender	City	Cabin
5	1	D	Mr. John Doe	27	male	Del	a
4	0	B	Mrs. Jacques Heath	25	female	Hyd	c

Passenger_id	Survived	Pclass	Name	Age	Gender	City	Cabin
6	1	E	Mr. Ben Tennison	22	male	Hyd	b
3	1	C	Mrs. Laina	26	female	Pune	b
1	0	A	Mr. Owen Harris	22	male	Pune	a
2	1	B	Mrs. John Bradley	38	female	Hyd	a

```

DROP TABLE onehot_titanic_dataset;
CREATE SET TABLE onehot_titanic_dataset(
  Passenger_id INTEGER,
  Survived INTEGER,
  Pclass VARCHAR(5) CHARACTER SET LATIN NOT CASESPECIFIC,
  Name VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
  Age INTEGER,
  Gender VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC,
  City VARCHAR(5) CHARACTER SET LATIN NOT CASESPECIFIC,
  Cabin VARCHAR(5) CHARACTER SET LATIN NOT CASESPECIFIC
);
-- Dense Input set.
INSERT INTO onehot_titanic_dataset VALUES (1, 0, 'A', 'Mr. Owen Harris',
22, 'male','Pune','a');
INSERT INTO onehot_titanic_dataset VALUES (2, 1, 'B', 'Mrs. John Bradley',
38, 'female','Hyd','a');
INSERT INTO onehot_titanic_dataset VALUES (3, 1, 'C', 'Mrs. Laina',
26, 'female','Pune','b');
INSERT INTO onehot_titanic_dataset VALUES (4, 0, 'B', 'Mrs. Jacques Heath',
25, 'female','Hyd','c');
INSERT INTO onehot_titanic_dataset VALUES (5, 1, 'D', 'Mr. John Doe',
27, 'male','Del','a');
INSERT INTO onehot_titanic_dataset VALUES (6, 1, 'E', 'Mr. Ben Tennision',
22, 'male','Hyd','b');

```

### TD\_OneHotEncodingFit SQL Call: Dense Input [Auto Approach]

```

SELECT * FROM TD_OneHotEncodingFit(
ON onehot_titanic_dataset AS INPUTTABLE
USING
TargetColumn('Gender','Cabin','City')
OtherColumnName('other')
IsInputDense('true')
CategoryCounts(2,3,3)

```

```
Approach('Auto')
) AS dt;
```

### **TD\_OneHotEncodingFit Output Table: Dense Input [Auto Approach]**

Gender	Gender_0	Gender_1	Gender_other	Cabin	Cabin_0	Cabin_1	Cabin_2	Cabin_other	City	City_0	City_1
	female	male									
					a	b	c				

### **TD\_OneHotEncodingFit CategoryTable: categoryTable**

```
CREATE TABLE categoryTable (column_name VARCHAR(20) CHARACTER SET Latin
NOT CASESPECIFIC,
category VARCHAR(20) CHARACTER SET Latin NOT CASESPECIFIC);

INSERT INTO categoryTable values('Gender','Male');
INSERT INTO categoryTable values('Gender','Female');
INSERT INTO categoryTable values('Cabin','a');
INSERT INTO categoryTable values('Cabin','b');
INSERT INTO categoryTable values('Cabin','c');
INSERT INTO categoryTable values('City','Del');
INSERT INTO categoryTable values('City','Hyd');
```

### **TD\_OneHotEncodingFit SQL Call: Dense Input [List Approach]**

```
SELECT * FROM TD_OneHotEncodingFit(
ON onehot_titanic_dataset AS INPUTTABLE
ON categoryTable AS categoryTable Dimension
USING
TargetColumn('Gender','Cabin','City')
CategoryCounts(2,3,2)
TargetColumnNames ('column_name')
CategoriesColumn ('category')
OtherColumnName('other')
IsInputDense('true')
Approach('List')
) AS dt ;
```

**TD\_OneHotEncodingFit Output Table: Dense Input [List Approach]**

Gender	Gender_0	Gender_1	Gender_other	Cabin	Cabin_0	Cabin_1	Cabin_2	Cabin_other	City	City_0	City_1
	female	male									
					a	b	c				

**TD\_OneHotEncodingFit Sparse InputTable: onehot\_sparse\_input**

```
DROP TABLE onehot_sparse_input;

CREATE TABLE onehot_sparse_input (id INTEGER, attribute_column VARCHAR(20),
value_column VARCHAR(20));

INSERT INTO onehot_sparse_input VALUES (1, 'Survived', 0);
INSERT INTO onehot_sparse_input VALUES (2, 'Survived', 1);
INSERT INTO onehot_sparse_input VALUES (3, 'Survived', 1);
INSERT INTO onehot_sparse_input VALUES (1, 'Pclass', 'A');
INSERT INTO onehot_sparse_input VALUES (2, 'Pclass', 'B');
INSERT INTO onehot_sparse_input VALUES (3, 'Pclass', 'C');
INSERT INTO onehot_sparse_input VALUES (1, 'Name', 'Mr. Owen Harris');
INSERT INTO onehot_sparse_input VALUES (2, 'Name', 'Mrs. John Bradley');
INSERT INTO onehot_sparse_input VALUES (3, 'Name', 'Mrs. Laina');
INSERT INTO onehot_sparse_input VALUES (1, 'Age', 22);
INSERT INTO onehot_sparse_input VALUES (2, 'Age', 38);
INSERT INTO onehot_sparse_input VALUES (3, 'Age', 26);
INSERT INTO onehot_sparse_input VALUES (1, 'Gender', 'male');
INSERT INTO onehot_sparse_input VALUES (2, 'Gender', 'female');
INSERT INTO onehot_sparse_input VALUES (3, 'Gender', 'female');
INSERT INTO onehot_sparse_input VALUES (1, 'City', 'Del');
INSERT INTO onehot_sparse_input VALUES (2, 'City', 'Hyd');
INSERT INTO onehot_sparse_input VALUES (3, 'City', 'Pune');
INSERT INTO onehot_sparse_input VALUES (1, 'Cabin', 'a');
INSERT INTO onehot_sparse_input VALUES (2, 'Cabin', 'a');
INSERT INTO onehot_sparse_input VALUES (3, 'Cabin', 'b');
```

**TD\_OneHotEncodingFit SQL Call: Sparse Input**

```
SELECT * FROM TD_OneHotEncodingFit(
ON onehot_sparse_input AS InputTable PARTITION BY attribute_column
USING
```

```

IsInputDense ('false')
TargetAttributes ('Gender','Cabin','City')
AttributeColumn ('attribute_column')
ValueColumn ('value_column')
) AS dt ORDER BY 1,2;

```

### **TD\_OneHotEncodingFit Output Table: Sparse Input**

attribute_column	value_column	TD_VALUE_TYPE_OHEFIT
Cabin	a	0
Cabin	v	0
City	Del	0
City	Hyd	0
City	Pune	0
Gender	female	0
Gender	male	0

## **TD\_OneHotEncodingTransform**

TD\_OneHotEncodingTransform encodes specified attributes and categorical values as one-hot numeric vectors, using [TD\\_OneHotEncodingFit](#) output.

This function takes the fitted OneHotEncodingFit object and uses it to transform the input data into a one hot encoded representation. See [TD\\_OneHotEncodingFit Usage Notes](#).

TD\_OneHotEncodingTransform follows this process:

1. Select the table and column to be encoded by the TD\_OneHotEncodingFit function.
2. Use TD\_OneHotEncodingTransform to encode the specific attributes and categorical values as one-hot numeric vectors, using TD\_OneHotEncodingTransform output.

### **Function Information**

- [TD\\_OneHotEncodingTransform Syntax](#)
- [Required Syntax Elements for TD\\_OneHotEncodingTransform](#)
- [Optional Syntax Elements for TD\\_OneHotEncodingTransform](#)
- [TD\\_OneHotEncodingTransform Input](#)
- [TD\\_OneHotEncodingTransform Output](#)
- [Example: How to Use TD\\_OneHotEncodingTransform](#)

## TD\_OneHotEncodingTransform Syntax

### TD\_OneHotEncodingTransform Syntax for Dense Input

```
TD_OneHotEncodingTransform (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    ON { table | view | (query) } AS FitTable DIMENSION
    USING
    IsInputDense ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})
)
```

### TD\_OneHotEncodingTransform Syntax for Sparse Input

```
TD_OneHotEncodingTransform (
    ON { table | view | (query) } AS InputTable PARTITION BY attribute_column
    ON { table | view | (query) } AS FitTable PARTITION BY attribute_column
    USING
    IsInputDense ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})
)
```

---

#### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_OneHotEncodingTransform

### ON clause

Accepts the InputTable and FitTable ON clauses.

### IsInputDense

Specifies whether the input is dense or sparse.

## Optional Syntax Elements for TD\_OneHotEncodingTransform

All syntax elements are required.

## TD\_OneHotEncodingTransform Input

### InputTable Schema

See [TD\\_OneHotEncodingFit Input](#).

### FitTable Schema

See [TD\\_OneHotEncodingFit Output](#).

## TD\_OneHotEncodingTransform Output

### Output Table Schema for Dense Input

Column	Data Type	Description
<i>input_table_column</i>	Same as in InputTable	Column copied from InputTable.
<i>targetColumns_values</i>	INTEGER	One-hot-encoded representation of categorical values for each target column.

### Output Table Schema for Sparse Input

Column	Data Type	Description
<i>input_table_column</i>	Same as in InputTable	Column copied from InputTable.
<i>attribute_column</i>	CHAR or VARCHAR (CHARACTER SET UNICODE)	Column which contains the attribute names.
<i>value_column</i>	CHAR or VARCHAR (CHARACTER SET UNICODE)	Column which contains the one-hot-encoded attribute values.

## Example: How to Use TD\_OneHotEncodingTransform

### Note:

Multiple column support for TD\_OrdinalEncoding, TD\_OneHotEncoding, and TD\_Histogram is available in release 17.20.03.07 and later. If you are using an older version, the TargetColumn argument accepts only one column.

### Dense InputTable: onehot\_titanic\_dataset

Passenger_id	Survived	Pclass	Name	Age	Gender	City	Cabin
5	1	D	Mr. John Doe	27	male	Del	a

Passenger_id	Survived	Pclass	Name	Age	Gender	City	Cabin
4	0	B	Mrs. Jacques Heath	25	female	Hyd	c
6	1	E	Mr. Ben Tennison	22	male	Hyd	b
3	1	C	Mrs. Laina	26	female	Pune	b
1	0	A	Mr. Owen Harris	22	male	Pune	a
2	1	B	Mrs. John Bradley	38	female	Hyd	a

**FitTable: onehot\_titanic\_fit\_output**

```
CREATE TABLE onehot_titanic_fit_output as (
SELECT * FROM TD_OneHotEncodingFit(
ON onehot_titanic_dataset AS INPUTTABLE
USING
TargetColumn('Gender','Cabin','City')
OtherColumnName('other')
IsInputDense('true')
CategoryCounts(2,3,3)
Approach('Auto')
) AS dt
)with data;
```

**TD\_OneHotEncodingTransform Output Table: FitTable**

Gender	Gender_0	Gender_1	Gender_other	Cabin	Cabin_0	Cabin_1	Cabin_2	Cabin_other	City	City_0	City_1
	female	male									
					a	b	c				
											Del Hyd

**TD\_OneHotEncodingTransform SQL Call: Dense Input**

```
SELECT * FROM TD_OneHotEncodingTransform (
ON onehot_titanic_dataset AS InputTable
ON onehot_titanic_fit_output AS FitTable Dimension
USING
IsInputDense('True')
) AS dt ORDER BY 1;
```

**TD\_OneHotEncodingTransform Output Table: Dense Input**

Passenger_id	Survived	Pclass	Name	Age	Gender_0	Gender_1	Gender_other	City_0	City_1	City_2
1	0	A	Mr. Owen Harris	22	0	1	0	0	0	1
2	1	B	Mrs. John Bradley	38	1	0	0	0	1	0
3	1	C	Mrs. Laina	26	1	0	0	0	0	1
4	0	B	Mrs. Jacques Heath	25	1	0	0	0	1	0
5	1	D	Mr. John Doe	27	0	1	0	1	0	0
6	1	E	Mr. Ben Tennison	22	0	1	0	0	1	0

**Sparse InputTable: onehot\_sparse\_input**

```

DROP TABLE onehot_sparse_input;
CREATE TABLE onehot_sparse_input (id INTEGER, attribute_column VARCHAR(20),
value_column VARCHAR(20));
INSERT INTO onehot_sparse_input VALUES (1, 'Survived', 0);
INSERT INTO onehot_sparse_input VALUES (2, 'Survived', 1);
INSERT INTO onehot_sparse_input VALUES (3, 'Survived', 1);
INSERT INTO onehot_sparse_input VALUES (1, 'Pclass', 'A');
INSERT INTO onehot_sparse_input VALUES (2, 'Pclass', 'B');
INSERT INTO onehot_sparse_input VALUES (3, 'Pclass', 'C');
INSERT INTO onehot_sparse_input VALUES (1, 'Name', 'Mr. Owen Harris');
INSERT INTO onehot_sparse_input VALUES (2, 'Name', 'Mrs. John Bradley');
INSERT INTO onehot_sparse_input VALUES (3, 'Name', 'Mrs. Laina');
INSERT INTO onehot_sparse_input VALUES (1, 'Age', 22);
INSERT INTO onehot_sparse_input VALUES (2, 'Age', 38);
INSERT INTO onehot_sparse_input VALUES (3, 'Age', 26);
INSERT INTO onehot_sparse_input VALUES (1, 'Gender', 'male');
INSERT INTO onehot_sparse_input VALUES (2, 'Gender', 'female');
INSERT INTO onehot_sparse_input VALUES (3, 'Gender', 'female');
INSERT INTO onehot_sparse_input VALUES (1, 'City', 'Del');
INSERT INTO onehot_sparse_input VALUES (2, 'City', 'Hyd');
INSERT INTO onehot_sparse_input VALUES (3, 'City', 'Pune');

```

```
INSERT INTO onehot_sparse_input VALUES (1, 'Cabin', 'a');
INSERT INTO onehot_sparse_input VALUES (2, 'Cabin', 'a');
INSERT INTO onehot_sparse_input VALUES (3, 'Cabin', 'b');
```

### **Input data: onehot\_sparse\_input**

<b>id</b>	<b>attribute_column</b>	<b>value_column</b>
3	Survived	1
3	Pclass	C
3	Name	Mrs. Laina
3	Age	26
3	Gender	female
3	City	Pune
3	Cabin	b
1	Survived	0
1	Pclass	A
1	Name	Mr. Owen Harris
1	Age	22
1	Gender	male
1	City	Del
1	Cabin	a
2	Survived	1
2	Pclass	B
2	Name	Mrs. John Bradley
2	Age	38
2	Gender	female
2	City	Hyd
2	Cabin	a

### **FitTable: onehot\_sparse\_fit**

```
CREATE TABLE onehot_sparse_fit AS (
SELECT * FROM TD_OneHotEncodingFit (
    ON onehot_sparse_input AS InputTable PARTITION BY attribute_column
```

```

USING
IsInputDense ('false')
TargetAttributes ('Gender','Cabin','City')
AttributeColumn ('attribute_column')
ValueColumn ('value_column')
) AS dt ) WITH DATA;

```

### **TD\_OneHotEncodingTransform FitTable Output Table: onehot\_sparse\_fit**

attribute_column	value_column	TD_VALUE_TYPE_OHEFIT
City	Pune	0
Gender	male	0
Cabin	a	0
City	Del	0
Gender	female	0
Cabin	b	0
City	Hyd	0

### **TD\_OneHotEncodingTransform SQL Call: Sparse Input**

```

SELECT * FROM TD_OneHotEncodingTransform (
ON onehot_sparse_input AS InputTable PARTITION BY attribute_column
ON onehot_sparse_fit AS FitTable PARTITION BY attribute_column
USING
IsInputDense ('false')
) AS dt ORDER BY 1,2;

```

### **TD\_OneHotEncodingTransform Output Table: Sparse Input**

id	attribute_column	value_column
1	Age	22
1	Cabin_a	1
1	Cabin_b	0
1	Cabin_other	0
1	City_Del	1
1	City_Hyd	0
1	City_other	0

<b>id</b>	<b>attribute_column</b>	<b>value_column</b>
1	City_Pune	0
1	Name	Mr. Owen Harris
1	Pclass	A
1	Gender_female	0
1	Gender_male	1
1	Gender_other	0
1	Survived	0
2	Age	38
2	Cabin_a	1
2	Cabin_b	0
2	Cabin_other	0
2	City_Del	0
2	City_Hyd	1
2	City_other	0
2	City_Pune	0
2	Name	Mrs. John Bradley
2	Pclass	B
2	Gender_female	1
2	Gender_male	0
2	Gender_other	0
2	Survived	1
3	Age	26
3	Cabin_a	0
3	Cabin_b	1
3	Cabin_other	0
3	City_Del	0
3	City_Hyd	0
3	City_other	0
3	City_Pune	1

<b>id</b>	<b>attribute_column</b>	<b>value_column</b>
3	Name	Mrs. Laina
3	Pclass	C
3	Gender_female	1
3	Gender_male	0
3	Gender_other	0
3	Survived	1

## TD\_OrdinalEncodingFit

The TD\_OrdinalEncodingFit function identifies distinct categorical values from an input table or a user-defined list and returns the distinct categorical values along with the ordinal value for each category.

TD\_OrdinalEncodingFit is useful in use cases where categorical data needs to be converted into numerical data for analysis or machine learning algorithms. For example, in a dataset with categorical variables such as "color" or "size", TD\_OrdinalEncodingFit can assign numerical values to each category, making it possible to perform mathematical operations on the data. This helps in tasks such as clustering, classification, and regression analysis.

---

### Note:

- The maximum number of unique categories in a particular column is 4000.
  - The maximum category length is 128 characters.
  - NULL categories are not encoded.
- 

### Function Information

- [TD\\_OrdinalEncodingFit Syntax](#)
- [Required Syntax Elements for TD\\_OrdinalEncodingFit](#)
- [Optional Syntax Elements for TD\\_OrdinalEncodingFit](#)
- [TD\\_OrdinalEncodingFit Input](#)
- [TD\\_OrdinalEncodingFit Output](#)
- [Example: How to Use TD\\_OrdinalEncodingFit](#)

## TD\_OrdinalEncodingFit Syntax

```
TD_ORDINALENCODINGFIT (
  ON { table | view | (query) } AS InputTable
  [ ON { table | view | (query) } AS CategoryTable Dimension ]
```

```
[ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table_name) ]
USING
TargetColumn ({'target_column' | 'target_column_range'},...])
[{
    Approach ('LIST')
    {
        Categories ('category'),...
        [ OrdinalValues (ordinal_value,...)]
    }
    |
    TargetColumnNames ('targetcolnames_column')
    CategoriesColumn ('category_column')
    [ OrdinalValuesColumn ('ordinalvalue_column') ]
}
]
|
Approach ('AUTO')
}]
[ StartValue ({start_value | start_value_i, ... })
[ DefaultValue ({default_value | default_value_i, ... })
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_OrdinalEncodingFit

### ON clause

Accepts the InputTable clause.

### TargetColumn

Specifies the InputTable categorical columns to be encoded.

**Note:**

The maximum number of unique columns in the TargetColumn argument is 2018.

**TargetColumnNames**

[When using CategoryTable] Specifies the CategoryTable column which contains the names of the target columns.

**CategoriesColumn**

[When using CategoryTable] Specifies the CategoryTable column which contains the category values.

**Categories**

[When using the LIST approach and a single target column] Specifies the list of categories for encoding in the required order.

This argument reads the category values, when you only provide one target column. Otherwise, they are read from the CategoryTable.

This argument supports backward compatibility.

**Note:**

The maximum length supported for the categorical value is 128 characters. The maximum number of categories that you can specify in the Categories argument is 4000.

## Optional Syntax Elements for TD\_OrdinalEncodingFit

**ON clause**

Accepts the CategoryTable clause.

**OUT clause**

Accepts the OutputTable clause.

**OrdinalValuesColumn**

[When using CategoryTable] Specifies the CategoryTable column which contains the ordinal values.

If omitted, ordinal values are generated based on the StartValue argument.

**Approach**

Specifies AUTO to obtain categories from the input table or specify LIST to obtain categories from the user.

Default value: AUTO.

## OrdinalValues

[When using the LIST approach and a single target column] Specifies the custom ordinal values to replace the categories when you use the LIST approach for encoding the categorical values.

If you do not provide the ordinal value and the start value, then by default, the first category contains the default start value 0, and the last category is assigned a value that is one lesser than the total number of categories.

For example, if there are three categories, then the categories contain the values 0, 1, 2 respectively.

However, if you only specify the ordinal values, then each ordinal value is associated with a categorical value. For example, if there are three categories and the ordinal values are 3, 4, 5 then the ordinal values are assigned to the respective categories.

The TD\_OrdinalEncodingFit function returns an error when the ordinal value count does not match the categorical value count or if both the ordinal values and the start value are provided.

### Note:

You can either use the OrdinalValues or the StartValue argument in the syntax.

If only one target column is provided, ordinal values are read from this argument. Otherwise, they are read from the CategoryTable.

If omitted, ordinal values are generated based on the StartValue argument.

This argument is supported to keep the backward compatibility.

Default value: 0 to NumOfCategories -1.

## StartValue

Specifies the starting value for the ordinal values list.

If only one value is specified, the value is applied to all the target columns. Otherwise, the number of start values must be equal to the number of target columns.

Default value: 0.

## DefaultValue

Specifies the ordinal value to use when the categorical value is not found during transform.

The TD\_OrdinalEncodingFit function adds the TD\_OTHER\_CATEGORY row and assigns the specified value to the row in the FIT table output.

If you specify the default value in the TD\_OrdinalEncodingFit function and if the TD\_OrdinalEncodingTransform function does not find the categorical value in the FIT table, then the function assigns the TD\_OTHER\_CATEGORY value to the missing category.

If you do not specify the default value in the TD\_OrdinalEncodingFit function and if the TD\_OrdinalEncodingTransform function does not find the categorical value in the FIT table, then the TD\_OrdinalEncodingTransform function returns an error.

If only one value is specified, the value is applied to all the target columns. Otherwise, the number of default values must be equal to the number of target columns.

Default value: 0.

## TD\_OrdinalEncodingFit Input

### InputTable Schema

Column	Data Type	Description
TargetColumn	CHAR or VARCHAR CHARACTER SET LATIN/UNICODE	Columns to be encoded.

### CategoryTable Schema

Column	Data Type	Description
ColumnName	CHAR or VARCHAR CHARACTER SET LATIN/UNICODE	Column which contains the names of the target columns.
CategoryValue	CHAR or VARCHAR CHARACTER SET LATIN/UNICODE	Column which contains the category values.
OrdinalValue	BYTEINT, SMALLINT, INTEGER	Column which contains the ordinal values.

## TD\_OrdinalEncodingFit Output

### Output Table Schema

Column	Data Type	Description
TD_ColumnName_ORDFIT	VARCHAR CHARACTER SET UNICODE	Column which contains the names of the target columns.
TD_Category_ORDFIT	VARCHAR CHARACTER SET UNICODE	Column which contains the category values along with TD_CategoryCount and TD_OtherCategory for each of the target columns.
TD_Value_ORDFIT	INTEGER	Column which contains the ordinal values for each category and for each target column.
TD_Index_ORDFIT	SMALLINT	Column which contains the index of the target columns.

Column	Data Type	Description
Target_column_1..... Target_column_N	Any	Column names copied to fit output, needed in the Transform function.

## Example: How to Use TD\_OrdinalEncodingFit

**TD\_OrdinalEncodingFit InputTable:** OrdEnc\_titanic\_train

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
873	0	1	Carlsson; Mr. Frans Olof	male	33	0	0	695	5.0000	B51 B53 B55	S
631	1	1	Barksworth; Mr. Algernon Henry Wilson	male	80	0	0	27042	30.0000	A23	S
97	0	1	Goldschmidt; Mr. George B	male	71	0	0	PC 17754	34.6542	A5	C
1000	0	1			71	0	0		34.6542		
488	0	1	Kent; Mr. Edward Austin	male	58	0	0	11771	29.7000	B37	C
505	1	1	Maiioni; Miss. Roberta	female	16	0	0	110152	86.5000	B79	S

**Example: TD\_OrdinalEncodingFit SQL Call Using Auto Approach**

```
SELECT * FROM TD_OrdinalEncodingFit (
ON ordinal_titanic_dataset AS InputTable
USING
TargetColumn('name','gender','ticket','cabin','embarked')
Approach ('AUTO')
StartValue (5, 10, 15, 0, -5)
DefaultValue (-1, -10, -15, 20, 0)
) AS dt ORDER BY 1,3;
```

**TD\_OrdinalEncodingFit Output Table Using Auto Approach**

TD_ColumnName_ORDFIT	TD_Category_ORDFIT	TD_Value_ORDFIT	TD_Index_ORDFIT	name	gender	ticket	cabin	embarked
cabin		0	3	None	None	None	None	None
cabin	A23	1	3	None	None	None	None	None
cabin	A5	2	3	None	None	None	None	None
cabin	B37	3	3	None	None	None	None	None
cabin	B51 B53 B55	4	3	None	None	None	None	None
cabin	B79	5	3	None	None	None	None	None
cabin	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
cabin	TD_OTHER_CATEGORY	20	-2	None	None	None	None	None
embarked		-5	4	None	None	None	None	None
embarked	C	-4	4	None	None	None	None	None
embarked	S	-3	4	None	None	None	None	None
embarked	TD_OTHER_CATEGORY	0	-2	None	None	None	None	None
embarked	TD_CATEGORY_COUNT	3	-1	None	None	None	None	None
name	TD_OTHER_CATEGORY	-1	-2	None	None	None	None	None
name		5	0	None	None	None	None	None
name	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
name	Barksworth; Mr. Algernon Henry Wilson	6	0	None	None	None	None	None
name	Carlsson; Mr. Frans Olof	7	0	None	None	None	None	None
name	Goldschmidt; Mr. George B	8	0	None	None	None	None	None

TD_ColumnName_ORDFIT	TD_Category_ORDFIT	TD_Value_ORDFIT	TD_Index_ORDFIT	name	gender	ticket	cabin	embarked
name	Kent; Mr. Edward Austin	9	0	None	None	None	None	None
name	Maioni; Miss. Roberta	10	0	None	None	None	None	None
gender	TD_OTHER_CATEGORY	-10	-2	None	None	None	None	None
gender	TD_CATEGORY_COUNT	3	-1	None	None	None	None	None
gender		10	1	None	None	None	None	None
gender	female	11	1	None	None	None	None	None
gender	male	12	1	None	None	None	None	None
ticket	TD_OTHER_CATEGORY	-15	-2	None	None	None	None	None
ticket	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
ticket		15	2	None	None	None	None	None
ticket	110152	16	2	None	None	None	None	None
ticket	11771	17	2	None	None	None	None	None
ticket	27042	18	2	None	None	None	None	None
ticket	695	19	2	None	None	None	None	None
ticket	PC 17754	20	2	None	None	None	None	None

**Note:**

Multiple column support for TD\_OrdinalEncoding, TD\_OneHotEncoding, and TD\_Histogram is available in release 17.20.03.07 and later. If you are using an older version, the TargetColumn argument accepts only one column.

**CategoryTable: catTable**

ColumnName	Category	ordinalValue
ticket	695	0
gender	female	0
cabin	B79	0
ticket	11771	1
gender	male	1
cabin	B51 B53 B55	1
ticket	PC 17754	2
gender		2
cabin	A5	2
ticket	27042	3
name	Maioni; Miss.Roberta	0
cabin	A23	3
ticket	110152	4
name	Carlsson; Mr. Frans Olof	1
cabin	B37	4
ticket		5
name	Goldschmidt; Mr. George B	2
cabin		5
name	Barksworth; Mr. Algernon Henry Wilson	3
embarked	S	0
name	Kent; Mr. Edward Austin	4
embarked	C	1
name		5

ColumnName	Category	ordinalValue
embarked		2

### Example: TD\_OrdinalEncodingFit SQL Call Using List Approach

```
SELECT * FROM TD_OrdinalEncodingFit (
ON ordinal_titanic_dataset AS InputTable
ON catTable AS CategoryTable Dimension
USING
TargetColumn('name','gender','ticket','cabin','embarked')
Approach ('LIST')
TargetColumnNames('ColumnName')
CategoriesColumn('Category')
OrdinalValuesColumn ('ordinalValue')
DefaultValue (-1, -10, -15, 20, 0)
) AS dt ORDER BY 1,3;
```

### TD\_OrdinalEncodingFit Output Table Using List Approach

TD_ColumnName_ORDFIT	TD_Category_ORDFIT	TD_Value_ORDFIT	TD_Index_ORDFIT	name	gender	ticket	cabin	embarked
cabin	B79	0	3	None	None	None	None	None
cabin	B51 B53 B55	1	3	None	None	None	None	None
cabin	A5	2	3	None	None	None	None	None
cabin	A23	3	3	None	None	None	None	None
cabin	B37	4	3	None	None	None	None	None
cabin		5	3	None	None	None	None	None
cabin	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
cabin	TD_OTHER_CATEGORY	20	-2	None	None	None	None	None
embarked	TD_OTHER_CATEGORY	0	-2	None	None	None	None	None
embarked	S	0	4	None	None	None	None	None
embarked	C	1	4	None	None	None	None	None
embarked		2	4	None	None	None	None	None
embarked	TD_CATEGORY_COUNT	3	-1	None	None	None	None	None
name	TD_OTHER_CATEGORY	-1	-2	None	None	None	None	None
name	Maioni; Miss. Roberta	0	0	None	None	None	None	None
name	Carlsson; Mr. Frans Olof	1	0	None	None	None	None	None
name	Goldschmidt; Mr. George B	2	0	None	None	None	None	None
name	Barksworth; Mr. Algernon Henry Wilson	3	0	None	None	None	None	None
name	Kent; Mr. Edward Austin	4	0	None	None	None	None	None

TD_ColumnName_ORDFIT	TD_Category_ORDFIT	TD_Value_ORDFIT	TD_Index_ORDFIT	name	gender	ticket	cabin	embarked
name		5	0	None	None	None	None	None
name	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
gender	TD_OTHER_CATEGORY	-10	-2	None	None	None	None	None
gender	female	0	1	None	None	None	None	None
gender	male	1	1	None	None	None	None	None
gender		2	1	None	None	None	None	None
gender	TD_CATEGORY_COUNT	3	-1	None	None	None	None	None
ticket	TD_OTHER_CATEGORY	-15	-2	None	None	None	None	None
ticket	695	0	2	None	None	None	None	None
ticket	11771	1	2	None	None	None	None	None
ticket	PC 17754	2	2	None	None	None	None	None
ticket	27042	3	2	None	None	None	None	None
ticket	110152	4	2	None	None	None	None	None
ticket		5	2	None	None	None	None	None
ticket	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None

## TD\_OrdinalEncodingTransform

The TD\_OrdinalEncodingTransform function maps the categorical value to a specified ordinal value using the TD\_OrdinalEncodingFit output.

TD\_OrdinalEncodingTransform follows this process:

1. Select the table and columns to be encoded by the TD\_OrdinalEncodingFit function.  
See [TD\\_OrdinalEncodingFit](#).
2. Use TD\_OrdinalEncodingTransform to map each category value to a specified ordinal value, using TD\_OrdinalEncodingTransform output.

### Function Information

- [TD\\_OrdinalEncodingTransform Syntax](#)
- [Required Syntax Elements for TD\\_OrdinalEncodingTransform](#)
- [TD\\_OrdinalEncodingTransform Input](#)
- [TD\\_OrdinalEncodingTransform Output](#)
- [Optional Syntax Elements for TD\\_OrdinalEncodingTransform](#)
- [Example: How to Use TD\\_OrdinalEncodingTransform](#)

## TD\_OrdinalEncodingTransform Syntax

```
TD_OrdinalEncodingTransform (
    ON { table | view | (query) } as InputTable
    ON { table | view | (query) } as FitTable DIMENSION
    USING
    [ Accumulate ({'accumulate_column' | 'accumulate_column_range'}[,...]) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

---

## Required Syntax Elements for TD\_OrdinalEncodingTransform

### ON clause

Accepts the the InputTable and FitTable clauses.

## Optional Syntax Elements for TD\_OrdinalEncodingTransform

### Accumulate

Specifies the InputTable column names to copy to the Output table.

## TD\_OrdinalEncodingTransform Input

### InputTable Schema

Column	Data Type	Description
target_column	CHAR or VARCHAR CHARACTER SET LATIN/UNICODE	Columns to be encoded.

### FitTable Schema

Column	Data Type	Description
TD_ColumnName_ORDFIT	VARCHAR CHARACTER SET UNICODE	Column which contains the names of the target columns.
TD_Category_ORDFIT	VARCHAR CHARACTER SET UNICODE	Column which contains the category values along with TD_CategoryCount and TD_OtherCategory for each of the target columns.
TD_Value_ORDFIT	INTEGER	Column which contains the ordinal values for each category and for each target column.
TD_Index_ORDFIT	SMALLINT	Column which contains the index of the target columns.
Target_column_1..... Target_column_N	Any	Column names copied to fit output, needed in the Transform function.

## TD\_OrdinalEncodingTransform Output

### Output Table Schema

Column	Data Type	Description
Accumulate_column	Any	Column copied from the InputTable to the Output table.
target_column	INTEGER	Target columns with encoded ordinal values.

## Example: How to Use TD\_OrdinalEncodingTransform

### TD\_OrdinalEncodingFit InputTable: OrdEnc\_titanic\_train

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
873	0	1	Carlsson; Mr. Frans Olof	male	33	0	0	695	5.0000	B51 B53 B55	S
631	1	1	Barksworth; Mr. Algernon Henry Wilson	male	80	0	0	27042	30.0000	A23	S
97	0	1	Goldschmidt; Mr. George B	male	71	0	0	PC 17754	34.6542	A5	C
1000	0	1			71	0	0		34.6542		
488	0	1	Kent; Mr. Edward Austin	male	58	0	0	11771	29.7000	B37	C
505	1	1	Maiioni; Miss. Roberta	female	16	0	0	110152	86.5000	B79	S

**Example: TD\_OrdinalEncodingFit SQL Call Using Auto Approach**

```
SELECT * FROM TD_OrdinalEncodingFit (
    ON ordinal_titanic_dataset AS InputTable
    OUT PERMANENT TABLE OutputTable (ordinal_titanic_fit_output)
    USING
        TargetColumn('name','gender','ticket','cabin','embarked')
        Approach ('AUTO')
        StartValue (5, 10, 15, 0, -5)
        DefaultValue (-1, -10, -15, 20, 0)
) AS dt ORDER BY 1,3;
```

**Note:**

Multiple column support for TD\_OrdinalEncoding, TD\_OneHotEncoding, and TD\_Histogram is available in release 17.20.03.07 and later. If you are using an older version, the TargetColumn argument accepts only one column.

**TD\_OrdinalEncodingTransform Output Table Using Auto Approach**

TD_ColumnName_ORDFIT	TD_Category_ORDFIT	TD_Value_ORDFIT	TD_Index_ORDFIT	name	gender	ticket	cabin	embarked
cabin		0	3	None	None	None	None	None
cabin	A23	1	3	None	None	None	None	None
cabin	A5	2	3	None	None	None	None	None
cabin	B37	3	3	None	None	None	None	None
cabin	B51 B53 B55	4	3	None	None	None	None	None
cabin	B79	5	3	None	None	None	None	None
cabin	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
cabin	TD_OTHER_CATEGORY	20	-2	None	None	None	None	None
embarked		-5	4	None	None	None	None	None
embarked	C	-4	4	None	None	None	None	None
embarked	S	-3	4	None	None	None	None	None
embarked	TD_OTHER_CATEGORY	0	-2	None	None	None	None	None
embarked	TD_CATEGORY_COUNT	3	-1	None	None	None	None	None
name	TD_OTHER_CATEGORY	-1	-2	None	None	None	None	None
name		5	0	None	None	None	None	None
name	Barksworth; Mr. Algernon Henry Wilson	6	0	None	None	None	None	None
name	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
name	Carlsson; Mr. Frans Olof	7	0	None	None	None	None	None
name	Goldschmidt; Mr. George B	8	0	None	None	None	None	None

TD_ColumnName_ORDFIT	TD_Category_ORDFIT	TD_Value_ORDFIT	TD_Index_ORDFIT	name	gender	ticket	cabin	embarked
name	Kent; Mr. Edward Austin	9	0	None	None	None	None	None
name	Maioni; Miss. Roberta	10	0	None	None	None	None	None
gender	TD_OTHER_CATEGORY	-10	-2	None	None	None	None	None
gender	TD_CATEGORY_COUNT	3	-1	None	None	None	None	None
gender		10	1	None	None	None	None	None
gender	female	11	1	None	None	None	None	None
gender	male	12	1	None	None	None	None	None
ticket	TD_OTHER_CATEGORY	-15	-2	None	None	None	None	None
ticket	TD_CATEGORY_COUNT	6	-1	None	None	None	None	None
ticket		15	2	None	None	None	None	None
ticket	110152	16	2	None	None	None	None	None
ticket	11771	17	2	None	None	None	None	None
ticket	27042	18	2	None	None	None	None	None
ticket	695	19	2	None	None	None	None	None
ticket	PC 17754	20	2	None	None	None	None	None

### **Example: TD\_OrdinalEncodingTransform SQL Call Using Output from TD\_OrdinalEncodingFit**

```
SELECT * FROM TD_OrdinalEncodingTransform (
    ON ordinal_titanic_dataset AS InputTable
    ON ordinal_titanic_fit_output as FitTable Dimension
    USING
    Accumulate ('passenger')
) AS dt ORDER BY 1;
```

### **TD\_OrdinalEncodingTransform Output**

passenger	name	gender	ticket	cabin	embarked
97	8	12	20	2	-4
488	9	12	17	3	-4
505	10	11	16	5	-3
631	6	12	18	1	-3
873	7	12	19	4	-3
1000	5	10	15	0	-5

## **TD\_Pivoting**

TD\_Pivoting function is used to pivot the data, that is, changes the data from sparse to dense format.

TD\_Unpivoting follows this process:

1. Select a table with data in sparse format.
2. Use TD\_Pivoting to change data from sparse to dense format.

Therefore, you can convert data from sparse to dense format and vice versa. To do this you can use the following functions:

1. TD\_Unpivoting.
2. TD\_Pivoting.

---

#### **Note:**

TD\_Pivoting function is aimed to be used in conjunction with other inDB analytic functions, in contrast to TD\_Pivot.

---

### **Function Information**

- [TD\\_Pivoting Syntax](#)
- [Required Syntax Elements for TD\\_Pivoting](#)

- [Optional Syntax Elements for TD\\_Pivoting](#)
- [TD\\_Pivoting Input](#)
- [TD\\_Pivoting Output](#)
- [Example: How to Use TD\\_Pivoting](#)

## TD\_Pivoting Syntax

```
TD_Pivoting (
    ON { table | view | (query) } AS InputTable PARTITION BY partition_column [, ...]
    [ ORDER BY order_column [, ...] ]
    USING
        PartitionColumns({'partition_column' | 'partition_column_range'} [, ...])
        TargetColumns({'target_column' | 'target_column_range'} [, ...])
        [ Accumulate({'accumulate_column' | 'accumulate_column_range'} [, ...]) ]
    {
        [ RowsPerPartition (rows_per_partition) ]
    |
        [ PivotColumn('pivot_column') ]
        [ PivotKeys('pivot_key' [, ...]) ]
        [ PivotKeysAlias('pivot_key_alias' [, ...]) ]
        [ DefaultPivotValues ('default_pivot_value' [, ...]) ]
    }
    [ Aggregation({'{CONCAT|UNIQUE_CONCAT|SUM|MIN|MAX|AVG}' |
                    'ColumnName:{CONCAT|UNIQUE_CONCAT|SUM|MIN|MAX|AVG}' [, ...]}) ]
    [ Delimiters('single_char' | 'ColumnName:single_char' [, ...]) ]
    [ CombinedColumnSizes(size_value | 'ColumnName:size_value' [, ...]) ]
    [ TruncateColumns({'truncate_column' | 'truncate_column_range'} [, ...]) ]
    [ OutputColumnNames('output_column_name' [, ...]) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

---

## Required Syntax Elements for TD\_Pivoting

### ON clause

Specifies the table name, view name or query as an InputTable.

**PartitionColumns**

Specifies the same columns from the InputTable which are specified in the PARTITION BY clause.

**TargetColumns**

Specifies the columns from the InputTable which needs to be pivoted.

## Optional Syntax Elements for TD\_Pivoting

**Accumulate**

Specifies the copied columns to the output columns. Accumulate Column values from the last row of the partition will be copied to the output.

**RowsPerPartition**

Use RowsPerPartition when no column contains pivot keys, but you can specify a maximum number of rows in any partition. The function pivots the input rows into this number of columns in the output table. If a partition has fewer than rows\_per\_partition rows, the function adds NULL values; if a partition has more than rows\_per\_partition rows, the function omits the extra rows. If you use this argument, you must use the ORDER BY clause to order the input data; otherwise, the contents of the output table columns may vary from run to run. MinValue = 1 MaxValue = 2047, if aggregation is not given If this argument is used with aggregation, then the aggregation will consider rows\_per\_partition rows for aggregation of the data. MaxValue =INT\_MAX, if aggregation is given. Note: RowsPerPartition cannot be used with PivotColumn.

**Note:**

Required if you omit both PivotColumn and Aggregation, otherwise ignored.

**PivotColumn**

Specifies the input column name that contains the pivot keys.

PivotColumn cannot be used with RowsPerPartition.

**Note:**

Required if you omit both RowsPerPartition and Aggregation, otherwise ignored.

**PivotKeys**

Specifies the pivot\_column values to use as pivot keys. The function ignores rows that contain other pivot\_column values.

---

**Note:**

Required if you specify PivotColumn, otherwise ignored.

---

**PivotKeysAlias**

Specifies exactly one pivot key alias for each pivot\_key. The nth pivot\_key\_alias\_value applies to the nth pivot\_key.

---

**Note:**

Optional with PivotKeys, otherwise not required.

---

**DefaultPivotValues**

Specifies exactly one default value for each pivot\_key.

The nth default\_pivot\_value applies to the nth pivot\_key. Each default\_pivot\_value data type must be compatible with the target\_column data type.

---

**Note:**

Optional with PivotKeys, otherwise not required.

---

**Aggregation**

Specifies the aggregation for the target columns.

You can provide the Aggregation as one of the single value {CONCAT | UNIQUE\_CONCAT | SUM | MIN | MAX | AVG} which applies to all target columns or you can specify multiple values for multiple target columns with the following format:

'ColumnName:{CONCAT|UNIQUE\_CONCAT|SUM|MIN|MAX|AVG}' [...]

Allows only 1 aggregation for 1 target column.

---

**Note:**

Required if you omit both RowsPerPartition and PivotColumn, otherwise ignored.

---

Default: No aggregation is done only one of the values is picked up for the output.

**Delimiters**

Specifies the delimiter to use for concatenating the target column values.

A delimiter is a single character string. You can specify a single delimiter values which will be applicable to all target columns or you can specify multiple delimiter values for multiple target columns with the following format:

---

'ColumnName:single\_char' [...]

**Note:**

Optional with CONCAT and UNIQUE\_CONCAT Aggregation, otherwise not required.

---

Default: comma.

### CombinedColumnSizes

Specifies the concatenated string maximum size.

You can specify a single values which will be applicable to all target columns or you can specify multiple size value for multiple target columns with the following format:

'ColumnName:size\_value' [...]

Default value of 'size' is 64000.

Maximum value of 'size' can be : 2097088000.

Consider the following:

- If size <= 64000, VARCHAR datatype is generated for concatenated string in the output.
- If size > 64000, CLOB datatype is generated for concatenated string in the output.

Considering that CLOB operations are relatively expensive, the function always generates a VARCHAR column for concatenated string in the output, unless user specifies a size > 64000 in the query. Therefore, it is mandatory to specify a size > 64000 in the query, for generating CLOB column for concatenated string in the output.

---

**Note:**

Optional with CONCAT and UNIQUE\_CONCAT Aggregation, otherwise not required.

---

Default: 64000.

### TruncateColumns

Specifies the columns from the target columns for which you want to truncate the concatenated string, if it is longer than the specified size.

Truncate columns must be part of the target columns.

---

**Note:**

Optional with CONCAT and UNIQUE\_CONCAT Aggregation, otherwise not required.

---

Default: Nothing will be truncated.

**OutputColumnNames**

Specifies the column name to use for the output column.

The nth column name value applies to the nth output column.

**TD\_Pivoting Input****InputTable Schema**

Column	Data Type	Description
partition_column	Any allowed by PARTITION BY clause	[Column appears one time for each specified partition_column. ] Column to which to partition input data.
target_column	CHAR,VARCHAR,BYTE,VARBYTE,BYTEINT,SMALLINT,INTEGER,BIGINT,FLOAT,REAL,DOUBLE PRECISION,DECIMAL,NUMBER,DATE,TIME,TIMESTAMP,TIME WITH TIME ZONE,TIMESTAMP WITH TIME ZONE,INTERVAL SECOND,INTERVAL MINUTE TO SECOND,INTERVAL MINUTE,INTERVAL HOUR TO SECOND,INTERVAL HOUR TO MINUTE,INTERVAL HOUR,INTERVAL DAY TO SECOND,INTERVAL DAY TO MINUTE,INTERVAL DAY TO HOUR,INTERVAL DAYINTERVAL MONTH,INTERVAL YEAR TO MONTH,INTERVAL YEAR	Column to be pivoted.
accumulate_column	Any	[Column appears one time for each specified partition_column. ] Column to copy to output table from input table.

**TD\_Pivoting Output****OutputTable Schema - Only RowsPerPartition**

Column	Data Type	Description
partition_column	Same as InputTable	Columns copied to output table.
<target_column_name>_<i> or User-specified name	Same as target_column	Pivoted value. (where i is in range [0, rows_per_partition-1]. Columns appear in order specified by TargetColumns argument.
accumulate_column	Same as InputTable	Columns copied to output table.

**OutputTable Schema - PivotColumn only or PivotColumn with Aggregation**

Column	Data Type	Description
partition_column	Same as InputTable	Columns copied to output table.
<target_column_name>_<pivot_key> or <target_column_name>_<pivot_key_alias> or User-specified name	Same as target_column	Pivoted value or Aggregated value. Columns appear in order specified by TargetColumns argument and pivot keys.
accumulate_column	Same as InputTable	Columns copied to output table.

**OutputTable Schema - RowsPerPartition with Aggregation or Aggregation only**

Column	Data Type	Description
partition_column	Same as InputTable	Columns copied to output table.
<target_column_name>_combined or User-specified name	VARCHAR, CLOB	Aggregated value. Columns appear in order specified by TargetColumns argument.
accumulate_column	Same as InputTable	Columns copied to output table.

## Example: How to Use TD\_Pivoting

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [RowsPerPartition](#)
- [PivotColumn](#)
- [Aggregation with PivotColumn](#)
- [Aggregation Only](#)

## RowsPerPartition

The following example assumes the titanic dataset table exists:

```
DROP TABLE titanic_dataset_unpivoted;
CREATE MULTISET TABLE titanic_dataset_unpivoted (
    passenger INTEGER,
    AttributeName VARCHAR(128) CHARACTER SET UNICODE NOT CASESPECIFIC,
   AttributeValue VARCHAR(20) CHARACTER SET UNICODE NOT CASESPECIFIC,
    survived INTEGER
```

```
) PRIMARY INDEX ( passenger );
Insert into titanic_dataset_unpivoted(2, 'pclass', '1', 1);
Insert into titanic_dataset_unpivoted(2, 'gender', 'female', 1);
Insert into titanic_dataset_unpivoted(4, 'pclass', '1', 1);
Insert into titanic_dataset_unpivoted(4, 'gender', 'female', 1);
Insert into titanic_dataset_unpivoted(7, 'pclass', '1', 0);
Insert into titanic_dataset_unpivoted(7, 'gender', 'male', 0);
Insert into titanic_dataset_unpivoted(10, 'pclass', '2', 1);
Insert into titanic_dataset_unpivoted(10, 'gender', 'female', 1);
Insert into titanic_dataset_unpivoted(16, 'pclass', '2', 1);
Insert into titanic_dataset_unpivoted(16, 'gender', 'female', 1);
Insert into titanic_dataset_unpivoted(21, 'pclass', '2', 0);
Insert into titanic_dataset_unpivoted(21, 'gender', 'male', 0);
Insert into titanic_dataset_unpivoted(40, 'pclass', '3', 1);
Insert into titanic_dataset_unpivoted(40, 'gender', 'female', 1);
Insert into titanic_dataset_unpivoted(61, 'pclass', '3', 0);
Insert into titanic_dataset_unpivoted(61, 'gender', 'male', 0);
Insert into titanic_dataset_unpivoted(1000, 'pclass', '3', 1);
Insert into titanic_dataset_unpivoted(1000, 'gender', NULL, 1);
```

Calling TD\_Pivoting with RowsPerPartition

```
SELECT * FROM TD_Pivoting (
    ON titanic_dataset_unpivoted AS InputTable PARTITION BY passenger order
    by AttributeName
    USING
        PartitionColumns ('passenger')
        TargetColumns ('AttributeValue')
        Accumulate ('survived')
        RowsPerPartition (2)
) AS dt Order By passenger;
```

TD\_Pivoting Output

passenger	AttributeValue_0	AttributeValue_1	survived
2	1	female	1
4	1	female	1
7	1	male	0
10	2	female	1
16	2	female	1
21	2	male	0

40	3	female	1
61	3	male	0
1000	3	?	1

## PivotColumn

The following example assumes the titanic dataset table exists.

Calling TD\_Pivoting with PivotColumn

```
SELECT * FROM TD_Pivoting (
    ON titanic_dataset_unpivoted AS InputTable PARTITION BY passenger
    USING
        PartitionColumns ('passenger')
        TargetColumns ('AttributeValue')
        Accumulate ('survived')
        PivotColumn('AttributeName')
        PivotKeys('pclass','sex')
) AS dt Order By passenger;
```

TD\_Pivoting Output

passenger	AttributeValue_pclass	AttributeValue_gender	survived
2	1	female	1
4	1	female	1
7	1	male	0
10	2	female	1
16	2	female	1
21	2	male	0
40	3	female	1
61	3	male	0
1000	3	?	1

## Aggregation with PivotColumn

The following example assumes the star1 table exists:

```
CREATE MULTISET TABLE star1(country VARCHAR(20),state VARCHAR(10), yr INTEGER,
qtr VARCHAR(3), sales INTEGER, cogs INTEGER, rating varchar(10));
insert into star1 values('USA',      'CA', 2001 , 'Q1', 30, 15, 'A');
```

```
insert into star1 values('Canada', 'ON', 2001 , 'Q2', 10, 0, 'B');
insert into star1 values('Canada', 'BC', 2001 , 'Q3', 15, 0, 'A');
insert into star1 values('Canada', 'BC', 2001 , 'Q3', 10, 0, 'A');
insert into star1 values('USA', 'NY', 2001 , 'Q1', 45, 25, 'D');
insert into star1 values('USA', 'CA', 2001 , 'Q2', 50, 20, 'A');
insert into star1 values('USA', 'CA', 2001 , 'Q2', 5, 5, 'B');
```

Calling TD\_Pivoting with Aggregation with PivotColumn

```
SELECT * FROM TD_Pivoting (
    ON star1 AS InputTable PARTITION BY country, state Order By qtr
    USING
        PartitionColumns ('country', 'state')
        TargetColumns ('sales', 'cogs', 'rating')
        Accumulate('yr')
        PivotColumn('qtr')
        PivotKeys('Q1','Q2','Q3')
        Aggregation('sales:SUM', 'cogs:AVG', 'rating:CONCAT')
        Delimiters('|')
        CombinedColumnSizes(64001)
) AS dt Order By country;
```

TD\_Pivoting Output

country	state	sales_Q1	sales_Q2	sales_Q3	cogs_Q1	cogs_Q2	cogs_Q3	rating_Q1	rating_Q2	rating_Q3	yr
Canada	BC	?	?	25	?	?	0.000	?	?	A A	2001
Canada	ON	?	10	?	?	0.000	?	?	B	?	2001
USA	CA	30	55	?	15.000	12.500	?	A	B A	?	2001
USA	NY	45	?	?	25.000	?	?	D	?	?	2001

## Aggregation Only

The following example assumes the star1 table exists.

Calling TD\_Pivoting with Aggregation Only

```
SELECT * FROM TD_Pivoting (
    ON star1 AS InputTable PARTITION BY country Order By state
    USING
        PartitionColumns ('country')
        TargetColumns ('sales', 'cogs', 'state ','rating')
        Accumulate('yr')
        Aggregation('sales:SUM', 'cogs:AVG', 'state:UNIQUE_CONCAT', 'rating:CONCAT')
        Delimiters('|')
```

```
CombinedColumnSizes(64001)
) AS dt Order By country;
```

### TD\_Pivoting Output

country	sales_combined	cogs_combined	state_combined	rating_combined	yr
Canada	35	0.000	BC   ON	A   A   B	2001
USA	130	16.250	CA   NY	A   B   A   D	2001

## TD\_PolynomialFeaturesFit

TD\_PolynomialFeaturesFit function stores all the specified values in an argument in a tabular format.

All polynomial combinations of the values with degrees less than or equal to the specified degree are generated. For example, for a 2-D input sample [x, y], the degree-2 polynomial features are [x, y, x-squared, xy, y-squared, 1].

Use the TD\_PolynomialFeatureFit method to fit a polynomial function to a given dataset. The method involves transforming the original input features into a set of polynomial features, which you can use to fit a polynomial function to the dataset.

For example, a dataset of  $n$  observations, where each observation is represented by a set of  $p$  input features,  $X = \{x_1, x_2, \dots, x_p\}$ , and a target variable,  $Y$ . You can represent this dataset as a matrix  $X$  of size  $n \times p$ , where each row represents an observation, and a vector  $Y$  of size  $n$ .

The TD\_PolynomialFeatureFit method involves transforming the input feature matrix  $X$  into a new matrix  $X'$ , where each column of  $X'$  represents a polynomial function of the original input features. The degree of the polynomial function used for the transformation is a hyperparameter that needs to be specified.

If the degree of the polynomial function is 2, then the transformed input feature matrix  $X'$  would be:

$$X' = [1, x_1, x_2, \dots, x_p, x_1^2, x_2^2, \dots, x_p^2]$$

where the first column represents a constant term, and the remaining columns represent the original input features and their pairwise products.

After obtaining the input feature matrix  $X'$ , you can fit a polynomial function to the dataset by solving the following equation:

$$Y = X' \beta + \epsilon$$

where  $\beta$  is a vector of coefficients that define the polynomial function and  $\epsilon$  is a vector of error terms. You can estimate the coefficients  $\beta$  using linear regression techniques, such as ordinary least squares (OLS) regression.

The degree of the polynomial function used for the fit is an important hyperparameter that you need to tune. Higher degree polynomial functions capture more complex relationships, but also run the risk of overfitting

the data. Best practice is to balance the complexity of the model with the available data and the desired level of accuracy.

## Function Information

- [TD\\_PolynomialFeaturesFit Syntax](#)
- [Required Syntax Elements for TD\\_PolynomialFeaturesFit](#)
- [Optional Syntax Elements for TD\\_PolynomialFeaturesFit](#)
- [TD\\_PolynomialFeaturesFit Input](#)
- [TD\\_PolynomialFeaturesFit Output](#)
- [Example: How to Use TD\\_PolynomialFeaturesFit](#)

## TD\_PolynomialFeaturesFit Syntax

```
TD_PolynomialFeaturesFit (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table) ]
    USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        [ IncludeBias ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
        [ InteractionOnly ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
        [ Degree (degree) ]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_PolynomialFeaturesFit

### ON clause

Accepts the *InputTable* clause.

### TargetColumns

Specifies the names of the *InputTable* columns to output polynomial combinations for features. Column limit is five.

## Optional Syntax Elements for TD\_PolynomialFeaturesFit

### OUT clause

Accepts the OutputTable clause.

### OutputTable

Specifies a name for the output table.

### IncludeBias

Specifies whether the output table is to include a bias column for the feature in which all polynomial powers are zero (that is, a column of ones). A bias column acts as an intercept term in a linear model.

Default: true

### InteractionOnly

Specifies whether to output polynomial combinations only features that are products of at most degree distinct input features.

Default: false

### Degree

Specifies the maximum degree of the input features for which to output polynomial combinations, an integer in the range [1,2,3].

Default: 2

## TD\_PolynomialFeaturesFit Input

### InputTable Schema

Column	Data Type	Description
target_column	NUMERIC	Column for which to output polynomial combinations for features.

## TD\_PolynomialFeaturesFit Output

### OutputTable Schema

Column	Data Type	Description
TD_IncludeBias_ POLFIT:Boolean	INTEGER	1 if Boolean is 'True', 0 if Boolean is 'False'.

Column	Data Type	Description
TD_InteractionOnly_POLFIT:Boolean	INTEGER	1 if Boolean is 'True', 0 if Boolean is 'False'.
TD_Degree_POLFIT:degree	INTEGER	degree
target_column	NUMERIC	Preserves target column name for TD_PolynomialFeaturesTransform. Contains only NULL values.

## Example: How to Use TD\_PolynomialFeaturesFit

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_PolynomialFeaturesFit InputTable: polynomialFeaturesFit\_input

```
id col1 col2 col3
-- -----
1   2   3   4
2   5   6   7
3   1   2   4
4   5   3   5
5   3   2   6
```

### Example: TD\_PolynomialFeaturesFit SQL Call

```
CREATE TABLE polynomialFit AS (
  SELECT * FROM TD_PolynomialFeaturesFit (
    ON polynomialFeaturesFit_input AS InputTable
    USING
      TargetColumns ('[1:2]')
      Degree (2)
  ) AS dt
) WITH DATA;
```

### TD\_PolynomialFeaturesFit Output

TD_INCLUDEBIAS_POLFIT:TRUE	TD_INTERACTIONONLY_POLFIT:FALSE	TD_DEGREE_POLFIT:2	col1	col2
1	0	2	NULL	NULL

## TD\_PolynomialFeaturesTransform

TD\_PolynomialFeaturesTranform function extracts values of arguments [TargetColumns, Degree, IncludeBias, and InteractionOnlygenerates] from the output of the [TD\\_PolynomialFeaturesFit](#) function and generates a feature matrix of all polynomial combinations of the features.

TD\_PolynomialFeaturesTransform is a method used to transform the original input feature matrix X into a new matrix X', where each column represents a polynomial combination of the original features. This can be expressed mathematically as follows:

$$X' = [1, X, X^2, \dots, X^d]$$

where X is the original input feature matrix, d is the degree of the polynomial function, and  $X^k$  represents the matrix of all combinations of the original features up to degree k. The first column of the transformed feature matrix represents a constant term, and the remaining columns represent the original input features and their pairwise products up to the specified degree.

For example, if X has two features,  $x_1$  and  $x_2$ , and the degree of the polynomial function is 2, then the transformed input feature matrix X' would be:

$$X' = [1, x_1, x_2, x_1x_2, x_1^2, x_2^2]$$

This transformation enables you to capture nonlinear relationships between the input features and the target variable. By including polynomial combinations of the original features, you can model more complex relationships than what is possible with linear models.

In addition to polynomial combinations, the TD\_PolynomialFeaturesTransform method can also include interaction terms and cross-terms, which are useful in certain applications. Interaction terms involve multiplying two or more features together, while cross-terms involve multiplying features from different inputs.

After obtaining the input feature matrix X', you can fit a polynomial function to the dataset using regression techniques such as ordinary least squares (OLS) regression or regularized regression.

Overall, the TD\_PolynomialFeaturesTransform method is a powerful tool for capturing nonlinear relationships between the input features and the target variable and is widely used in fields such as data science, engineering, and economics.

### Function Information

- [TD\\_PolynomialFeaturesTransform Syntax](#)
- [Required Syntax Elements for TD\\_PolynomialFeaturesTransform](#)
- [Optional Syntax Elements for TD\\_PolynomialFeaturesTransform](#)
- [TD\\_PolynomialFeaturesTransform Input](#)
- [TD\\_PolynomialFeaturesTransform Output](#)
- [Example: How to Use TD\\_PolynomialFeaturesTransform](#)

## TD\_PolynomialFeaturesTransform Syntax

```
TD_PolynomialFeaturesTransform (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    ON { table | view | (query) } AS FitTable DIMENSION
    USING
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_PolynomialFeaturesTransform

**ON clause**

Accepts the InputTable and FitTable clauses.

## Optional Syntax Elements for TD\_PolynomialFeaturesTransform

**Accumulate**

Specifies the names of the InputTable columns to copy to the output table.

**Note:**

If two or more column names are concatenated and the column name exceeds 128 characters, then the function replaces the actual column names with names such as col1, col2, col3, col4, col5 in the output.

## TD\_PolynomialFeaturesTransform Input

**InputTable Schema**

Column	Data Type	Description
target_column	NUMERIC	Column for which to output polynomial combinations for features.

## FitTable Schema

Column	Data Type	Description
TD_IncludeBias_ POLFIT:Boolean	INTEGER	1 if Boolean is 'True', 0 if it is 'False'.
TD_InteractionOnly_ POLFIT:Boolean	INTEGER	1 if Boolean is 'True', 0 if it is 'False'.
TD_Degree_POLFIT:degree	INTEGER	degree
target_column	NUMERIC	Preserves target column name for TD_PolynomialFeaturesTransform. Contains only NULL values.

## TD\_PolynomialFeaturesTransform Output

### Output Table Schema

Column	Data Type	Description
accumulate_column	Same as in InputTable.	Column copied from InputTable.
One	DOUBLE PRECISION	[Column appears only with IncludeBias ('true').] Column for feature in which all polynomial powers are zero (column of ones).
output_column	DOUBLE PRECISION	[Column appears once for each polynomial combination of degree or less.] Polynomial combination. For example, for two-dimensional input feature [x, y], output columns are x, y, x_square, y_square, x_y, where x and y are InputTable target columns.

## Example: How to Use TD\_PolynomialFeaturesTransform

### TD\_PolynomialFeaturesFit InputTable: polynomialFeaturesFit\_input

InputTable: polynomialFeatures, as in [Example: How to Use TD\\_PolynomialFeaturesFit](#)

```
id col1 col2 col3
-- ---- -----
1   2   3   4
2   5   6   7
3   1   2   4
4   5   3   5
5   3   2   6
```

## TD\_PolynomialFeaturesFit OutputTable: polynomialFit

FitTable: polynomialFit, created by [Example: How to Use TD\\_PolynomialFeaturesFit](#)

```
TD_PolynomialFeaturesFit Output
TD_INCLUDEBIAS_POLFIT:TRUE TD_INTERACTIONONLY_POLFIT:FALSE TD_DEGREE_POLFIT:2
col1 col2
-----
----- 1 0 2
NULL NULL
```

## Example: Using TD\_PolynomialFeaturesTransform SQL Call

```
SELECT * FROM TD_PolynomialFeaturesTransform (
    ON polynomialFeaturesFit_input AS InputTable PARTITION BY ANY
    ON polynomialFit AS FitTable DIMENSION
    USING
    Accumulate ('[0:0]')
) AS dt;
```

## TD\_PolynomialFeaturesTransform Output

id	ONE	col1	col1_col2	col1_SQUARE	col2	col2_SQUARE
1	1.0	2.0	6.0	4.0	3.0	9.0
2	1.0	5.0	30.0	25.0	6.0	36.0
3	1.0	1.0	2.0	1.0	2.0	4.0
4	1.0	5.0	15.0	25.0	3.0	9.0
5	1.0	3.0	6.0	9.0	2.0	4.0

## TD\_RandomProjectionFit

The TD\_RandomProjectionFit function returns a random projection matrix based on the specified arguments.

The function also returns the required parameters for transforming the input data into lower-dimensional data. The TD\_RandomProjectionTransform function uses the TD\_RandomProjectionFit output to reduce the dimensionality of the input data.

### Function Information

- [TD\\_RandomProjectionFit Syntax](#)
- [TD\\_RandomProjectionFit Syntax Elements](#)

- [TD\\_RandomProjectionFit Input](#)
- [TD\\_RandomProjectionFit Output](#)
- [TD\\_RandomProjectionFit Example](#)

## TD\_RandomProjectionFit Syntax

```
TD_RandomProjectionFit
(ON {table | view | (query)} as InputTable
[ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable(out_table_name) ]
USING
TargetColumns({'target_column' | 'target_column_range'} [,...])
NumComponents(num_components)
[ Seed(seed_value) ]
[ Epsilon(epsilon_value) ]
[ ProjectionMethod({'GAUSSIAN' | 'SPARSE'}) ]
[ Density(density_value) ]
[ OutputFeatureNamesPrefix('output_feature_names_prefix') ])
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## TD\_RandomProjectionFit Syntax Elements

### TargetColumns

[Required]: Specify the input table columns for dimensionality reduction.

### NumComponents

[Required]: Specify the target dimension (number of features) on which the data points from the original dimension are projected.

The NumComponents value cannot be greater than the original dimension (number of features) and must satisfy the Johnson-Lindenstrauss Lemma result. The minimum value allowed for the NumComponents argument is calculated using the TD\_RandomProjectionMinComponents function.

**Seed**

[Optional]: Specify the random seed the algorithm uses for repeatable results. The algorithm uses the seed to generate a random projection matrix. The seed must be a non-negative integer value.

Default Value: The Random Seed value is used for generating a random projection matrix, and hence the output is non-deterministic.

**Epsilon**

[Optional]: Specify a value to control distortion introduced while projecting the data to a lower dimension. The amount of distortion increases if you increase the value.

Default Value: 0.1

Allowed Values: Between 0 and 1

**ProjectionMethod**

[Optional]: Specify the method name for generating the random projection matrix.

Default Value: GAUSSIAN

Allowed Values: [GAUSSIAN, SPARSE]

**Density**

[Optional]: Specify the approximate ratio of non-zero elements in the random projection matrix when SPARSE is used as the projection method.

Default Value: 0.33333333

Allowed Values: 0 < Density <= 1

**OutputFeatureNamesPrefix**

[Optional]: Specify the prefix for the output column names.

Default Value: td\_rpi\_feature

**TD\_RandomProjectionFit Input****Input Table Schema**

Column	Data Type	Description
Target_column	BYTEINT,SMALLINT,INTEGER,BIGINT, Decimal/Numeric,Float,Real,Double precision	The input table columns for dimensionality reduction.

## TD\_RandomProjectionFit Output

### Output Table Schema

Column	Data Type	Description
OutputFeatureNamesPrefix_NumComponents_0	INTEGER	The combination of OutputFeatureNamesPrefix and NumComponents values is used for the column name and contains a unique identifier of rows in the Random Projection matrix.
Target_column	REAL	The columns that have the elements of the Random Projection Matrix.

## TD\_RandomProjectionFit Example

### TD\_RandomProjectionFit Input Table

Each of the input data points in the following input table consists of 963 columns and a unique identifier of the data point:

company_id	date_2010_01_04	date_2010_01_05	...	date_2013_10_28	date_2013_10_29
1.840019	0.58	-0.220005	....		
2	-19.589981				
3	-0.640002	-0.65	-0.400002	0.66	
3.740021	-2.350006	1.260009		....	-1.760009
4	0.109997	0	0.040001	0.540001	
5	0.459999	1.77	....	1.130005	0.309998
6	0.45				
0.460001			-0.06	-0.11	
7	0.18	0.220001		....	0.330002
1.150001					
8	0.73	0.369999			
0.090001	-0.110001				
9	0.899997	0.700001		....	-0.220001
0.159996					
10	0.36	0.909996			
1.070003	1.050003				

### TD\_RandomProjectionFit SQL Call

```

SELECT * FROM TD_RandomProjectionFit(
ON stock_movement as InputTable
OUT PERMANENT TABLE OutputTable(rand_proj_fit_tbl_ex)
USING
TargetColumns('[1:]')
NumComponents(353)
Seed(0)
Epsilon(0.25)
ProjectionMethod('GAUSSIAN')
) as dt ORDER BY 1,2;

```

## Output Table

td_rpj_feature_353_0	date_2010_01_04	date_2010_01_05	...	date_2013_10_28	date_2013_10_29	
0		-0.068021509		-0.021542237	....	0.047328448
0.031534748	-0.021109446		0.076606803	0.033381927	....	
1			-0.100981365			
0.039908753		-0.035816093	-0.009469693			
2		....	....	....	....	
0.026625478						
....		....	....	....	....	
....			....	....	....	
....				....	....	
351			0.039755885	-0.040469189	....	0.046338113
0.018347439			0.040278453			
352		....	-0.04456492	-0.031728421		
0.072748211						

## TD\_RandomProjectionMinComponents

The TD\_RandomProjectionMinComponents function calculates the minimum number of components required for applying RandomProjection on the given dataset for the specified epsilon(distortion) parameter value. The function estimates the minimum value of the NumComponents argument in the TD\_RandomProjectionFit function for a given dataset. The function uses the Johnson-Lindenstrauss Lemma algorithm to calculate the value.

### Function Information

- [TD\\_RandomProjectionMinComponents Syntax](#)
- [TD\\_RandomProjectionMinComponents Syntax Elements](#)
- [TD\\_RandomProjectionMinComponents Input](#)
- [TD\\_RandomProjectionMinComponents Output](#)
- [TD\\_RandomProjectionMinComponents Example](#)

## TD\_RandomProjectionMinComponents Syntax

```
TD_RandomProjectionMinComponents(
  ON {table | view | (query)} as InputTable
  USING
  TargetColumns({'target_column' | 'target_column_range'} [, ...])
  [ Epsilon(epsilon_value) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## TD\_RandomProjectionMinComponents Syntax Elements

### TargetColumns

[Required]: Specify the input columns for random projection.

### Epsilon

[Optional]: Specify a value to control distortion introduced while projecting the data to a lower dimension. The amount of distortion increases if you increase the value.

Default Value: 0.1

Allowed Values: Between 0 and 1

## TD\_RandomProjectionMinComponents Input

### Input Table Schema

Column	Data Type	Description
Target_column	BYTEINT,SMALLINT,INTEGER,BIGINT, Decimal /Numeric,Float,Real,Double precision	The input columns for random projection.

## TD\_RandomProjectionMinComponents Output

### Output Table Schema

Column	Data Type	Description
RandomProjection_MinComponents	INTEGER	The minimum number of components computed using the Epsilon value to apply the RandomProjection function to the dataset.

## TD\_RandomProjectionMinComponents Example

### TD\_RandomProjectionMinComponents Input Table

Each of the input data points in the following input table consists of 963 columns and a unique identifier of the data point:

	company_id	date_2010_01_04	date_2010_01_05	....	date_2013_10_28	date_2013_10_29
1	0.840019	0.58	-0.220005			
2		-19.589981				
3		-0.640002	-0.65	-0.400002	0.66	
3.740021		-2.350006	1.260009		....	-1.760009
4		0.109997	0	0.040001	0.540001	
5		0.459999	1.77	....	1.130005	0.309998
6		0.45				
0.460001				-0.06	-0.11	
7		0.18		0.220001	....	0.330002
1.150001						

8	0.73	0.369999		
0.090001	-0.110001			
9	0.899997	0.700001	....	-0.220001
0.159996				
10	0.36	0.909996		
1.070003	1.050003			

## TD\_RandomProjectionMinComponents SQL Call

```
SELECT * FROM TD_RandomProjectionMinComponents(
ON stock_movement as InputTable
USING
TargetColumns('[1:]')
Epsilon(0.25)
) as dt;
```

## TD\_RandomProjectionMinComponentsOutput

```
randomprojection_mincomponents
-----
353
```

## TD\_RandomProjectionTransform

The TD\_RandomProjectionTransform function converts the high-dimensional input data to a lower-dimensional space using the TD\_RandomProjectionFit function output.

### Function Information

- [TD\\_RandomProjectionTransform Syntax](#)
- [TD\\_RandomProjectionTransform Syntax Elements](#)
- [TD\\_RandomProjectionTransform Input](#)
- [TD\\_RandomProjectionTransform Output](#)
- [TD\\_RandomProjectionTransform Example](#)

## TD\_RandomProjectionTransform Syntax

```
TD_RandomProjectionTransform(
ON {table | view | (query)} as InputTable
ON {table | view | (query)} as FitTable DIMENSION
USING
[ Accumulate({'accumulate_column' | 'accumulate_column_range'} [,...]) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## TD\_RandomProjectionTransform Syntax Elements

### Accumulate

[Optional]: Specify the input table columns to copy to the output table.

Default: Only transformed columns are present in the output.

## TD\_RandomProjectionTransform Input

### Input Table Schema

Column	Data Type	Description
Target_column	BYTEINT,SMALLINT, INTEGER,BIGINT, Decimal /Numeric,Float,Real, Double precision	The input table columns for dimensionality reduction.
Accumulate_column	ANY	The input table columns that you want to copy to the output table.

## TD\_RandomProjectionTransform Output

### Output Table Schema

Column	Data Type	Description
Accumulate	ANY	The specified columns in the Accumulate element is copied to the output table.
OutputFeatureNamesPrefix_i	REAL	The rendered columns after converting the data points to lower-dimensional space wherein i is the sequence number of the generated column.

## TD\_RandomProjectionTransform Example

### TD\_RandomProjectionTransform Input Table

Each of the input data points in the following input table consists of 963 columns and a unique identifier of the data point:

company_id	date_2010_01_04	date_2010_01_05	....	date_2013_10_28	date_2013_10_29
1	0.58	-0.220005	....		
0.840019	-19.589981				
2	-0.640002	-0.65	-0.400002	0.66	
3	-2.350006	1.260009		....	-1.760009
3.740021					
4	0.109997	0	0.040001	0.540001	
5	0.459999	1.77	....	1.130005	0.309998
6	0.45				
0.460001			-0.06	-0.11	
7	0.18	0.220001		....	0.330002
1.150001					
8	0.73	0.369999			
0.090001	-0.110001				
9	0.899997	0.700001		....	-0.220001
0.159996					
10	0.36	0.909996			
1.070003	1.050003				

### FITTable (Generated from TD\_RandomProjectionFit)

td_rpj_feature_353_0	date_2010_01_04	date_2010_01_05	....	date_2013_10_28	date_2013_10_29
0			-0.068021509	-0.021542237	....
0.031534748	-0.021109446				
1		0.076606803		0.033381927	....
0.039908753					0.047328448
2			-0.100981365		
0.026625478	....	-0.035816093	-0.009469693		
....		....	....	....	....
....		....	....	....	....
....		....	....	....	....
351		0.039755885		-0.040469189	....
0.018347439					0.046338113
352		0.040278453			
0.072748211	....	-0.04456492	-0.031728421		

### TD\_RandomProjectionTransform SQL Call

```
SELECT * FROM TD_RandomProjectionTransform(
  ON stock_movement as InputTable
  ON rand_proj_fit_tbl_ex as FitTable DIMENSION
  USING
  Accumulate('company_id')
) as dt ORDER BY 1,2;
```

### TD\_RandomProjectionTransform Output Table

company_id	td_rpj_feature_0	td_rpj_feature_1	....	td_rpj_feature_351
1	5.545038445	-18.76927716		-4.260967918
1.729951186				
2				
0.38584145	-1.436265087	....	-0.666327415	-0.820435756
3	-2.173481389	-1.528033248		
2.975176363	-4.719261735			
4	-0.193832387	-1.093648166	....	0.07893702
0.923864022				
5	-1.542301006	-1.794068037		-0.795435421
0.797221691				
6	-0.344011761	-0.344862717	....	0.13398249
0.088007141				
7	0.899506286	-1.592144274		-1.225987822

0.192506954				
8	0.446963102	-0.736407073	....	0.133092795
0.192294245		-3.933652254		-0.086266401
9	-3.02747734			
-2.545520414				
10	1.937290638	-0.393961865	-0.88549645	
0.498908831	....			

## TD\_RowNormalizeFit

TD\_RowNormalizeFit outputs a table of parameters and specified input columns to input to [TD\\_RowNormalizeTransform](#), which normalizes the input columns row-wise.

Normalization is a data preprocessing technique used in machine learning to scale data to a common range. It is a process of transforming numerical data to a standardized format, ensuring that no variable dominates the others.

Row normalization is used to scale the values in each row of a dataset to a specific norm or range. This ensures that all rows have equal importance in the analysis and reduces the influence of the magnitude of the features on the results.

There are several benefits of normalizing your data row-wise:

- Comparing variables: Compare variables that are measured on different scales. By normalizing the rows, variables with different scales can be compared on a common scale.
- Data preprocessing: Pre-processing machine learning pipelines to improve the performance of machine learning models. Normalizing the rows can make the data more adaptable for clustering, classification, and other machine learning algorithms.
- Interpretation: Improve interpretation of the data. By normalizing the rows, the data can be transformed into a standard format that is easier to understand and analyze.
- Outlier handling: Handle outliers. Outliers can skew the data and make it difficult to identify patterns. Normalizing the rows can help to reduce the impact of outliers by scaling the values to a standard range.

Row normalization is particularly useful when dealing with datasets that have features with significantly different scales or ranges. By normalizing the rows, the algorithm can effectively compare the similarity or distance between the rows based on the direction of the feature vectors, rather than the magnitude of their values. Different normalization functions are used based on the type of data and the normalization goal.

## Function Information

- [TD\\_RowNormalizeFit Syntax](#)
- [Required Syntax Elements for TD\\_RowNormalizeFit](#)
- [Optional Syntax Elements for TD\\_RowNormalizeFit](#)
- [TD\\_RowNormalizeFit Input](#)
- [TD\\_RowNormalizeFit Output](#)
- [Example: How to Use TD\\_RowNormalizeFit](#)

## TD\_RowNormalizeFit Syntax

```
TD_RowNormalizeFit (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table_name) ]
    USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        [ Approach ({ 'UNITVECTOR' | 'FRACTION' | 'PERCENTAGE' | 'INDEX' }) ]
        [ BaseColumn ('base_column')
            BaseValue (base_value)
        )
    )
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_RowNormalizeFit

### ON clause

Accept the InputTable clause.

### TargetColumns

Specify the names of the InputTable columns to normalize row-wise.

### BaseColumn

[Required with Approach ('INDEX'), ignored otherwise.] Specify the name of the InputTable column that has the  $B$  values to use in the normalizing formula.

### BaseValue

[Required with Approach ('INDEX'), ignored otherwise.] Specify the  $V$  value to use in the normalizing formula.

## Optional Syntax Elements for TD\_RowNormalizeFit

### OUT clause

Accept the OutputTable clause.

## Approach

Specify the normalization method:

Option	Description	Normalizing Formula
UNITVECTOR (Default)	Scales each row of the dataset to have a unit norm or magnitude of 1.	$X' = X / (\sqrt{(\sum_{i \in [1, n]} X_i^2)})$
FRACTION	Scales each row of the dataset to have a sum of 1. This function is useful when the values in each row represent fractions or probabilities and need to add up to 1.	$X' = X / (\sum_{i \in [1, n]} X_i)$
PERCENTAGE	Scales each row of the dataset to have a sum of 100. This function is useful when the values in each row represent percentages and need to add up to 100.	$X' = X * 100 / (\sum_{i \in [1, n]} X_i)$
INDEX	Scales each row of the dataset to a specified range between V and B.	$X' = V + ((X - B) / B) * 100$

In the normalizing formulas:

$X'$  is the normalized value.

$X$  is the original value.

$B$  is the value in the base column.

$V$  is the base value.

## TD\_RowNormalizeFit Input

### InputTable Schema

Column	Data Type	Description
target_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Column to normalize row-wise.
base_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	[Column appears only with Approach ('INDEX').] B value to use in normalizing formula.

## TD\_RowNormalizeFit Output

### OutputTable Schema

Column	Data Type	Description
TD_KEY_ROWFIT	VARCHAR	Parameter key.

Column	Data Type	Description
	(CHARACTER SET LATIN)	
TD_VALUE_ROWFIT	VARCHAR (CHARACTER SET UNICODE)	Parameter value.
target_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	[Column appears once for each specified target_column.] NULL value.

## Example: How to Use TD\_RowNormalizeFit

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### InputTable: rowNormalizeFit\_input

```
id x y
-- - --
1 0 1
2 3 4
3 5 12
4 7 24
```

### TD\_RowNormalizeFit SQL Call

```
CREATE TABLE rowNormalizeFit_output AS (
  SELECT * FROM TD_RowNormalizeFit (
    ON rowNormalizeFit_input AS InputTable
    USING
      TargetColumns ('[1:2]')
      Approach ('INDEX')
      BaseColumn ('y')
      BaseValue (100)
  ) AS dt
) WITH DATA;
```

### TD\_RowNormalizeFit Output

```
TD_KEY_ROWFIT TD_VALUE_ROWFIT x     y
----- -----
Approach      INDEX            null  null
```

BaseColumn	y	null null
BaseValue	100	null null

## TD\_RowNormalizeTransform

TD\_RowNormalizeTransform normalizes input columns row-wise, using [TD\\_RowNormalizeFit](#) output.

Row normalization is a technique to transform a matrix or a dataset so that each row has the same magnitude or scale. This is typically done to make it easier to compare rows and to avoid bias towards variables with higher values.

Suppose you have a table that represents the daily sales figures for a retail store over a period of five days.

Day 1	Day 2	Day 3	Day 4	Day 5
120	150	80	200	90
90	110	100	120	130
200	180	150	170	190

One method of normalization is to divide each element in a row by the sum of all the elements in that row.

Day 1	Day 2	Day 3	Day 4	Day 5
120/640	150/640	80/640	200/640	90/640
90/550	110/550	100/550	120/550	130/550
200/890	180/890	150/890	170/890	190/890

The result is a table where each row has a sum of 1, representing a proportion or percentage of the total sales for each day.

Day 1	Day 2	Day 3	Day 4	Day 5
0.1875	0.2344	0.125	0.3125	0.1406
0.2079	0.2539	0.2308	0.2771	0.3001
0.2247	0.2022	0.1685	0.1910	0.2135

Each row contains normalized values for total sales for each day, making it easier to compare the performance of the store on different days using a machine learning pipeline because all values have the same impact and magnitude.

### Function Information

- [TD\\_RowNormalizeTransform Syntax](#)
- [Required Syntax Elements for TD\\_RowNormalizeTransform](#)

- [Optional Syntax Elements for TD\\_RowNormalizeTransform](#)
- [TD\\_RowNormalizeTransform Input](#)
- [TD\\_RowNormalizeTransform Output](#)
- [Example: How to Use TD\\_RowNormalizeTransform](#)

## TD\_RowNormalizeTransform Syntax

```
TD_RowNormalizeTransform (
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY [ ORDER BY
order_column ] ]
    ON { table | view | (query) } AS FitTable DIMENSION
    USING
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_RowNormalizeTransform

### ON clause

Accept the InputTable and FitTable clauses.

## Optional Syntax Elements for TD\_RowNormalizeTransform

### Accumulate

Specify the names of the InputTable columns to copy to the output table.

## TD\_RowNormalizeTransform Input

### InputTable Schema

Column	Data Type	Description
target_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL	Column to normalize row-wise.

Column	Data Type	Description
	/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	
base_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL /NUMERIC, FLOAT, REAL, DOUBLE PRECISION	[Column appears only with Approach ('INDEX') from the TD_RowNormalizeFit function.] B value to use in normalizing formula.
accumulate_column	Any	The input table column names copied to the output table.

### FitTable Schema

Column	Data Type	Description
TD_KEY_ROWFIT	VARCHAR (CHARACTER SET LATIN)	Parameter key.
TD_VALUE_ROWFIT	VARCHAR (CHARACTER SET UNICODE)	Parameter value.
target_column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	[Column appears once for each specified target_column.] NULL value.

### TD\_RowNormalizeTransform Output

Column	Data Type	Description
accumulate_column	Any	Column copied from InputTable.
target_column	DOUBLE PRECISION	Row-normalized values.

### Example: How to Use TD\_RowNormalizeTransform

#### InputTable from TD\_RowNormalizeFit Output: rowNormalizeFit\_input

```
id x y
-- - -
1 0 1
2 3 4
3 5 12
4 7 24
```

## FitTable from TD\_RowNormalizeFit Output: rowNormalizeFit\_output

TD_KEY_ROWFIT	TD_VALUE_ROWFIT	x	y
Approach	INDEX	null	null
BaseColumn	y	null	null
BaseValue	100	null	null

## TD\_RowNormalizeTransform SQL Call

```
SELECT * FROM TD_RowNormalizeTransform (
    ON rowNormalizeFit_input AS InputTable
    ON rowNormalizeFit_output AS FitTable DIMENSION
    USING
        Accumulate ('id')
) AS dt;
```

## TD\_RowNormalizeTransform Output

id	x	y
1	0.00	100.00
2	75.00	100.00
3	41.66	100.00
4	29.16	100.00

## TD\_ScaleFit

TD\_ScaleFit outputs a table of statistics to input to [TD\\_ScaleTransform](#), which scales specified input table columns. TD\_ScaleFit accepts input data in dense and sparse format.

Scale fit refers to the process of transforming variables in a dataset to have a particular scale or range of values that is more suitable for analysis or modeling. This is important because variables in a dataset may have different units of measurement and/or different ranges of values, which can lead to bias in the analysis or model performance.

There are several methods of scaling variables in a dataset, including:

- **Min-Max scaling:** This method scales variables to a range between 0 and 1. The formula for min-max scaling is:  

$$\text{scaled value} = (\text{value}-\min)/(\max-\min)$$
 where min is the minimum value of the variable in the dataset, max is the maximum value of the variable in the dataset, and value is the original value of the variable.

- **Standardization:** This method scales variables to have a mean of 0 and a standard deviation of 1. The formula for standardization is:

scaled value=(value-mean)/(standard deviation)

where mean is the mean value of the variable in the dataset, standard\_deviation is the standard deviation of the variable in the dataset, and value is the original value of the variable.

- **Logarithmic scaling:** This method transforms variables by taking the natural logarithm of their values. This can be useful for variables that have a wide range of values or are heavily skewed.
- **Power transformation:** This method transforms variables by raising them to a power. Common power transformations include the square root, cube root, and inverse transformations.
- **Robust scaling:** This method scales variables to have a median of 0 and a range between the 25th and 75th percentiles. This can be useful for variables that have extreme outliers.

It's important to note that the choice of scaling method depends on the specific dataset and analysis or modeling goals. Some methods may work better for certain types of variables or distributions than others. Additionally, it's important to evaluate the impact of scaling on the analysis or model performance to ensure that the scaling method is appropriate for the data.

## Usage Considerations

The following are usage considerations for TD\_ScaleFit function:

ON clause

- The InputTable in the TD\_ScaleFit query can have no partition at all or have the following combinations of PARTITION BY/ORDER BY clauses.
  - PARTITION BY ANY ORDER BY
  - PARTITION BY ANY
  - PARTITION BY KEY
  - PARTITION BY KEY ORDER BY
- The ParameterTable and AttributeTable in TD\_ScaleFit query can have partition with following combinations:
  - PARTITION BY KEY
  - PARTITION BY KEY ORDER BY
- TargetColumns and ScaleMethod arguments are mandatory for TD\_ScaleFit function. Otherwise an error is reported. For example: "Failure 7810 Error in function TD\_ScaleFit: Required argument TargetColumns is missing."
- If IgnoreInvalidLocationScale is set to false, and ScaleMethod is set to STD, then the TD\_ScaleFit function will report an error if standard deviation of the column is 0. For example: "\*\*\* Failure 9134 Error in function TD\_ScaleFit: Cannot apply STD ScaleMethod on columns with STD = 0."
- If the TD\_ScaleFit and TD\_ScaleTransform functions are invoked on data with large partition sizes and high concurrency, the spool space for the database user may exhaust during partitioning of data and throw one of the following errors:

- \*\*\* Failure 9794 File system has reported ERRAMPOUTOFPHYSPACE error.
- \*\*\* Failure 2646 No more spool space in *database\_user*.

## Limits and Restrictions

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs) and KanjiSJIS or Graphic data types.

## Function Information

- [TD\\_ScaleFit Syntax](#)
- [Required Syntax Elements for TD\\_ScaleFit](#)
- [Optional Syntax Elements for TD\\_ScaleFit](#)
- [TD\\_ScaleFit Input](#)
- [TD\\_ScaleFit Output](#)
- [Examples: How to Use TD\\_ScaleFit](#)

## TD\_ScaleFit Syntax

### TD\_ScaleFit Syntax without Partition

```
TD_ScaleFit (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable(output_table_name) ]
    USING
        ScaleMethod ({'MEAN' | 'SUM' | 'USTD' | 'STD' | 'RANGE' | 'MIDRANGE' | 'MAXABS'
        | 'RESCALE({lb=lower_bound | ub=upper_bound | lb=lower_bound,ub=upper_bound})'}
        [, ...])
        [ MissValue ({ 'KEEP' | 'ZERO' | 'LOCATION' }) ]
        [ GlobalScale ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
        [ Multiplier ('multiplier' [, ...]) ]
        [ Intercept ('intercept' [, ...]) ]
        [ IgnoreInvalidLocationScale
        ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
        { for_dense_input | for_sparse_input }
)
```

### *for\_dense\_input*

```
TargetColumns ( { 'target_column' | 'target_column_range' } [, ...] )
```

**for\_sparse\_input**

```

AttributeNameColumn ('attribute_name_column')
AttributeValueColumn ('attribute_value_column')
[ TargetAttributes ('target_attribute' [,...]) ]

```

**TD\_ScaleFit Syntax with Partition**

```

TD_ScaleFit (
ON { table | view | (query) } AS InputTable PARTITION BY partition_column [,...]
[ ON { table | view | (query) } AS ParameterTable PARTITION BY partition_column
[,...] ]
[ ON { table | view | (query) } AS AttributeTable PARTITION BY partition_column
[,...] ]
[ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable(output_table_name) ]
USING
ScaleMethod ({'MEAN' | 'SUM' | 'USTD' | 'STD' | 'RANGE' | 'MIDRANGE' | 'MAXABS'
| 'RESCALE({lb=lower_bound | ub=upper_bound | lb=lower_bound,ub=upper_bound})'}
[,...])
[ PartitionColumns ('partition_column' [,...]) ]
[ MissValue ({ 'KEEP' | 'ZERO' | 'LOCATION' }) ]
[ GlobalScale ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ Multiplier ('multiplier' [,...]) ]
[ Intercept ('intercept' [,...]) ]
[ IgnoreInvalidLocationScale
({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
{ for_dense_input | for_sparse_input }
)

```

**for\_dense\_input**

```

TargetColumns ( { 'target_column' | 'target_column_range' } [,...] )
[ UnusedAttributes ({'UNSCALED' | 'NULLIFY'}) ]

```

**for\_sparse\_input**

```

AttributeNameColumn ('attribute_name_column')
AttributeValueColumn ('attribute_value_column')
[ TargetAttributes ('target_attribute' [,...]) ]

```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_ScaleFit

### **ON clause**

Specifies the table name, view name or query as an InputTable.

### **ScaleMethod**

Specifies the name of the method to be used for Scaling data.

If you specify only one argument value, it applies to all columns specified by the TargetColumns argument. If you specify multiple argument values, each argument value applies to the corresponding input column. For example, the first argument value applies to the first column specified by the TargetColumns argument, the second argument value applies to the second input column, and so on.

Allowed values: [MEAN, SUM, USTD, STD, RANGE, MIDRANGE, MAXABS, RESCALE]

For the RESCALE method, you must specify whether to apply lower\_bound only, upper\_bound only, or both lower\_bound and upper\_bound and corresponding bound values, in the format mentioned in [TD\\_ScaleFit Syntax](#).

### **TargetColumns**

Specifies the name of the InputTable columns for which to output statistics. The columns must contain numeric data in the range (-1e<sup>308</sup>, 1e<sup>308</sup>).

**Note:**

Argument for dense input, disallowed otherwise.

### **AttributeNameColumn**

Specifies the InputTable column that contains the attribute or feature names.

**Note:**

Argument for sparse input, disallowed otherwise.

**AttributeValueColumn**

Specifies the InputTable column that contains the attribute or feature values.

**Note:**

Argument for sparse input, disallowed otherwise.

## Optional Syntax Elements for TD\_ScaleFit

**ON clause**

Specifies the table name, view name or query as a ParameterTable and AttributeTable.

**OutputTable**

Specifies the name of the output table. If this argument is provided, the output is stored in the specified OutputTable name.

**TargetAttributes**

Specifies the target attributes names from AttributeNameColumn which need to be considered for Scaling.

Default: All the attributes from AttributeNameColumn will be considered for Scaling.

**Note:**

Argument for sparse input, disallowed otherwise.

**PartitionColumns**

Specifies the name of the InputTable columns on which to partition the input.

**Note:**

- Column range is not supported for the PartitionColumns argument.
- If the partition column names specified in the PARTITION BY clause contains UNICODE characters, you must specify the PartitionColumns argument.
- Columns specified in the PartitionColumns argument must match exactly with the columns specified in the PARTITION BY clause(s) of ON clause(s) for InputTable, ParameterTable, and AttributeTable.

**MissValue**

Specifies how to process NULL values in input.

Default: KEEP

Option	Description
KEEP	Keep NULL values.
ZERO	Replace each NULL value with 0.
LOCATION	Replace each NULL value with its location value.

### GlobalScale

Specifies whether all input columns are scaled to the same location and scale. If set to false, each input column is scaled separately.

Default: false

### Multiplier

Specifies one or more multiplying factors to apply to the input variables—*multiplier* in the following formula:

$$X' = \text{intercept} + \text{multiplier} * ((X - \text{location})/\text{scale})$$

If you specify only one argument value, it applies to all columns specified by the TargetColumns argument. If you specify multiple argument values, each argument value applies to the corresponding input column. For example, the first argument value applies to the first column specified by the TargetColumns argument, the second argument value applies to the second input column, and so on..

Default: *multiplier* is 1

### Intercept

Specifies one or more addition factors incrementing the scaled results *intercept* in the following formula:

$$X' = \text{intercept} + \text{multiplier} * ((X - \text{location})/\text{scale})$$

The syntax of *intercept* is: [-] {number | min | mean | max }

Where min, mean, and max are the scaled global minimum, maximum, mean values of the corresponding columns.

The formula for computing the scaled global minimum is:

$$\text{scaledmin} = (\text{minX} - \text{location})/\text{scale}$$

The formulas for computing the scaled global mean and maximum are analogous to the preceding formula.

For example, if *intercept* is '- min' and *multiplier* is 1, the formula for computing the scaled result X' from the original value X is:

$$X' = -scaled\min + 1 * ((X - location)/scale)$$

If you specify only one argument value, it applies to all columns specified by the TargetColumns argument. If you specify multiple argument values, each argument value applies to the corresponding input column. For example, the first argument value applies to the first column specified by the TargetColumns argument, the second argument value applies to the second input column, and so on.

Default: *intercept* is 0

#### **IgnoreInvalidLocationScale**

Specifies whether to ignore invalid values of location and scale parameters.

If it is set to false, the function will report an error for invalid values of location and scale parameters.

If it is set to true, the function will ignore invalid values for location and scale parameters and replace them with 0 and 1 respectively.

Default: false

#### **UnusedAttributes**

Specifies whether to emit out unused attributes of different partitions as unscaled values or NULLs. This argument is allowed only when AttributeTable is specified.

Default: UNSCALED

Option	Description
UNSCALED	Emit out unscaled values for unused attributes.
NULIFY	Emit out NULL values for unused attributes.

#### **Note:**

UnusedAttributes argument is applicable only for dense input.

## **TD\_ScaleFit Input**

### **Dense Input**

#### **InputTable Schema**

Column Name	Data Type	Description
PartitionColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, VARCHAR, CHAR	Columns on which to partition the input.

Column Name	Data Type	Description
TargetColumns	NUMERIC	Column to be scaled.

**ParameterTable Schema**

Column Name	Data Type	Description
PartitionColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, VARCHAR, CHAR	Columns on which to partition the input.
PARAMETER_COLUMN	VARCHAR	Name of the parameter.
VALUE_COLUMN	VARCHAR	Value or values of the parameter. Parameter values can be specified as single quoted strings separated by a comma or as unquoted strings separated by a comma. If any parameter value contains a comma, it must be enclosed within single quotes.

**AttributeTable Schema**

Column Name	Data Type	Description
PartitionColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, VARCHAR, CHAR	Columns on which to partition the input.
ATTRIBUTE_COLUMN	VARCHAR	Name of the attribute or feature.

**Sparse Input****InputTable Schema**

Column Name	Data Type	Description
PartitionColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, VARCHAR, CHAR	Columns on which to partition the input.
AttributeNameColumn	VARCHAR	Column containing attribute names.
AttributeValueColumn	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Column containing attribute values.

**ParameterTable Schema - Type 1**

Column Name	Data Type	Description
PartitionColumns	BYTEINT, SMALLINT, INTEGER,	Columns on which to partition the input.

Column Name	Data Type	Description
	BIGINT, VARCHAR, CHAR	
PARAMETER_COLUMN	VARCHAR	Name of the parameter.
VALUE_COLUMN	VARCHAR	<p>Value or values of the parameter.            You can specify the parameter values as single quoted strings separated by comma or as un-quoted strings separated by comma.</p> <p><b>Note:</b></p> <ul style="list-style-type: none"> <li>• If any parameter value contains comma, then it must be enclosed within single quotes.</li> <li>• If multiple parameter values are specified, then maximum number of allowed parameter values is 2048.</li> </ul>

**ParameterTable Schema - Type 2**

Column Name	Data Type	Description
PartitionColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, VARCHAR, CHAR	Columns on which to partition the input.
ATTRIBUTE_COLUMN	VARCHAR	Name of the attribute or feature.
PARAMETER_COLUMN	VARCHAR	Name of the parameter.
VALUE_COLUMN	VARCHAR	Value of the parameter.

**AttributeTable Schema**

Column Name	Data Type	Description
PartitionColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, VARCHAR, CHAR	Columns on which to partition the input.
ATTRIBUTE_COLUMN	VARCHAR	Name of the attribute or feature.

## TD\_ScaleFit Output

### Dense Input

**Output Table Schema**

Column	Data Type	Description
PartitionColumns	Same as InputTable	Columns on which input is partitioned. This appears only when the PartitionColumns argument is specified.
TD_STATTYPE_SCLFIT	VARCHAR (CHARACTER SET LATIN)	Statistic names and parameters—see following table.
TargetColumns	REAL	Statistics values for TargetColumns argument.

### Sparse Input

**Output Table Schema**

Column	Data Type	Description
PartitionColumns	Same as InputTable	Columns on which input is partitioned. This appears only when the PartitionColumns argument is specified.
TD_STATTYPE_SCLFIT	VARCHAR CHARACTER SET LATIN	Statistic names and parameters.
AttributeNameColumn	VARCHAR CHARACTER SET UNICODE	Name of the attributes.
AttributeValueColumn	REAL	Statistics values for the attributes.

Statatypes the function outputs are:

Statistic Name	Description
min	Minimum value in the corresponding column.
max	Maximum value in the corresponding column.
sum	Sum value in the corresponding column.
count	Count of valid values in the corresponding column.
null	Count of NULL values in the corresponding column.
avg	Average of valid values in corresponding column.
variance	Variance of values in corresponding column. Variance is calculated according to N-1 degrees of freedom. (number of valid values minus one).

Statistic Name	Description
ustd	Unbiased Standard deviation of values in corresponding column. Standard deviation is calculated according to N-1 degrees of freedom (number of valid values minus one).
std	Standard deviation of values in corresponding column. Standard deviation is calculated according to N degrees of freedom (number of valid values).
missvalue_*	* is the value of the MissValue argument: KEEP, ZERO, or LOCATION.
globalscale_*	* is the value of the GlobalScale argument : true or false.
unusedattributes_*	* is the value of the UnusedAttributes argument: unscaled or nullify.

The values for location and scale parameters for different ScaleMethods are:

Method	Description	Location	Scale
mean	Mean	Xmean	1
sum	Sum	0	$\Sigma X$
ustd	Unbiased Standard Deviation (Z-Score using Unbiased Standard Deviation)	Xmean	Standard deviation, calculated according to the unbiased estimator of the variance. $\text{ustd} = \sqrt{(\sum(X_i - \text{Xmean})^2) / (N - 1)}$ where N is the count of valid values
std	Standard Deviation (Z-Score using Standard Deviation)	Xmean	Standard deviation, calculated according to the biased estimator of the variance. $\text{std} = \sqrt{(\sum(X_i - \text{Xmean})^2) / N}$ where N is the count of valid values
range	Range	Xmin	Xmax-Xmin
midrange	Midrange	(Xmax+Xmin)/2	(Xmax+Xmin)/2
maxabs	Maximum Absolute Value	0	Maximum of the absolute value of X.
rescale	Rescale using specified lower bound, upper bound, or both.	Lower_bound only: $(\text{Xmin} - \text{lb}^*)$ Upper_bound only: $(\text{Xmax} - \text{ub}^*)$ Lower_bound and Upper_bound: $\text{Xmin} - (\text{lb} / (\text{ub}-\text{lb}))$ *where • lb is lower_bound	Lower_bound only: 1 Upper_bound only: 1 Lower_bound and Upper_bound: $(\text{Xmax} - \text{Xmin}) / (\text{ub}-\text{lb})$

Method	Description	Location	Scale
		• ub is upper_bound	

## Examples: How to Use TD\_ScaleFit

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Dense Input](#)
- [Sparse Input without Partition](#)
- [Sparse Input with Partition](#)

### Dense Input

#### Example: InputTable: scale\_input\_table

```
CREATE TABLE input_table AS
(SELECT * FROM titanic_train WHERE passenger IN (97,488,505,631,873) WITH data;
```

passenger	survived	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked
97	0	1	Goldschmidt, Mr. George B	Male	71	0	0	PC 17754	34. 6542	A5	C
488	0	1	Kent, Mr. Edward Austin	Male	58	0	0	11771	29.7	B37	C
505	1	1	Maioni, Miss, Roberta	Female	16	0	0	110152	86.5	B79	S
631	1	1	Barkworth, Mr. Algernon Henry Wilson	Male	80	0	0	27042	30	A23	S
873	0	1	Carlsson, Mr. Frans Olof	Male	33	0	0	695	5	B51 B53 B55	S

#### TD\_ScaleFit SQL Call

```
SELECT * FROM TD_ScaleFit (
  ON input_table AS InputTable
  OUT PERMANENT TABLE OutputTable (scaleFitOut)
```

```

USING
TargetColumns ('fare')
MissValue ('keep')
ScaleMethod ('range')
GlobalScale ('f')
) AS dt2;

```

### **TD\_ScaleFit Output**

TD_STATTYPE_SCLFIT	Fare
min	5
max	86.5
sum	185.8542
count	5
null	0
avg	37.17084
multiplier	1
intercept	0
location	5
scale	81.5
globalscale_false	NULL
MethodNumberMapping: [0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs, 7:rescale]	4
missvalue_KEEP	NULL

### **Example: InputTable: scale\_input\_partitioned**

```

DROP TABLE scale_input_partitioned;
CREATE multiset TABLE scale_input_partitioned (
pid INTEGER,
passenger INTEGER,
survived INTEGER,
pclass INTEGER,
name VARCHAR(90) CHARACTER SET LATIN NOT CASESPECIFIC,
gender VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC,
age INTEGER,
sibsp INTEGER,
parch INTEGER,

```

```

ticket VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
fare FLOAT,
cabin VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
embarked VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( passenger );

INSERT INTO scale_input_partitioned VALUES (1, 2, 1, 1, 'Cumings; Mrs.
John Bradley (Florence Briggs Thayer)', 'female', 38, 1, 0, 'PC 17599',
7.12833000000000E 001, 'C85', 'C');
INSERT INTO scale_input_partitioned VALUES (1, 4, 1, 1, 'Futrelle;
Mrs. Jacques Heath (Lily May Peel)', 'female', 35, 1, 0, '113803',
5.31000000000000E 001, 'C123', 'S');
INSERT INTO scale_input_partitioned VALUES (1, 7, 0, 1, 'McCarthy;
Mr. Timothy J', 'male', 54, 0, 0, '17463', 5.18625000000000E 001,
'E46', 'S');
INSERT INTO scale_input_partitioned VALUES (1, 11, 1, 3, 'Sandstrom; Miss.
Marguerite Rut', 'female', 4, 1, 1, 'PP 9549', 1.67000000000000E 001,
'G6', 'S');
INSERT INTO scale_input_partitioned VALUES (1, 12, 1, 1, 'Bonnell; Miss.
Elizabeth', 'female', 58, 0, 0, '113783', 2.65500000000000E 001,
'C103', 'S');
INSERT INTO scale_input_partitioned VALUES (2, 22, 1, 2, 'Beesley;
Mr. Lawrence', 'male', 34, 0, 0, '248698', 1.30000000000000E 001,
'D56', 'S');
INSERT INTO scale_input_partitioned VALUES (2, 24, 1, 1, 'Sloper; Mr. William
Thompson', 'male', 28, 0, 0, '113788', 3.55000000000000E 001, 'A6', 'S');
INSERT INTO scale_input_partitioned VALUES (2, 32, 1, 1, 'Spencer; Mrs.
William Augustus (Marie Eugenie)', 'female', NULL, 1, 0, 'PC 17569',
1.46520800000000E 002, 'B78', 'C');
INSERT INTO scale_input_partitioned VALUES (2, 53, 1, 1, 'Harper; Mrs. Henry
Sleeper (Myra Haxton)', 'female', 49, 1, 0, 'PC 17572', 7.67292000000000E
001, 'D33', 'C');
INSERT INTO scale_input_partitioned VALUES (2, 55, 0, 1, 'Ostby; Mr.
Engelhart Cornelius', 'male', 65, 0, 1, '113509', 6.19792000000000E 001,
'B30', 'C');
INSERT INTO scale_input_partitioned VALUES (3, 56, 1, 1, 'Woolner; Mr. Hugh',
'male', NULL, 0, 0, '19947', 3.55000000000000E 001, 'C52', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 63, 0, 1, 'Harris; Mr.
Henry Birkhardt', 'male', 45, 1, 0, '36973', 8.34750000000000E 001,
'C83', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 67, 1, 2, 'Nye; Mrs.
(Elizabeth Ramell)', 'female', 29, 0, 0, 'C.A. 29395', 1.05000000000000E
001, 'F33', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 76, 0, 3, 'Moen; Mr.

```

```

Sigurd Hansen', 'male', 25, 0, 0, '348123', 7.6500000000000E 000, 'F
G73', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 93, 0, 1, 'Chaffee; Mr.
Herbert Fuller', 'male', 46, 1, 0, 'W.E.P. 5734', 6.1175000000000E 001,
'E31', 'S');

DROP TABLE scale_parameters;
CREATE MULTISET TABLE scale_parameters (
pid INTEGER,
parameter_column VARCHAR(150) CHARACTER SET LATIN NOT CASESPECIFIC,
value_column VARCHAR(150) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( pid );
INSERT INTO scale_parameters values(1, 'scalemethod', 'std');
INSERT INTO scale_parameters values(2, 'scalemethod', 'range');

DROP TABLE scale_attributes;
CREATE MULTISET TABLE scale_attributes (
pid INTEGER,
attribute_column VARCHAR(150) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( pid );
INSERT INTO scale_attributes values(1, 'fare');
INSERT INTO scale_attributes values(2, 'age');

```

The following example uses this as input table.

pid	Passenger	Survived	Pclass	Name	Gender	Age	Sibsp	Parch	Ticket	Fare
1	2	1	1	Cumings; Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	7. 1283300000 001
1	4	1	1	Futrelle; Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	5. 3100000000 001
1	7	0	1	McCarthy; Mr. Timothy J	male	54	0	0	17463	5. 1862500000 001
1	11	1	3	Sandstrom; Miss. Marguerite Rut	female	4	1	1	PP 9549	1. 6700000000 001

pid	Passenger	Survived	Pclass	Name	Gender	Age	Sibsp	Parch	Ticket	Fare
1	12	1	1	Bonnell; Miss. Elizabeth	female	58	0	0	113783	2.6550000000001
2	22	1	2	Beesley; Mr. Lawrence	male	34	0	0	248698	1.3000000000001
2	24	1	1	Sloper; Mr William Thompson	male	28	0	0	113788	3.5500000000001
2	32	1	1	Spencer; Mrs. William Augustus (Marie Eugenie)	female	NULL	1	0	PC 17569	1.4652080000002
2	53	1	1	Harper; Mrs. Henry Sleeper (Myna Haxtun)	female	49	1	0	PC 17572	7.6729200000001
2	55	0	1	Ostby; Mr. Engelhart Cornelius	male	65	0	1	113509	6.1979200000001
3	56	1	1	Woolner; Mr. Hugh	male	NULL	0	0	19447	3.5500000000001
3	63	0	1	Harris; Mr. Henry Birkhardt	male	45	1	0	36973	8.3475000000001
3	67	1	2	Nye; Mrs. (Elizabeth Ramell)	female	29	0	0	C.A. 29395	1.0500000000001
3	76	0	3	Moen; Mr. Sigurd Hansen	male	25	0	0	348123	7.6500000000000
3	93	0	1	Chaffee; Mr. Herbert Fuller	male	46	1	0	W.E.P. 5734	6.1175000000001

ParameterTable:

pid	parameter_column	value_column
1	scalemETHOD	midrange

pid	parameter_column	value_column
2	scalemethod	range

AttributeTable:

pid	attribute_column
1	fare
2	age

## TD\_ScaleFit SQL Call

```

SELECT * FROM TD_scaleFit(
ON scale_input_partitioned AS InputTable PARTITION BY pid
ON scale_parameters AS ParameterTable PARTITION BY pid
ON scale_attributes AS AttributeTable PARTITION BY pid
OUT PERMANENT TABLE OutputTable(scaleFitOut_partitioned)
USING
TargetColumns('fare', 'age')

MissValue('zero')
ScaleMethod('maxabs')
globalScale('f')
)AS dt2;

```

## TD\_ScaleFit Output

pid	TD_Stattype_SCLFIT	Fare	Age
1	avg	4.38991600000000E 001	NULL
1	count	5.00000000000000E 000	NULL
1	globalscale_false	NULL	NULL
1	intercept	0.00000000000000E 000	NULL
1	location	4.39916500000000E 001	NULL
1	max	7.12833000000000E 001	NULL
1	min	1.67000000000000E 001	NULL
1	missvalue_KEEP	NULL	NULL
1	multiplier	1.00000000000000E 000	NULL
1	null	0.00000000000000E 000	NULL

pid	TD_Stattype_SCLFIT	Fare	Age
1	scale	2.729165000000000E 001	NULL
1	ScaleMethodNumberMapping: [0:mean, 1:sum,2:ustd,3:std,4:range,5:midrange, 6:maxabs,7:rescale]	5.000000000000000E 000	NULL
1	sum	2.194958000000000E 002	NULL
1	unusedattributes_unscaled	NULL	NULL
2	avg	NULL	4.400000000000000E 001
2	count	NULL	4.000000000000000E 000
2	globalscale_false	NULL	NULL
2	intercept	NULL	NULL
2	location	NULL	2.800000000000000E 001
2	max	NULL	6.500000000000000E 002
2	min	NULL	2.800000000000000E 001
2	missvalue_KEEP	NULL	NULL
2	multiplier	NULL	1.000000000000000E 000
2	null	NULL	1.000000000000000E 000
2	scale	NULL	3.700000000000000E 002
2	ScaleMethodNumberMapping: [0:mean, 1:sum,2:ustd,3:std,4:range,5:midrange, 6:maxabs,7:rescale]	NULL	4.000000000000000E 000
2	sum	NULL	1.760000000000000E 002
2	unusedattributes_unscaled	NULL	NULL
3	avg	3.966000000000000E 001	3.625000000000000E 001
3	count	5.000000000000000E 000	4.000000000000000E 000
3	globalscale_false	NULL	NULL
3	intercept	0.000000000000000E 000	0.000000000000000E 000
3	location	0.000000000000000E 000	0.000000000000000E 000
3	max	8.347500000000000E 001	4.600000000000000E 001
3	min	7.650000000000000E 000	2.500000000000000E 001
3	missvalue_ZERO	NULL	NULL
3	multiplier	1.000000000000000E 000	1.000000000000000E 000

pid	TD_Stattype_SCLFIT	Fare	Age
3	null	0.000000000000000E 000	1.000000000000000E 000
3	scale	8.34750000000000E 001	4.60000000000000E 001
3	ScaleMethodNumberMapping: [0:mean, 1:sum,2:ustd,3:std,4:range,5:midrange, 6:maxabs,7:rescale]	6.00000000000000E 000	6.00000000000000E 000
3	sum	1.98300000000000E 002	1.45000000000000E 002
3	unusedattributes_unscaled	NULL	NULL

## Sparse Input without Partition

The following example assumes the sparse input format for InputTable:

```
create multiset table scale_input_sparse (passenger int, attribute_column
varchar(20), attribute_value real);
insert into scale_input_sparse values (97,      'age',    71);
insert into scale_input_sparse values (97,      'fare',   34.6542);
insert into scale_input_sparse values (488,     'age',    58);
insert into scale_input_sparse values (488,     'fare',   29.7);
insert into scale_input_sparse values (505,     'age',    16);
insert into scale_input_sparse values (505,     'fare',   86.5);
insert into scale_input_sparse values (631,     'age',    80);
insert into scale_input_sparse values (631,     'fare',   30);
insert into scale_input_sparse values (873,     'age',    33);
insert into scale_input_sparse values (873,     'fare',    5);
```

## TD\_ScaleFit Call

```
SELECT * FROM TD_ScaleFit(
ON scale_input_sparse AS InputTable
OUT PERMANENT TABLE OutputTable(sparseScaleFitOut)
USING
TargetAttributes('fare')
AttributeNameColumn('attribute_column')
AttributeValueColumn('attribute_value')
MissValue('Keep')
ScaleMethod('range')
GlobalScale('f')
)AS dt2;
```

## TD\_ScaleFit Output

```

TD_STATTYPE_SCLFIT attribute_column attribute_value
avg                  fare          3.71708400000000E 001
count                fare          5.00000000000000E 000
globalscale_false    fare          NULL
intercept             fare          0.00000000000000E 000
location              fare          5.00000000000000E 000
max                  fare          8.65000000000000E 001
min                  fare          5.00000000000000E 000
missvalue_KEEP       fare          NULL
multiplier            fare          1.00000000000000E 000
null                 fare          0.00000000000000E 000
scale                 fare          8.15000000000000E 001
ScaleMethodNumberMapping:
[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]
                           fare          4.00000000000000E 000
sum                   fare          1.85854200000000E 002

```

## Sparse Input with Partition

The following example assumes the sparse input format for InputTable, ParameterTable, and AttributeTable:

```

CREATE MULTISET TABLE scale_input_part_sparse (pid int, passenger int,
attribute_column varchar(20), attribute_value real);
insert into scale_input_part_sparse values (1, 2, 'age', 38);
insert into scale_input_part_sparse values (1, 2, 'fare', 71.2833);
insert into scale_input_part_sparse values (1, 4, 'age', 35);
insert into scale_input_part_sparse values (1, 4, 'fare', 53.1);
insert into scale_input_part_sparse values (1, 7, 'age', 54);
insert into scale_input_part_sparse values (1, 7, 'fare', 51.8625);
insert into scale_input_part_sparse values (1, 11, 'age', 4);
insert into scale_input_part_sparse values (1, 11, 'fare', 16.7);
insert into scale_input_part_sparse values (1, 12, 'age', 58);
insert into scale_input_part_sparse values (1, 12, 'fare', 26.55);
insert into scale_input_part_sparse values (2, 22, 'age', 34);
insert into scale_input_part_sparse values (2, 22, 'fare', 13);
insert into scale_input_part_sparse values (2, 24, 'age', 28);
insert into scale_input_part_sparse values (2, 24, 'fare', 35.5);
insert into scale_input_part_sparse values (2, 32, 'age', NULL);

```

```

insert into scale_input_part_sparse values (2, 32, 'fare', 146.5208);
insert into scale_input_part_sparse values (2, 53, 'age', 49);
insert into scale_input_part_sparse values (2, 53, 'fare', 76.7292);
insert into scale_input_part_sparse values (2, 55, 'age', 65);
insert into scale_input_part_sparse values (2, 55, 'fare', 61.9792);
insert into scale_input_part_sparse values (3, 56, 'age', NULL);
insert into scale_input_part_sparse values (3, 56, 'fare', 35.5);
insert into scale_input_part_sparse values (3, 63, 'age', 45);
insert into scale_input_part_sparse values (3, 63, 'fare', 83.475);
insert into scale_input_part_sparse values (3, 67, 'age', 29);
insert into scale_input_part_sparse values (3, 67, 'fare', 10.5);
insert into scale_input_part_sparse values (3, 76, 'age', 25);
insert into scale_input_part_sparse values (3, 76, 'fare', 7.65);
insert into scale_input_part_sparse values (3, 93, 'age', 46);
insert into scale_input_part_sparse values (3, 93, 'fare', 61.175);

CREATE MULTISET TABLE sparse_scale_parameters (
pid INTEGER,
parameter_column VARCHAR(150) CHARACTER SET UNICODE NOT CASESPECIFIC,
value_column VARCHAR(150) CHARACTER SET UNICODE NOT CASESPECIFIC)
PRIMARY INDEX ( pid );
insert into sparse_scale_parameters values(1, 'scalemethod', 'midrange');
insert into sparse_scale_parameters values(2, 'scalemethod', 'range');

CREATE MULTISET TABLE sparse_scale_attributes (
pid INTEGER,
attribute_column VARCHAR(150) CHARACTER SET UNICODE NOT CASESPECIFIC)
PRIMARY INDEX ( pid );
insert into sparse_scale_attributes values(1, 'fare');
insert into sparse_scale_attributes values(2, 'age');

```

## TD\_ScaleFit Call

```

SELECT * FROM TD_ScaleFit(
ON scale_input_part_sparse AS InputTable PARTITION BY pid
ON sparse_scale_parameters AS ParameterTable PARTITION BY pid
ON sparse_scale_attributes AS AttributeTable PARTITION BY pid
OUT PERMANENT TABLE OutputTable(sparseScaleFitOutPartitioned)
USING
AttributeNameColumn('attribute_column')
AttributeValueColumn('attribute_value')
MissValue('zero')
ScaleMethod('maxabs')

```

```
globalScale('f')
)AS dt2 order by 1,3,2;
```

## TD\_ScaleFit Output

pid	TD_STATTYPE_SCLFIT	attribute_column	attribute_value
1	avg	fare	4.38991600000000E 001
1	count	fare	5.00000000000000E 000
1	globalscale_false	fare	NULL
1	intercept	fare	0.00000000000000E 000
1	location	fare	4.39916500000000E 001
1	max	fare	7.12833000000000E 001
1	min	fare	1.67000000000000E 001
1	missvalue_ZERO	fare	NULL
1	multiplier	fare	1.00000000000000E 000
1	null	fare	0.00000000000000E 000
1	scale	fare	2.72916500000000E 001
1	ScaleMethodNumberMapping:		
	[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]		
		fare	5.00000000000000E 000
1	sum	fare	2.19495800000000E 002
2	avg	age	4.40000000000000E 001
2	count	age	4.00000000000000E 000
2	globalscale_false	age	NULL
2	intercept	age	0.00000000000000E 000
2	location	age	2.80000000000000E 001
2	max	age	6.50000000000000E 001
2	min	age	2.80000000000000E 001
2	missvalue_ZERO	age	NULL
2	multiplier	age	1.00000000000000E 000
2	null	age	1.00000000000000E 000
2	scale	age	3.70000000000000E 001
2	ScaleMethodNumberMapping:		
	[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]		
		age	4.00000000000000E 000
2	sum	age	1.76000000000000E 002
3	avg	age	3.62500000000000E 001
3	count	age	4.00000000000000E 000
3	globalscale_false	age	NULL
3	intercept	age	0.00000000000000E 000
3	location	age	0.00000000000000E 000
3	max	age	4.60000000000000E 001

```

3 min age 2.5000000000000E 001
3 missvalue_ZERO age NULL
3 multiplier age 1.0000000000000E 000
3 null age 1.0000000000000E 000
3 scale age 4.6000000000000E 001
3 ScaleMethodNumberMapping:
[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]
                    age 6.0000000000000E 000
3 sum age 1.4500000000000E 002
3 avg fare 3.9660000000000E 001
3 count fare 5.0000000000000E 000
3 globalscale_false fare NULL
3 intercept fare 0.0000000000000E 000
3 location fare 0.0000000000000E 000
3 max fare 8.3475000000000E 001
3 min fare 7.6500000000000E 000
3 missvalue_ZERO fare NULL
3 multiplier fare 1.0000000000000E 000
3 null fare 0.0000000000000E 000
3 scale fare 8.3475000000000E 001
3 ScaleMethodNumberMapping:
[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]
                    fare 6.0000000000000E 000
3 sum fare 1.9830000000000E 002

```

## TD\_ScaleTransform

TD\_ScaleTransform scales specified input table columns, using [TD\\_ScaleFit](#) output. TD\_ScaleTransform accepts input data in dense and sparse format.

ScaleFitTransform is a data preprocessing technique that applies various scaling methods to transform the variables in a dataset to have a specific scale or range of values suitable for analysis or modeling. The technique involves fitting the scaler on a training dataset and then using the same scaler to transform the variables in the test dataset.

The ScaleFitTransform technique involves two main steps:

1. Fit the scaler: The scaler is fitted on the training dataset to learn the scaling parameters, such as the minimum and maximum values or mean and standard deviation, depending on the chosen scaling method.
2. Transform the data: Use the fitted scaler to transform the variables in both the training and test datasets. This ensures that the same scaling is applied to both datasets, preventing any bias or inconsistencies in the analysis or modeling.

Some commonly used scaling methods in ScaleFitTransform include Min-Max scaling, Standardization, Logarithmic scaling, Power transformation, and Robust scaling. The choice of scaling method depends on the nature of the data and the analysis or modeling goals.

Overall, ScaleFitTransform is an essential preprocessing step that can improve the performance of machine learning models and reduce bias in statistical analysis

## Usage Considerations

The following are usage considerations for TD\_ScaleTransform function:

ON clause

- The InputTable in the TD\_ScaleTransform query can have no partition at all or have the following combinations of PARTITION BY/ORDER BY clauses.
  - PARTITION BY ANY ORDER BY
  - PARTITION BY ANY
  - PARTITION BY KEY ORDER BY
  - PARTITION BY KEY
- FitTable must be a DIMENSION table with an optional ORDER BY clause or can have PARTITION BY KEY.
- None of the arguments are mandatory for TD\_ScaleTransform function.
- If the InputTable uses PARTITION BY ANY syntax, but FitTable is not specified as DIMENSION, then an error is reported. For example: \*\*\* Failure 9850 Invalid input to table operator: There can be only one input with no partitioning attributes / PARTITION BY ANY/ LOCAL ORDER BY clause.

## Limits and Restrictions

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs) and KanjiSJIS or Graphic data types.

## Function Information

- [TD\\_ScaleTransform Syntax](#)
- [Required Syntax Elements for TD\\_ScaleTransform](#)
- [Optional Syntax Elements for TD\\_ScaleTransform](#)
- [TD\\_ScaleTransform Input](#)
- [TD\\_ScaleTransform Output](#)
- [Examples: How to Use TD\\_ScaleTransform](#)

## TD\_ScaleTransform Syntax

### Dense Input

Syntax without partition:

```
TD_ScaleTransform (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS FitTable DIMENSION
    [ USING
        Accumulate ({ 'accumulate_column' | 'accumulate_column_range' }[,...])
    ]
)
```

Syntax with partition:

```
TD_ScaleTransform (
    ON { table | view | (query) } AS InputTable PARTITION BY partition_column [,...]
    ON { table | view | (query) } AS FitTable PARTITION BY partition_column [,...]
    [ USING
        Accumulate ({ 'accumulate_column' | 'accumulate_column_range' }[,...])
    ]
)
```

## Sparse Input

Syntax without partition

```
TD_ScaleTransform (
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS FitTable DIMENSION
    USING
        AttributeNameColumn ('attribute_name_column')
        AttributeValueColumn ('attribute_value_column')
        [ Accumulate ({ 'accumulate_column' | 'accumulate_column_range' }[,...])
    ]
)
```

Syntax with partition

```
TD_ScaleTransform (
    ON { table | view | (query) } AS InputTable PARTITION BY partition_column [,...]
    ON { table | view | (query) } AS FitTable PARTITION BY partition_column [,...]
    USING
        AttributeNameColumn ('attribute_name_column')
        AttributeValueColumn ('attribute_value_column')
        [ Accumulate ({ 'accumulate_column' | 'accumulate_column_range' }[,...])
    ]
)
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_ScaleTransform

**ON clause**

Specifies the table name, view name or query as an InputTable and FitTable.

---

**Note:**

Applicable for dense and sparse input format.

---

**AttributeNameColumn**

Specifies the attribute or features names.

---

**Note:**

Argument for sparse input, disallowed otherwise.

---

**AttributeValueColumn**

Specifies the attribute or features values.

---

**Note:**

Argument for sparse input, disallowed otherwise.

---

## Optional Syntax Elements for TD\_ScaleTransform

**Accumulate**

Specifies the InputTable columns to copy to the output table.

---

## TD\_ScaleTransform Input

### Dense Input

#### InputTable Schema

Column Name	Data Type	Description
PartitionColumns	Any	Columns on which to partition the input.
TargetColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Column to be scaled.

#### FitTable Schema

Column	Data Type	Description
PartitionColumns	Same as InputTable	Columns on which input is partitioned. This appears only when the PartitionColumns argument is specified.
TD_STATTYPE_SCLFIT	VARCHAR CHARACTER SET LATIN	Statistics name and parameter values.
TargetColumns	REAL	Statistics values for TargetColumns argument.

### Sparse Input

#### InputTable Schema

Column Name	Data Type	Description
PartitionColumns	Any	Columns on which to partition the input.
AttributeNameColumn	VARCHAR	Column containing attribute names.
AttributeValueColumn	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	Column containing attribute values.

#### FitTable Schema

Column	Data Type	Description
PartitionColumns	Same as InputTable	Columns on which input is partitioned. This appears only when the PartitionColumns argument is specified.
TD_STATTYPE_SCLFIT	VARCHAR CHARACTER SET LATIN	Statistic names and parameters.

Column	Data Type	Description
AttributeNameColumn	VARCHAR CHARACTER SET UNICODE	Name of the Attributes.
AttributeValueColumn	REAL	Statistics values for the attributes.

## TD\_ScaleTransform Output

### Dense Input

Output Table Schema

Column	Data Type	Description
AccumulateColumns	Any	Columns copied from input to output.
TargetColumns	REAL	Columns containing scaled values.

### Sparse Input

Output Table Schema

Column Name	Data Type	Description
AccumulateColumns	Any	Columns copied from input to output.
AttributeNameColumn	VARCHAR	Name of the attributes.
AttributeValueColumn	REAL	Scaled values for the attributes.

## Examples: How to Use TD\_ScaleTransform

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Dense Input](#)
- [Sparse Input without Partition](#)
- [Sparse Input with Partition](#)

### Dense Input

#### Example: InputTable: scale\_input\_table

```
CREATE TABLE input_table AS
(SELECT * FROM titanic_train WHERE passenger IN (97,488,505,631,873) WITH data:
```

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
97	0	1	Goldschmidt, Mr. George B	Male	71	0	0	PC 17754	34.6542	A5	C
488	0	1	Kent, Mr. Edward Austin	Male	58	0	0	11771	29.7	B37	C
505	1	1	Maioni, Miss. Roberta	Female	16	0	0	110152	86.5	B79	S
631	1	1	Barkworth, Mr. Algernon Henry Wilson	Male	80	0	0	27042	30	A23	S
873	0	1	Carlsson, Mr. Frans Olof	Male	33	0	0	695	5	B51 B53 B55	S

### FitTable (Generated using TD\_ScaleFit function)

TD_STATTYPE_SCLFIT	fare
Min	5
Max	86.5
Sum	185.8542
Count	5
Null	0
Avg	37.17084
Multiplier	1
Intercept	0
Location	5
Scale	81.5
methodNumberMapping: [0:mean, 1:sum, 3:std, 4:range, 5:midrange, 6:maxabs, 7:rescale]	4
globalscale_false	NULL
Missvalue_KEEP	NULL

### TD\_ScaleTransform SQL Call

```
SELECT * FROM TD_scaleTransform (
    ON scale_input_table AS InputTable
```

```

ON scaleFitOut AS FitTable DIMENSION
USING
Accumulate ('passenger')
) AS dt2 ORDER BY 1;

```

### **TD\_ScaleTransform Output**

passenger	fare
97	0.363855214723926
488	0.303067484662577
505	1
631	0.306748466257669
873	0

### **Example: InputTable: scale\_input\_partitioned**

```

DROP TABLE scale_input_partitioned;
CREATE multiset TABLE scale_input_partitioned (
pid INTEGER,
passenger INTEGER,
survived INTEGER,
pclass INTEGER,
name VARCHAR(90) CHARACTER SET LATIN NOT CASESPECIFIC,
gender VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC,
age INTEGER,
sibsp INTEGER,
parch INTEGER,
ticket VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
fare FLOAT,
cabin VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
embarked VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( passenger );

INSERT INTO scale_input_partitioned VALUES (1, 2, 1, 1, 'Cumings; Mrs.
John Bradley (Florence Briggs Thayer)', 'female', 38, 1, 0, 'PC 17599',
7.1283300000000E 001, 'C85', 'C');

INSERT INTO scale_input_partitioned VALUES (1, 4, 1, 1, 'Futrelle;
Mrs. Jacques Heath (Lily May Peel)', 'female', 35, 1, 0, '113803',
5.3100000000000E 001, 'C123', 'S');

INSERT INTO scale_input_partitioned VALUES (1, 7, 0, 1, 'McCarthy;
Mr. Timothy J', 'male', 54, 0, 0, '17463', 5.18625000000000E 001,

```

```
'E46', 'S');
INSERT INTO scale_input_partitioned VALUES (1, 11, 1, 3, 'Sandstrom; Miss.
Marguerite Rut', 'female', 4, 1, 1, 'PP 9549', 1.67000000000000E 001,
'G6', 'S');
INSERT INTO scale_input_partitioned VALUES (1, 12, 1, 1, 'Bonnell; Miss.
Elizabeth', 'female', 58, 0, 0, '113783', 2.65500000000000E 001,
'C103', 'S');
INSERT INTO scale_input_partitioned VALUES (2, 22, 1, 2, 'Beesley;
Mr. Lawrence', 'male', 34, 0, 0, '248698', 1.30000000000000E 001,
'D56', 'S');
INSERT INTO scale_input_partitioned VALUES (2, 24, 1, 1, 'Sloper; Mr. William
Thompson', 'male', 28, 0, 0, '113788', 3.55000000000000E 001, 'A6', 'S');
INSERT INTO scale_input_partitioned VALUES (2, 32, 1, 1, 'Spencer; Mrs.
William Augustus (Marie Eugenie)', 'female', NULL, 1, 0, 'PC 17569',
1.46520800000000E 002, 'B78', 'C');
INSERT INTO scale_input_partitioned VALUES (2, 53, 1, 1, 'Harper; Mrs. Henry
Sleeper (Myra Haxton)', 'female', 49, 1, 0, 'PC 17572', 7.67292000000000E
001, 'D33', 'C');
INSERT INTO scale_input_partitioned VALUES (2, 55, 0, 1, 'Ostby; Mr.
Engelhart Cornelius', 'male', 65, 0, 1, '113509', 6.19792000000000E 001,
'B30', 'C');
INSERT INTO scale_input_partitioned VALUES (3, 56, 1, 1, 'Woolner; Mr. Hugh',
'male', NULL, 0, 0, '19947', 3.55000000000000E 001, 'C52', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 63, 0, 1, 'Harris; Mr.
Henry Birkhardt', 'male', 45, 1, 0, '36973', 8.34750000000000E 001,
'C83', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 67, 1, 2, 'Nye; Mrs.
(Elizabeth Ramell)', 'female', 29, 0, 0, 'C.A. 29395', 1.05000000000000E
001, 'F33', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 76, 0, 3, 'Moen; Mr.
Sigurd Hansen', 'male', 25, 0, 0, '348123', 7.65000000000000E 000, 'F
G73', 'S');
INSERT INTO scale_input_partitioned VALUES (3, 93, 0, 1, 'Chaffee; Mr.
Herbert Fuller', 'male', 46, 1, 0, 'W.E.P. 5734', 6.11750000000000E 001,
'E31', 'S');
```

The following example uses this as input table.

pid	Passenger	Survived	Pclass	Name	Gender	Age	Sibsp	Parch	Ticket	Fare	Class
1	2	1	1	Cumings; Mrs. John Bradley (Florence)	female	38	1	0	PC 17599	7. 12833	C

pid	Passenger	Survived	Pclass	Name	Gender	Age	Sibsp	Parch	Ticket	Fare	Class
				Briggs Thayer)							
1	4	1	1	Futrelle; Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	5.31	C
1	7	0	1	McCarthy; Mr. Timothy J	male	54	0	0	17463	5.18625	E4
1	11	1	3	Sandstrom; Miss. Marguerite Rut	female	4	1	1	PP 9549	1.67	G
1	12	1	1	Bonnell; Miss. Elizabeth	female	58	0	0	113783	26.55	C
2	22	1	2	Beesley; Mr. Lawrence	male	34	0	0	248698	13.	D3
2	24	1	1	Sloper; Mr William Thompson	male	28	0	0	113788	35.5	A6
2	32	1	1	Spencer; Mrs. William Augustus (Marie Eugenie)	female	NULL	1	0	PC 17569	146.5208	B7
2	53	1	1	Harper; Mrs. Henry Sleeper (Myra Haxtun)	female	49	1	0	PC 17572	76.7292	D3
2	55	0	1	Ostby; Mr. Engelhart Cornelius	male	65	0	1	113509	61.9792	B3
3	56	1	1	Woolner; Mr. Hugh	male	NULL	0	0	19447	35.5	C5
3	63	0	1	Harris; Mr. Henry Birkhardt	male	45	1	0	36973	83.475	C8

pid	Passenger	Survived	Pclass	Name	Gender	Age	Sibsp	Parch	Ticket	Fare	Ca
3	67	1	2	Nye; Mrs. (Elizabeth Ramell)	female	29	0	0	C.A. 29395	10.5	F3
3	76	0	3	Moen; Mr. Sigurd Hansen	male	25	0	0	348123	7.65	F
3	93	0	1	Chaffee; Mr. Herbert Fuller	male	46	1	0	W.E.P. 5734	61. 175	E3

**FitTable (generated by TD\_ScaleFit)**

pid	TD_Stattype_SCLFIT	Fare	Age
1	avg	4.38991600000000E 001	NULL
1	count	5.00000000000000E 000	NULL
1	globalscale_false	NULL	NULL
1	intercept	0.00000000000000E 000	NULL
1	location	4.39916500000000E 001	NULL
1	max	7.12833000000000E 001	NULL
1	min	1.67000000000000E 001	NULL
1	missvalue_KEEP	NULL	NULL
1	multiplier	1.00000000000000E 000	NULL
1	null	0.00000000000000E 000	NULL
1	scale	2.72916500000000E 001	NULL
1	ScaleMethodNumberMapping: [0:mean, 1:sum,2:ustd,3:std,4:range,5:midrange, 6:maxabs,7:rescale]	5.00000000000000E 000	NULL
1	sum	2.19495800000000E 002	NULL
1	unusedattributes_unscaled	NULL	NULL
2	avg	NULL	4.40000000000000E 001
2	count	NULL	4.00000000000000E 000
2	globalscale_false	NULL	NULL
2	intercept	NULL	0.00000000000000E 000
2	location	NULL	2.80000000000000E 001

pid	TD_Stattype_SCLFIT	Fare	Age
2	max	NULL	6.50000000000000E 001
2	min	NULL	2.80000000000000E 001
2	missvalue_KEEP	NULL	NULL
2	multiplier	NULL	1.00000000000000E 000
2	null	NULL	1.00000000000000E 000
2	scale	NULL	3.70000000000000E 001
2	ScaleMethodNumberMapping: [0:mean, 1:sum,2:ustd,3:std,4:range,5:midrange, 6:maxabs,7:rescale]	NULL	4.00000000000000E 000
2	sum	NULL	1.76000000000000E 002
2	unusedattributes_unscaled	NULL	NULL
3	avg	3.96600000000000E 001	3.62500000000000E 001
3	count	5.00000000000000E 000	4.00000000000000E 000
3	globalscale_false	NULL	NULL
3	intercept	0.00000000000000E 000	0.00000000000000E 000
3	location	0.00000000000000E 000	0.00000000000000E 000
3	max	8.34750000000000E 001	4.60000000000000E 001
3	min	7.65000000000000E 000	2.50000000000000E 001
3	missvalue_ZERO	NULL	NULL
3	multiplier	1.00000000000000E 000	1.00000000000000E 000
3	null	0.00000000000000E 000	1.00000000000000E 000
3	scale	8.34750000000000E 001	4.60000000000000E 001
3	ScaleMethodNumberMapping: [0:mean, 1:sum,2:ustd,3:std,4:range,5:midrange, 6:maxabs,7:rescale]	6.00000000000000E 000	6.00000000000000E 000
3	sum	1.98300000000000E 002	1.45000000000000E 002
3	unusedattributes_unscaled	NULL	NULL

### TD\_ScaleTransform SQL Call

```
SELECT * FROM TD_scaleTransform(
ON scale_input_partitioned AS InputTable PARTITION BY pid
```

```
ON scaleFitOut_partitioned AS FitTable PARTITION BY pid
USING
Accumulate('pid', 'passenger')
)AS dt2;
```

## TD\_ScaleTransform Output

pid	Passenger	Fare	Age
1	2	1.00000000000000E 000	3.80000000000000E 001
1	4	3.33741272513754E-001	3.50000000000000E 001
1	7	2.88397733372662E-001	5.40000000000000E 001
1	11	-1.00000000000000E 000	4.00000000000000E 001
1	12	-6.39083749058778E-001	5.80000000000000E 001
2	22	1.30000000000000E 001	1.62162162162162E-001
2	24	3.55000000000000E 001	0.00000000000000E 000
2	32	1.46520800000000E 002	-7.56756756756757E-001
2	53	7.67292000000000E 001	5.67567567567568E-001
2	55	6.19792000000000E 001	1.00000000000000E 000
3	56	4.25277029050614E-001	0.00000000000000E 000
3	63	1.00000000000000E 000	9.78260869565217E-001
3	67	1.25786163522013E-001	6.30434782608696E-001
3	76	9.16442048517520E-002	5.43478260869565E-001
3	93	7.32854147948488E-001	1.00000000000000E 000

## Sparse Input without Partition

The following example assumes sparse input format for InputTable:

```
create multiset table scale_input_sparse (passenger int, attribute_column
varchar(20), attribute_value real);
insert into scale_input_sparse values (97,      'age',    71);
insert into scale_input_sparse values (97,      'fare',   34.6542);
insert into scale_input_sparse values (488,      'age',    58);
insert into scale_input_sparse values (488,      'fare',   29.7);
insert into scale_input_sparse values (505,      'age',    16);
insert into scale_input_sparse values (505,      'fare',  86.5);
```

```
insert into scale_input_sparse values (631, 'age', 80);
insert into scale_input_sparse values (631, 'fare', 30);
insert into scale_input_sparse values (873, 'age', 33);
insert into scale_input_sparse values (873, 'fare', 5);
```

### FitTable Generated with TD\_ScaleFit Function

TD_STATTYPE_SCLFIT	attribute_column	attribute_value
avg	fare	3.71708400000000E 001
count	fare	5.00000000000000E 000
globalscale_false	fare	NULL
intercept	fare	0.00000000000000E 000
location	fare	5.00000000000000E 000
max	fare	8.65000000000000E 001
min	fare	5.00000000000000E 000
missvalue_KEEP	fare	NULL
multiplier	fare	1.00000000000000E 000
null	fare	0.00000000000000E 000
scale	fare	8.15000000000000E 001
ScaleMethodNumberMapping:		
[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]		
	fare	4.00000000000000E 000
sum	fare	1.85854200000000E 002

### TD\_ScaleTransform SQL Call

```
SELECT * FROM TD_ScaleTransform(
    ON scale_input_sparse AS InputTable
    ON sparseScaleFitOut AS FitTable DIMENSION
    USING
        AttributeNameColumn('attribute_column')
        AttributeValueColumn('attribute_value')
        Accumulate('passenger')
    )as dt2 order by 1,2;
```

### TD\_ScaleTransform Output

passenger	attribute_column	attribute_value
97	fare	0.363855214723926
488	fare	0.303067484662577

505	fare	1.00000000000000E 000
631	fare	0.306748466257669
873	fare	0.00000000000000E 000

## Sparse Input with Partition

The following example assumes sparse input format for InputTable:

```
CREATE MULTISET TABLE scale_input_part_sparse (pid int, passenger int,
attribute_column varchar(20), attribute_value real);
insert into scale_input_part_sparse values (1, 2, 'age', 38);
insert into scale_input_part_sparse values (1, 2, 'fare', 71.2833);
insert into scale_input_part_sparse values (1, 4, 'age', 35);
insert into scale_input_part_sparse values (1, 4, 'fare', 53.1);
insert into scale_input_part_sparse values (1, 7, 'age', 54);
insert into scale_input_part_sparse values (1, 7, 'fare', 51.8625);
insert into scale_input_part_sparse values (1, 11, 'age', 4);
insert into scale_input_part_sparse values (1, 11, 'fare', 16.7);
insert into scale_input_part_sparse values (1, 12, 'age', 58);
insert into scale_input_part_sparse values (1, 12, 'fare', 26.55);
insert into scale_input_part_sparse values (2, 22, 'age', 34);
insert into scale_input_part_sparse values (2, 22, 'fare', 13);
insert into scale_input_part_sparse values (2, 24, 'age', 28);
insert into scale_input_part_sparse values (2, 24, 'fare', 35.5);
insert into scale_input_part_sparse values (2, 32, 'age', NULL);
insert into scale_input_part_sparse values (2, 32, 'fare', 146.5208);
insert into scale_input_part_sparse values (2, 53, 'age', 49);
insert into scale_input_part_sparse values (2, 53, 'fare', 76.7292);
insert into scale_input_part_sparse values (2, 55, 'age', 65);
insert into scale_input_part_sparse values (2, 55, 'fare', 61.9792);
insert into scale_input_part_sparse values (3, 56, 'age', NULL);
insert into scale_input_part_sparse values (3, 56, 'fare', 35.5);
insert into scale_input_part_sparse values (3, 63, 'age', 45);
insert into scale_input_part_sparse values (3, 63, 'fare', 83.475);
insert into scale_input_part_sparse values (3, 67, 'age', 29);
insert into scale_input_part_sparse values (3, 67, 'fare', 10.5);
insert into scale_input_part_sparse values (3, 76, 'age', 25);
insert into scale_input_part_sparse values (3, 76, 'fare', 7.65);
insert into scale_input_part_sparse values (3, 93, 'age', 46);
insert into scale_input_part_sparse values (3, 93, 'fare', 61.175);
```

**FitTable Generated with TD\_ScaleFit Function**

pid	TD_STATTYPE_SCLFIT	attribute_column	attribute_value
1	avg	fare	4.38991600000000E 001
1	count	fare	5.00000000000000E 000
1	globalscale_false	fare	NULL
1	intercept	fare	0.00000000000000E 000
1	location	fare	4.39916500000000E 001
1	max	fare	7.12833000000000E 001
1	min	fare	1.67000000000000E 001
1	missvalue_ZERO	fare	NULL
1	multiplier	fare	1.00000000000000E 000
1	null	fare	0.00000000000000E 000
1	scale	fare	2.72916500000000E 001
1	ScaleMethodNumberMapping:		
	[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]		
		fare	5.00000000000000E 000
1	sum	fare	2.19495800000000E 002
2	avg	age	4.40000000000000E 001
2	count	age	4.00000000000000E 000
2	globalscale_false	age	NULL
2	intercept	age	0.00000000000000E 000
2	location	age	2.80000000000000E 001
2	max	age	6.50000000000000E 001
2	min	age	2.80000000000000E 001
2	missvalue_ZERO	age	NULL
2	multiplier	age	1.00000000000000E 000
2	null	age	1.00000000000000E 000
2	scale	age	3.70000000000000E 001
2	ScaleMethodNumberMapping:		
	[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]		
		age	4.00000000000000E 000
2	sum	age	1.76000000000000E 002
3	avg	age	3.62500000000000E 001
3	count	age	4.00000000000000E 000
3	globalscale_false	age	NULL
3	intercept	age	0.00000000000000E 000
3	location	age	0.00000000000000E 000
3	max	age	4.60000000000000E 001
3	min	age	2.50000000000000E 001
3	missvalue_ZERO	age	NULL

```

3   multiplier      age      1.0000000000000E 000
3   null           age      1.0000000000000E 000
3   scale          age      4.6000000000000E 001
3   ScaleMethodNumberMapping:
[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]
                    age      6.0000000000000E 000
3   sum            age      1.4500000000000E 002
3   avg             fare    3.9660000000000E 001
3   count           fare    5.0000000000000E 000
3   globalscale_false fare    NULL
3   intercept       fare    0.0000000000000E 000
3   location        fare    0.0000000000000E 000
3   max              fare    8.3475000000000E 001
3   min              fare    7.6500000000000E 000
3   missvalue_ZERO  fare    NULL
3   multiplier      fare    1.0000000000000E 000
3   null            fare    0.0000000000000E 000
3   scale           fare    8.3475000000000E 001
3   ScaleMethodNumberMapping:
[0:mean,1:sum,2:ustd,3:std,4:range,5:midrange,6:maxabs,7:rescale]
                    fare    6.0000000000000E 000
3   sum            fare    1.9830000000000E 002

```

## TD\_ScaleTransform SQL Call

```

SELECT * FROM TD_ScaleTransform(
ON scale_input_part_sparse AS InputTable PARTITION BY pid
ON sparseScaleFitOutPartitioned AS FitTable PARTITION BY pid
USING
AttributeNameColumn('attribute_column')
AttributeValueColumn('attribute_value')
Accumulate('pid','passenger')
)AS dt2 order by 1,2,3;

```

## TD\_ScaleTransform Output

pid	passenger	attribute_column	attribute_value
1	2	fare	1.0000000000000E 000
1	4	fare	3.33741272513754E-01
1	7	fare	2.88397733372662E-01
1	11	fare	-1.0000000000000E 000
1	12	fare	-6.39083749058778E-01

2	22	age	1.62162162162162E-01
2	24	age	0.00000000000000E 000
2	32	age	-7.56756756756757E-01
2	53	age	5.67567567567568E-01
2	55	age	1.00000000000000E 000
3	56	age	0.00000000000000E 000
3	56	fare	4.25277029050614E-01
3	63	age	9.78260869565217E-01
3	63	fare	1.00000000000000E 000
3	67	age	6.30434782608696E-01
3	67	fare	1.25786163522013E-01
3	76	age	5.43478260869565E-01
3	76	fare	9.16442048517520E-02
3	93	age	1.00000000000000E 000
3	93	fare	7.32854147948488E-01

## TD\_TargetEncodingFit

TargetEncoding generally uses the likelihood or expected value of the target variable for each category and encodes that category with that value. This technique works for both binary classification and regression and for multiclass classification a similar technique is applied, which encodes the categorical variable with k new variables, where k is the number of classes.

The TD\_TargetEncodingFit function takes the InputTable and a CategoricalTable as input and generates the required hyperparameters, which will be used by the [TD\\_TargetEncodingTransform](#) function for encoding the categorical values.

---

### Note:

- This function requires the UTF8 client character set.
- This function does not support Pass-Through Characters (PTCs).
- This function does not support KanjiSJIS or Graphic data types.
- The maximum number of unique categories in the particular column is 4000.
- The maximum category length is 128 characters.
- Columns with a large number of distinct categories can have an impact on query execution time.

---

### Function Information

- [TD\\_TargetEncodingFit Syntax](#)
- [TD\\_TargetEncodingFit Syntax Elements](#)
- [TD\\_TargetEncodingFit Input](#)
- [TD\\_TargetEncodingFit Output](#)
- [TD\\_TargetEncodingFit Usage Notes](#)

- [TD\\_TargetEncodingFit Examples](#)

## TD\_TargetEncodingFit Syntax

```
TD_TargetEncodingFit (
    ON {table | view | (query)} AS InputTable
    ON {table | view | (query)} AS CategoryTable DIMENSION
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table_name) ]
    USING
        EncoderMethod({'CBM_BETA' | 'CBM_DIRICHLET' | 'CBM_GAUSSIAN_INVERSE_GAMMA'})
        TargetColumns ({'target_column' | 'target_column_range'} [, ...])
        ResponseColumn ('response_column')

    [AlphaPrior(alpha_prior_value)]
    [BetaPrior(beta_prior_value)]

    [AlphaPriors(list_of_values)]
    [NumDistinctResponses(num_distinct_responses)]

    [U0Prior(u0_prior_value)]
    [V0Prior(v0_prior_value)]
    [Alpha0Prior(alpha0_prior_value)]
    [Beta0Prior(beta0_prior_value)]

    [DefaultValues ({ default_value | default_value_i, ... })]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## TD\_TargetEncodingFit Syntax Elements

### EncoderMethod

[Required]

- If the response variable is following a binary classification, for example, values are either 0 or 1, use EncoderMethod as CBM\_BETA.

- If the response variable is following a multi-class classification, for example. values are (1,...,k, where k is the number of classes), use EncoderMethod as CBM\_DIRICHLET.
- If the response variable is following a regression, for example, values are contiguous numeric values, use EncoderMethod as CBM\_GAUSSIAN\_INVERSE\_GAMMA.

The maximum length supported is 128. Not case sensitive.

Tokens are CBM\_BETA, CBM\_DIRICHLET, and CBM\_GAUSSIAN\_INVERSE\_GAMMA.

#### **TargetColumns**

[Required] Specify the column from the InputTable that contains the categorical values to be encoded. The maximum length supported is 128. The maximum list length is 2018. Not case sensitive.

#### **ResponseColumn**

[Required] Specify column from the InputTable that contains the response values. The maximum length supported is 128. Not case sensitive.

#### **AlphaPrior**

[Optional] Specify the prior parameter of the CBM\_BETA encoder method.

#### **BetaPrior**

[Optional] Specify the prior parameter of the CBM\_BETA encoder method.

#### **AlphaPriors**

[Optional] Specify the prior parameter of the CBM\_DIRICHLET encoder method. The number of values specified in this argument must be equal to the NumDistinctResponses value. The maximum list length is 2018.

#### **NumDistinctResponses**

Required if EncoderMethod is 'CBM\_DIRICHLET', otherwise optional. Specify the number of distinct values present in ResponseColumn.

#### **U0Prior**

[Optional] Specify the prior parameter of the CBM\_GAUSSIAN\_INVERSE\_GAMMA encoder method.

#### **V0Prior**

[Optional] Specify the prior parameter of the CBM\_GAUSSIAN\_INVERSE\_GAMMA encoder method.

**Alpha0Prior**

[Optional] Specify the prior parameter of the CBM\_GAUSSIAN\_INVERSE\_GAMMA encoder method.

**Beta0Prior**

[Optional] Specify the prior parameter of the CBM\_GAUSSIAN\_INVERSE\_GAMMA encoder method.

**DefaultValues**

[Optional] Specify the values to use when the category is not found during transform. If only one value is specified, it will be applied to all the target columns, otherwise the number of default values must be equal to the number of target columns. The maximum list length is 2018.

## **TD\_TargetEncodingFit Input**

### **TD\_TargetEncodingFit Input**

The input table schema is as follows:

Column Name	Data Type	Description	Charset	MaxValueLen
target_column	CHAR, VARCHAR	The column that contains the categorical values to be encoded.	LATIN/ UNICODE	128 characters
response_column	INTEGER for classification and NUMERIC group for regression	The column that contains the response values.	Not applicable	Not applicable

The category table schema is as follows:

Column Name	Data Type	Description	Charset	MaxValueLen
ColumnName	CHAR, VARCHAR	The column that contains the column name from the target columns.	LATIN/ UNICODE	128 characters
CategoryCount	BYTEINT, SMALLINT, INTEGER, BIGINT	The column that contains the count of the categories.	Not applicable	Not applicable

## TD\_TargetEncodingFit Output

### TD\_TargetEncodingFit Primary Output Table Schema

The CBM\_BETA encoder output schema is as follows:

Column Name	Data Type	Description	Charset	MaxValueLen
TD_ColumnName_TEFIT	VARCHAR	Contains the names of the target column.	Unicode	128 characters
TD_Category_TEFIT	VARCHAR	Contains the category values for each of the target columns.	Unicode	128 characters
TD_Alpha_TEFIT	REAL	Contains the hyperparameter value.	Not applicable	Not applicable
TD_Beta_TEFIT	REAL	Contains the hyperparameter value.	Not applicable	Not applicable
TD_DefaultCategory_TEFIT	REAL	Contains the default values.	Not applicable	Not applicable
TD_CategoryCount_TEFIT	SMALLINT	Contains the count of the categories.	Not applicable	Not applicable
TD_ColumnIndex_TEFIT	SMALLINT	Contains the index for the target columns.	Not applicable	Not applicable
TD_EncoderMethod_TEFIT_CBM_BETA	BYTEINT	Stores the name of the encoder method.	Not applicable	Not applicable
target_column_i	Same as InputTable	Column names copied to fit output. Needed in TD_TargetEncodingTransform function.	Not applicable	Not applicable

The CBM\_DIRICHLET encoder output schema is as follows:

Column Name	Data Type	Description	Charset	MaxValueLen
TD_ColumnName_TEFIT	VARCHAR	Contains the names of the target column.	Unicode	128 characters
TD_Category_TEFIT	VARCHAR	Contains the category values for each of the target columns.	Unicode	128 characters
TD_Alpha_k_TEFIT_target_column_i	REAL	Contains the hyperparameter value.	Not applicable	Not applicable
TD_DefaultCategory_TEFIT	REAL	Contains the default values.	Not applicable	Not applicable

Column Name	Data Type	Description	Charset	MaxValueLen
TD_CategoryCount_TEFIT	SMALLINT	Contains the count of the categories.	Not applicable	Not applicable
TD_ColumnIndex_TEFIT	SMALLINT	Contains the index for the target columns.	Not applicable	Not applicable
TD_EncoderMethod_TEFIT_CBM_DIRICHLET	BYTEINT	Stores the name of the encoder method.	Not applicable	Not applicable
TD_NumResponses_TEFIT_K	BYTEINT	Stores the name of the NumDistinctResponses value.	Not applicable	Not applicable

The CBM\_GAUSSIAN\_INVERSE\_GAMMA encoder output schema is as follows:

Column Name	Data Type	Description	Charset	MaxValueLen
TD_ColumnName_TEFIT	VARCHAR	Contains the names of the target column.	Unicode	128 characters
TD_Category_TEFIT	VARCHAR	Contains the category values for each of the target columns.	Unicode	128 characters
TD_U0_TEFIT	REAL	Contains the hyperparameter value.	Not applicable	Not applicable
TD_V0_TEFIT	REAL	Contains the hyperparameter value.	Not applicable	Not applicable
TD_Alpha0_TEFIT	REAL	Contains the hyperparameter value.	Not applicable	Not applicable
TD_Beta0_TEFIT	REAL	Contains the hyperparameter value.	Not applicable	Not applicable
TD_DefaultCategory_TEFIT	REAL	Contains the default values.	Not applicable	Not applicable
TD_CategoryCount_TEFIT	SMALLINT	Contains the count of the categories.	Not applicable	Not applicable
TD_ColumnIndex_TEFIT	SMALLINT	Contains the index for the target columns.	Not applicable	Not applicable
TD_EncoderMethod_TEFIT_CBM_GAUSSIAN_INVERSE_GAMMA	BYTEINT	Stores the name of the encoder method.	Not applicable	Not applicable
target_column_i	Same as InputTable	Column names copied to fit output. Needed in TD_	Not applicable	Not applicable

Column Name	Data Type	Description	Charset	MaxValueLen
		TargetEncodingTransform function.		

## TD\_TargetEncodingFit Secondary Output Table Schema

The secondary output table will have the same schema as the primary output table.

## TD\_TargetEncodingFit Examples

### TD\_TargetEncodingFit Example

The following input table contains a subset of titanic dataset.

```

CREATE TABLE titanic_train (
    passenger INTEGER,
    survived INTEGER,
    pclass INTEGER,
    name VARCHAR(90) CHARACTER SET LATIN NOT CASESPECIFIC,
    gender VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC,
    age INTEGER,
    sibsp INTEGER,
    parch INTEGER,
    ticket VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
    fare FLOAT,
    cabin VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
    embarked VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( passenger );

INSERT INTO titanic_train (2, 1, 1, 'Cumings; Mrs. John Bradley (Florence
Briggs Thayer)', 'female', 38, 1, 0, 'PC 17599', 71.2833, 'C85', 'C');
INSERT INTO titanic_train (4, 1, 1, 'Futrelle; Mrs. Jacques Heath (Lily May
Peel)', 'female', 35, 1, 0, '113803', 53.1, 'C123', 'S');
INSERT INTO titanic_train (7, 0, 1, 'McCarthy; Mr. Timothy J', 'male', 54, 0,
0, '17463', 51.8625, 'E46', 'S');
INSERT INTO titanic_train (10, 1, 2, 'Nasser; Mrs. Nicholas (Adele Achem)',
'female', 14, 1, 0, '237736', 30.0708, '', 'C');
INSERT INTO titanic_train (16, 1, 2, 'Hewlett; Mrs. (Mary D Kingcome) ',
'female', 55, 0, 0, '248706', 16, '', 'S');
INSERT INTO titanic_train (21, 0, 2, 'Fynney; Mr. Joseph J', 'male', 35, 0, 0,
'239865', 26, '', 'S');
INSERT INTO titanic_train (40, 1, 3, 'Nicola-Yarred; Miss. Jamila', 'female',
14, 1, 0, '2651', 11.2417, '', 'C');

```

```
INSERT INTO titanic_train (61, 0, 3, 'Sirayanian; Mr. Orsen', 'male', 22, 0, 0,  
'2669', 7.2292, '', 'C');  
INSERT INTO titanic_train (80, 1, 3, 'Dowdell; Miss. Elizabeth', 'female', 30,  
0, 0, '364516', 12.475, '', 'S');  
  
SELECT * FROM titanic_train ORDER BY 1;
```

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
2	1	1	Cumings; Mrs John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	CBS	C
4	1	1	Futrell; Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
7	0	1	McCarthy; Mr. Timothy J	male	54	0	0	117463	51.8625	E46	S
10	1	2	Nasser; Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0707		C
16	1	2	Hewlett; Mrs. (Mary D Kingcome)	female	55	0	0	248706	16		S
21	0	2	Fynney; Mr. Joseph J	male	35	0	0	239865	26		S
40	1	3	Nicola-Yarred; Miss. Jamila	female	14	1	0	2651	11.2417		C
61	0	3	Sirayanian; Mr. Orsen	male	22	0	0	2669	7.2292		C
80	1	3	Dowdell; Miss. Elizabeth	female	30	0	0	364516	12.475		S

CategoryTable:

```
CREATE TABLE categoryTable AS (
SELECT ColumnName, count(*) AS CategoryCount FROM (
SELECT * FROM TD_CategoricalSummary(
ON titanic_train AS InputTable
USING
TargetColumns('gender','embarked')
) AS dt WHERE DistinctValue IS NOT NULL) AS CatTable GROUP BY ColumnName)
WITH data;

SELECT * FROM categoryTable;
```

ColumnName	CategoryCount
embarked	2
gender	2

### **TD\_TargetEncodingFit Query with EncoderMethod CBM\_BETA**

```
TD_TargetEncodingFit (
    ON titanic_train AS InputTable
    ON categoryTable AS CategoryTable DIMENSION
    USING
    EncoderMethod('CBM_BETA')
    TargetColumns('gender','embarked')
    ResponseColumn ('survived')
    ...DefaultValues(-1, -2)
)
```

Output:

TD_ColumnName_TEFIT	TD_Category_TEFIT	TD_Alpha_TEFIT	TD_Beta_TEFIT	TD_DefaultCategory_TEFIT	TD_CategoryCount_TEFIT	TD_ColumnIndex_TEFIT	TD_EncoderMethod_TEFIT_CBM_BETA	gender	embarked
embarked	C	3.666667	1.333333	-2	2	1	?	?	?
embarked	S	3.666667	2.333333	-2	2	1	?	?	?
gender	female	6.666667	0.333333	-1	2	0	?	?	?
gender	male	0.666667	3.333333	-1	2	0	?	?	?

**TD\_TargetEncodingFit Query with EncoderMethod CBM\_DIRICHLET**

```
TD_TargetEncodingFit (
    ON titanic_train AS InputTable
    ON categoryTable AS CategoryTable DIMENSION
    USING
        EncoderMethod('CBM_DIRICHLET')
        TargetColumns('gender', 'embarked')
        ResponseColumn ('pclass')
        DefaultValues(-1, -2)
        NumDistinctResponses(3)
)
```

Output:

## 4: Feature Engineering Transform Functions

TD_ColumnName_TEFIT	TD_Category_TEFIT	TD_Alpha1_TEFIT_gender	TD_Alpha2_TEFIT_gender	TD_Alpha3_TEFIT_gender	TD_Alpha1_TEFIT_embarked	TD_Alpha2_TEFIT_embarked	TD_Alpha3_TEFIT_embarked	TD_DefaultCategory_TEFIT	TD_CategoryCount_TEFIT	TD_ColumnIndex_TEFIT	TD_EncoderMethod_TEFIT_CBM_DIRICHLET	TD_NumResponse_TEFIT_3
embarked	C	?	?	?	1.333333	1.333333	2.333333	-2	2	1	?	?
embarked	S	?	?	?	2.333333	2.333333	1.333333	-2	2	1	?	?
gender	female	2. 333333	2. 333333	2. 333333	?	?	?	-1	2	0	?	?
gender	male	1. 333333	1. 333333	1. 333333	?	?	?	-1	2	0	?	?

**TD\_TargetEncodingFit Query with  
EncoderMethod CBM\_GAUSSIAN\_INVERSE\_GAMMA**

```
TD_TargetEncodingFit (
    ON titanic_train AS InputTable
    ON categoryTable AS CategoryTable DIMENSION
    USING
        EncoderMethod('CBM_GAUSSIAN_INVERSE_GAMMA')
        TargetColumns('gender', 'embarked')
        ResponseColumn ('age')
        DefaultValues(-1, -2)
)
```

Output:

TD_ColumnName_TEFIT	TD_Category_TEFIT	TD_U0_TEFIT	TD_V0_TEFIT	TD_Alpha0_TEFIT	TD_Beta0_TEFIT	TD_DefaultCategory_TEFIT	TD_CategoryCount_TEFIT	TD_Columnindex_TEFIT	TD_EncoderMethod_TEFIT_CBM_GAUSSIAN_INVERSE_GAMMA	gender	embarked
embarked	C	17. 600000	5. 000000	4. 000000	258. 600000	-2	2	1	?	?	?
embarked	S	34. 833333	6. 000000	4. 500000	798. 366667	-2	2	1	?	?	?
gender	female	26. 571429	7. 000000	5. 000000	534. 857143	-1	2	0	?	?	?
gender	male	27. 750000	4. 000000	3. 500000	643. 875000	-1	2	0	?	?	?

## TD\_TargetEncodingFit Usage Notes

- The InputTable in the TD\_TargetEncodingFit query can have no partition at all or have PARTITION BY ANY clause.
- The TD\_TargetEncodingFit function requires a CategoryTable to be passed as a dimension. The CategoryTable can be generated by the TD\_CategoricalSummary function using this query:

```
CREATE TABLE categoryTable AS (
  SELECT ColumnName, count(*) AS CategoryCount FROM (
    SELECT * FROM TD_CategoricalSummary(
      ON titanic_train AS InputTable
      USING
      TargetColumns('[1:2]')
    ) AS dt WHERE DistinctValue IS NOT NULL) AS CatTable GROUP BY ColumnName)
  WITH data;
```

- Null categories are not encoded.
- The DefaultValue argument must be provided to TD\_TargetEncodingFit if you want to assign any target value for missing categories in the TD\_TargetEncodingTransform function.
- These arguments are mandatory, otherwise an error is reported:
  - EncoderMethod
  - TargetColumns
  - ResponseColumn

## TD\_TargetEncodingTransform

The TD\_TargetEncodingTransform function takes the InputTable and a FitTable generated by the [TD\\_TargetEncodingFit](#) function for encoding the categorical values.

---

### Note:

- This function requires the UTF8 client character set.
- This function does not support Pass-Through Characters (PTCs).
- This function does not support KanjiSJIS or Graphic data types.

---

### Function Information

- [TD\\_TargetEncodingTransform Syntax](#)
- [TD\\_TargetEncodingTransform Syntax Elements](#)
- [TD\\_TargetEncodingTransform Input](#)
- [TD\\_TargetEncodingTransform Output](#)

- [TD\\_TargetEncodingTransform Examples](#)
- [TD\\_TargetEncodingTransform Usage Notes](#)

## TD\_TargetEncodingTransform Syntax

```
TD_TargetEncodingTransform (
    ON {table | view | (query)} AS InputTable
    ON {table | view | (query)} AS FitTable DIMENSION
    USING
    [ Accumulate ({'accumulate_column' | 'accumulate_column_range'} [, ...]) ]
)
```

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## TD\_TargetEncodingTransform Syntax Elements

### Accumulate

[Optional] Specifies the columns copied from the input table to the output table. The maximum length supported is 128. The maximum list length is 2047. Not case sensitive.

## TD\_TargetEncodingTransform Input

### TD\_TargetEncodingTransform Input

The input table schema is as follows:

Column Name	Data Type	Description	Charset	MaxLen
target_column	CHAR or VARCHAR	The column that needs to be encoded.	LATIN or UNICODE	128 characters

The FitTable schema is the same as the output schema of [TD\\_TargetEncodingFit Output](#).

## TD\_TargetEncodingTransform Output

### **CBM\_Beta and CBM\_GAUSSIAN\_INVERSE\_GAMMA Encoder Output Schema**

Column Name	Data Type	Description	Charset	MaxLen	Case Sensitive
target_column	INTEGER	Target columns with encoded target values.	Not applicable	Not applicable	Not applicable
accumulate_column	any	Column copied from input table to output.	Same as the Input table.	Same as the Input table.	Same as the Input table.

### **CRM\_DIRICHLET Encoder Output Schema**

Column Name	Data Type	Description	Charset	MaxLen	Case Sensitive
target_column_k	INTEGER	Target columns with encoded target values.	Not applicable	Not applicable	Not applicable
accumulate_column	any	Column copied from input table to output.	Same as the Input table.	Same as the Input table.	Same as the Input table.

## TD\_TargetEncodingTransform Examples

### **TD\_TargetEncodingTransform Example**

InputTable: The following table contains a subset of titanic dataset.

```
CREATE TABLE titanic_train (
    passenger INTEGER,
    survived INTEGER,
    pclass INTEGER,
    name VARCHAR(90) CHARACTER SET LATIN NOT CASESPECIFIC,
    gender VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC,
    age INTEGER,
    sibsp INTEGER,
    parch INTEGER,
    ticket VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
    fare FLOAT,
    cabin VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
    embarked VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC)
```

```

PRIMARY INDEX ( passenger );

INSERT INTO titanic_train (2, 1, 1, 'Cumings; Mrs. John Bradley (Florence Briggs Thayer)', 'female', 38, 1, 0, 'PC 17599', 71.2833, 'C85', 'C');
INSERT INTO titanic_train (4, 1, 1, 'Futrelle; Mrs. Jacques Heath (Lily May Peel)', 'female', 35, 1, 0, '113803', 53.1, 'C123', 'S');
INSERT INTO titanic_train (7, 0, 1, 'McCarthy; Mr. Timothy J', 'male', 54, 0, 0, '17463', 51.8625, 'E46', 'S');
INSERT INTO titanic_train (10, 1, 2, 'Nasser; Mrs. Nicholas (Adele Achem)', 'female', 14, 1, 0, '237736', 30.0708, '', 'C');
INSERT INTO titanic_train (16, 1, 2, 'Hewlett; Mrs. (Mary D Kingcome) ', 'female', 55, 0, 0, '248706', 16, '', 'S');
INSERT INTO titanic_train (21, 0, 2, 'Fynney; Mr. Joseph J', 'male', 35, 0, 0, '239865', 26, '', 'S');
INSERT INTO titanic_train (40, 1, 3, 'Nicola-Yarred; Miss. Jamila', 'female', 14, 1, 0, '2651', 11.2417, '', 'C');
INSERT INTO titanic_train (61, 0, 3, 'Sirayanian; Mr. Orsen', 'male', 22, 0, 0, '2669', 7.2292, '', 'C');
INSERT INTO titanic_train (80, 1, 3, 'Dowdell; Miss. Elizabeth', 'female', 30, 0, 0, '364516', 12.475, '', 'S')

SELECT * FROM titanic_train ORDER BY 1;

```

Input table:

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	emb
2	1	1	Cumings; Mrs John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	CBS	C
4	1	1	Futrell; Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
7	0	1	McCarthy; Mr. Timothy J	male	54	0	0	117463	51.8625	E46	S
10	1	2	Nasser; Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	emb
16	1	2	Hewlett; Mrs. (Mary D Kingcome)	female	55	0	0	248706	16		S
21	0	2	Fynney; Mr. Joseph J	male	35	0	0	239865	26		S
40	1	3	Nicola-Yarred; Miss. Jamila	female	14	1	0	2651	11.2417		C
61	0	3	Sirayanian; Mr. Orsen	male	22	0	0	2669	7.2292		C
80	1	3	Dowdell; Miss. Elizabeth	female	30	0	0	364516	12.475		S

### TD\_TargetEncodingTransform Query with EncoderMethod CBM\_BETA

```
SELECT * FROM TD_TargetEncodingFit (
    ON titanic_train AS InputTable
    ON categoryTable AS CategoryTable DIMENSION
    OUT PERMANENT TABLE OutputTable(betaEncFitTbl)
    USING
        EncoderMethod('CBM_BETA')
        TargetColumns('gender', 'embarked')
        ResponseColumn ('survived')
        DefaultValues(-1, -2)
) AS dt;
```

CBM\_BETA FitTable:

TD_ColumnName_TEFIT	TD_Category_TEFIT	TD_Alpha_TEFIT	TD_Beta_TEFIT	TD_DefaultCategory_TEFIT	TD_CategoryCount_TEFIT	TD_ColumnIndex_TEFIT	TD_Enc_TEFIT_CBM
embarked	C	3.666667	1.333333	-2	2	1	?
embarked	S	3.666667	2.333333	-2	2	1	?
gender	female	6.666667	0.333333	-1	2	0	?

TD_ColumnName_TEFIT	TD_Category_TEFIT	TD_Alpha_TEFIT	TD_Beta_TEFIT	TD_DefaultCategory_TEFIT	TD_CategoryCount_TEFIT	TD_ColumnIndex_TEFIT	TD_EncTEFITCBM
gender	male	0.666667	3.333333	-1	2	0	?

Transform query:

```
SELECT * FROM TD_TargetEncodingTransform (
    ON titanic_train AS InputTable
    ON betaEncFitTbl AS FitTable DIMENSION
    USING
        Accumulate('passenger')
) AS dt ORDER BY passenger;
```

Transform output:

gender	embarked	passenger
0.952381	0.733333	2
0.952381	0.611111	4
0.166667	0.611111	7
0.952381	0.733333	10
0.952381	0.611111	16
0.166667	0.611111	21
0.952381	0.733333	40
0.166667	0.733333	61
0.952381	0.611111	80

### TD\_TargetEncodingFit Query with EncoderMethod CBM\_DIRICHLET

```
SELECT * FROM TD_TargetEncodingFit (
    ON titanic_train AS InputTable
    ON categoryTable AS CategoryTable DIMENSION
    OUT PERMANENT TABLE OutputTable(dirichletEncFitTbl)
    USING
        EncoderMethod('CBM_DIRICHLET')
        TargetColumns('gender', 'embarked')
        ResponseColumn ('pclass')
```

```

DefaultValues(-1, -2)
NumDistinctResponses(3)
) AS dt;

```

CBM\_DIRICHLET FitTable:

TD_ColumnName_TEFIT	TD_Category_TEFIT	TD_Alpha1_TEFIT_gender	TD_Alpha2_TEFIT_gender	TD_Alpha3_TEFIT_gender	TD_Alpha1_TEFIT_embarked	TD_Alpha2_TEFIT_embarked	TD_Alpha3_TEFIT_embarked	TD_DefaultCatTEFIT
embarked	C	?	?	?	1.333333	1.333333	2.333333	-2
embarked	S	?	?	?	2.333333	2.333333	1.333333	-2
gender	female	2. 333333	2. 333333	2. 333333	?	?	?	-1
gender	male	1. 333333	1. 333333	1. 333333	?	?	?	-1

Transform query:

```

SELECT * FROM TD_TargetEncodingTransform (
    ON titanic_train AS InputTable
    ON dirichletEncFitTbl AS FitTable DIMENSION
    USING
        Accumulate('passenger')
) AS dt ORDER BY passenger;

```

Transform output:

gender_1	gender_2	gender_3	embarked_1	embarked_2	embarked_3	passenger
0.333333	0.333333	0.333333	0.266667	0.266667	0.466667	2
0.333333	0.333333	0.333333	0.388889	0.388889	0.222222	4
0.333333	0.333333	0.333333	0.388889	0.388889	0.222222	7
0.333333	0.333333	0.333333	0.266667	0.266667	0.466667	10
0.333333	0.333333	0.333333	0.388889	0.388889	0.222222	16
0.333333	0.333333	0.333333	0.388889	0.388889	0.222222	21
0.333333	0.333333	0.333333	0.266667	0.266667	0.466667	40
0.333333	0.333333	0.333333	0.266667	0.266667	0.466667	61
0.333333	0.333333	0.333333	0.388889	0.388889	0.222222	80

### **TD\_TargetEncodingTransform Query with EncoderMethod CBM\_GAUSSIAN\_INVERSE\_GAMMA**

```
SELECT * FROM TD_TargetEncodingFit (
    ON titanic_train AS InputTable
    ON categoryTable AS CategoryTable DIMENSION
    OUT PERMANENT TABLE OutputTable(GIGEncFitTbl)
    USING
        EncoderMethod('CBM_GAUSSIAN_INVERSE_GAMMA')
        TargetColumns('gender', 'embarked')
        ResponseColumn ('age')
        DefaultValues(-1, -2)
) AS dt;
```

Output:

TD_ColumnName_TEFIT	TD_Category_TEFIT	TD_U0_TEFIT	TD_V0_TEFIT	TD_Alpha0_TEFIT	TD_Beta0_Count	TD_DefaultCategory_TEFIT	TD_CategoryCount_TEFIT	TD_Gamma0_TEFIT
embarked	C	17. 600000	5. 000000	4. 000000	258. 600000	-2	2	1. 000000
embarked	S	34. 833333	6. 000000	4. 500000	798. 366667	-2	2	1. 000000
gender	female	26. 571429	7. 000000	5. 000000	534. 857143	-1	2	0. 000000
gender	male	27. 750000	4. 000000	3. 500000	643. 875000	-1	2	0. 000000

Transform query:

```
SELECT * FROM TD_TargetEncodingTransform (
    ON titanic_train AS InputTable
    ON GIGEncFitTbl AS FitTable DIMENSION
    USING
        Accumulate('passenger')
) AS dt ORDER BY passenger;
```

Transform output:

gender	embarked	passenger
26.57143	17.6	2
26.57143	34.83333	4
27.75	34.83333	7
26.57143	17.6	10
26.57143	34.83333	16
27.75	34.83333	21
26.57143	17.6	40
27.75	17.6	61
26.57143	34.83333	80

## TD\_TargetEncodingTransform Usage Notes

- Errors are generated in these cases:
  - Two tables must be input to TD\_TargetEncodingTransform function. If one ON clause is missing, an error is generated.
  - When the fit table does not meet the criteria.
  - When Category from input table is not found in the FIT table and the defaultValues argument is also not used during Fit function.

## TD\_Unpivoting

TD\_Unpivoting function is used to unpivot the data, that is, changes the data from dense format to sparse format.

TD\_Unpivoting follows this process:

- Select a table with data in dense format.
- Use TD\_Unpivoting to change data from dense format to sparse format.

### Function Information

- [TD\\_Unpivoting Syntax](#)
- [Required Syntax Elements for TD\\_Unpivoting](#)
- [Optional Syntax Elements for TD\\_Unpivoting](#)
- [TD\\_Unpivoting Input](#)
- [TD\\_Unpivoting Output](#)
- [TD\\_Unpivoting Usage Notes](#)
- [Example: How to Use TD\\_Unpivoting](#)

## TD\_Unpivoting Syntax

```
TD_Unpivoting(
    ON {table | view | (query)} AS InputTable PARTITION BY ANY
    USING
        IDColumn('id_column')
    TARGETCOLUMNS ({'target_column' | target_column_range}[,...])
    [ ATTRIBUTEALIASLIST ( 'attribute_alias_list_value' [, ...] ) ]
    [ ATTRIBUTECOLNAME('Output_ColumnName_For_AttributeName') ]
    [ VALUECOLNAME('Output_ColumnName_For_AttributeValue') ]
    [ ACCUMULATE ({'accumulate_column' | accumulate_column_range }[,...]) ]
    [ INCLUDENULLS({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ INPUTTYPES({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ OUTPUTVARCHAR({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ INDEXEDATTRIBUTE({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ INCLUDEDATATYPES({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ])
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_Unpivoting

### ON clause

Specifies the table name, view name or query as an InputTable.

### IDColumn

Specifies the column from the InputTable which contains the identifier.

### TargetColumns

Specifies the columns from the InputTable which need to be unpivoted.

## Optional Syntax Elements for TD\_Unpivoting

### **AttributeAliasList**

Specifies alternate name for the values in the AttributeName column. The number of attribute aliases must match with number of target columns.

### **AttributeColName**

Specifies the output column name for the AttributeName column.

Default: AttributeName.

### **ValueColName**

Specifies the output column name for theAttributeValue column.

Default: AttributeValue.

### **Accumulate**

Specifies the name of the input columns to copy to the output table.

### **IncludeNulls**

Specifies whether or not to include nulls in the transformation.

Default: false.

### **InputTypes**

If you specify 'true', instead of one column for all attribute values, the output table includes multiple columns. The attribute values appears in the column corresponding to the compatible data type of the value in the input table.

Default: 'false' (for each unpivoted column, the function outputs the values in a single column.)

Default: false.

### **OutputVarchar**

Specify 'true' if you want to output theAttributeValue column in VARCHAR format irrespective of its data type.

Default: false.

---

#### **Note:**

Not required with the InputTypes argument.

---

**IndexedAttribute**

Specify 'true' if you want the column indexes instead of column names in the AttributeName column.

Default: false.

**IncludeDataTypes**

Specify 'true' to output the original data type name.

Default: false.

## TD\_Unpivoting Input

**InputTable Schema**

Column Name	Data Type	Description
id_column	Any	Column which contains the identifier values.
target_column	CHAR, VARCHAR, CLOB, BYTE, VARBYTE, BLOB, BYTEINT, SMALLINT, INTEGER, BIGINT, FLOAT, REAL, DOUBLE PRECISION, DECIMAL, NUMBER, DATE, TIME, TIMESTAMP, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE, INTERVAL SECOND, INTERVAL MINUTE TO SECOND, INTERVAL MINUTE, INTERVAL HOUR TO SECOND, INTERVAL HOUR TO MINUTE, INTERVAL HOUR, INTERVAL DAY TO SECOND, INTERVAL DAY TO MINUTE, INTERVAL DAY TO HOUR, INTERVAL DAY, INTERVAL MONTH, INTERVAL YEAR TO MONTH, INTERVAL YEAR	Column which needs to be unpivoted.
accumulate_column	Any	Column which needs to be copied to the output table.

## TD\_Unpivoting Output

**Output Table Schema**

Column	Data Type	Description
id_column	Any	Column copied to output table.
AttributeName	VARCHAR or SMALLINT	Column which contains the unpivoted attribute or attribute alias or index of the target column.
AttributeValue	Original compatible data type from InputTable if all	Column which contains the unpivoted values of the attribute. (Column appears if InputTypes is 'false').

Column	Data Type	Description
	columns are of same group or VARCHAR	
AttributeValue_<GRP_NAME>	According to the group	Column which contains the unpivoted values of the attribute, for example, AttributeValue_Num.(Columns appears if InputTypes is 'true'). Supported GRP_NAME are: {Num, Char, Clob, Byte, Blob, Date, Time, TimeWithTZ,TimeStamp,TimeStampWithTZ,IntervalSecond, IntervalMinute, IntervalHour, IntervalDay, IntervalMonth, IntervalYear, IntervalMinuteSecond, IntervalHourSecond, IntervalHourMinute, IntervalDaySecond, IntervalDayMinute, IntervalDayHour, IntervalYearMonth }
DataTypes	VARCHAR	Column which contains the original data type name from the InputTable.
accumulate_column	Any	Column copied to output table.

## TD\_Unpivoting Usage Notes

The following are usage considerations for the TD\_Unpivoting function:

- TD\_Unpivoting function is aimed to be used in conjunction with other inDB analytic functions, in contrast to SQLE Unpivot.
- The function returns N\*M output rows, if there are N rows in the input table and M features (columns). Due to this large output size, expect the query to run longer as the number of rows increases in the InputTable. The output size is limited by the user spool space, which in turn, limits the size or scalability of the input.
- With InputTypes as false, do note combine BLOB/CLOB data type columns with any other data type columns.

## Example: How to Use TD\_Unpivoting

### InputTable

---

#### Note:

The following contains a subset of titanic dataset.

---

```
DROP TABLE unpivoting_titanic_dataset;
CREATE TABLE unpivoting_titanic_dataset (
    passenger INTEGER,
    survived INTEGER,
    pclass INTEGER,
    name VARCHAR(90) CHARACTER SET LATIN NOT CASESPECIFIC,
```

```

gender VARCHAR(10) CHARACTER SET UNICODE NOT CASESPECIFIC,
age INTEGER,
sibsp INTEGER,
parch INTEGER,
ticket VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
fare FLOAT,
cabin VARCHAR(20) CHARACTER SET LATIN NOT CASESPECIFIC,
embarked VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( passenger );

INSERT INTO unpivoting_titanic_dataset (2, 1, 1, 'Cumings; Mrs. John Bradley
(Florence Briggs Thayer)', 'female', 38, 1, 0, 'PC 17599', 71.2833, 'C85', 'C');
INSERT INTO unpivoting_titanic_dataset (4, 1, 1, 'Futrelle; Mrs. Jacques Heath
(Lily May Peel)', 'female', 35, 1, 0, '113803', 53.1, 'C123', 'S');
INSERT INTO unpivoting_titanic_dataset (7, 0, 1, 'McCarthy; Mr. Timothy J',
'male', 54, 0, 0, '17463', 51.8625, 'E46', 'S');
INSERT INTO unpivoting_titanic_dataset (10, 1, 2, 'Nasser; Mrs. Nicholas (Adele
Achem)', 'female', 14, 1, 0, '237736', 30.0708, '', 'C');
INSERT INTO unpivoting_titanic_dataset (16, 1, 2, 'Hewlett; Mrs. (Mary D
Kingcome)', 'female', 55, 0, 0, '248706', 16, '', 'S');
INSERT INTO unpivoting_titanic_dataset (21, 0, 2, 'Fynney; Mr. Joseph J',
'male', 35, 0, 0, '239865', 26, '', 'S');
INSERT INTO unpivoting_titanic_dataset (40, 1, 3, 'Nicola-Yarred; Miss.
Jamila', 'female', 14, 1, 0, '2651', 11.2417, '', 'C');
INSERT INTO unpivoting_titanic_dataset (61, 0, 3, 'Sirayanian; Mr. Orsen',
'male', 22, 0, 0, '2669', 7.2292, '', 'C');
INSERT INTO unpivoting_titanic_dataset (1000, 1, 3, 'ABC', NULL, 30, 0, 0,
'00000', 100.50, '', 'S');

```

## SQL Call 1

```

SELECT * FROM TD_UNPIVOTING(
ON unpivoting_titanic_dataset AS InputTable PARTITION BY ANY
USING
IDCOLUMN('passenger')
TARGETCOLUMNS ('gender')
ACCUMULATE ('survived')
INCLUDENULLS('true')
)AS dt ORDER BY 2, 1;

```

## Output 1

passenger	AttributeName	AttributeValue	survived

	gender	female	1
2	gender	female	1
4	gender	female	1
7	gender	male	0
10	gender	female	1
16	gender	female	1
21	gender	male	0
40	gender	female	1
61	gender	male	0

**SQL Call 2**

```
SELECT * FROM TD_UNPIVOTING(
ON unpivoting_titanic_dataset AS InputTable PARTITION BY ANY
USING
IDCOLUMN('passenger')
TARGETCOLUMNS ('gender')
ATTRIBUTEALIASLIST ('gender_titanic')
ATTRIBUTECOLNAME('Attribute')
VALUECOLNAME('value')
ACCUMULATE ('survived')
INCLUDENULLS('true')
)AS dt ORDER BY 2, 1;
```

**Output 2**

passenger	Attribute	Value	survived
2	gender_titanic	female	1
4	gender_titanic	female	1
7	gender_titanic	male	0
10	gender_titanic	female	1
16	gender_titanic	female	1
21	gender_titanic	male	0
40	gender_titanic	female	1
61	gender_titanic	male	0
1000	gender_titanic	?	1

**SQL Call 3**

```
SELECT * FROM TD_UNPIVOTING(
ON unpivoting_titanic_dataset AS InputTable PARTITION BY ANY
USING
```

```

IDCOLUMN('passenger')
TARGETCOLUMNS ('gender','embarked')
ATTRIBUTECOLNAME('Attribute')
VALUECOLNAME('value')
ACCUMULATE ('survived')
INCLUDENULLS('true')
INDEXEDATTRIBUTE('true')
INCLUDEDATATYPES('true')
)AS dt ORDER BY 2, 1;

```

### Output 3

passenger	Attribute	Value	DataTypes		survived
2	2	female	VARCHAR(10) CHARACTER SET UNICODE	1	
4	2	female	VARCHAR(10) CHARACTER SET UNICODE	1	
7	2	male	VARCHAR(10) CHARACTER SET UNICODE	0	
10	2	female	VARCHAR(10) CHARACTER SET UNICODE	1	
16	2	female	VARCHAR(10) CHARACTER SET UNICODE	1	
21	2	male	VARCHAR(10) CHARACTER SET UNICODE	0	
40	2	female	VARCHAR(10) CHARACTER SET UNICODE	1	
61	2	male	VARCHAR(10) CHARACTER SET UNICODE	0	
1000	2	?	VARCHAR(10) CHARACTER SET UNICODE	1	
2	3	C	VARCHAR(10) CHARACTER SET LATIN	1	
4	3	S	VARCHAR(10) CHARACTER SET LATIN	1	
7	3	S	VARCHAR(10) CHARACTER SET LATIN	0	
10	3	C	VARCHAR(10) CHARACTER SET LATIN	1	
16	3	S	VARCHAR(10) CHARACTER SET LATIN	1	
21	3	S	VARCHAR(10) CHARACTER SET LATIN	0	
40	3	C	VARCHAR(10) CHARACTER SET LATIN	1	
61	3	C	VARCHAR(10) CHARACTER SET LATIN	0	
1000	3	S	VARCHAR(10) CHARACTER SET LATIN	1	

### SQL Call 4

```

SELECT * FROM TD_UNPIVOTING(
ON unpivoting_titanic_dataset AS InputTable PARTITION BY ANY
USING
IDCOLUMN('passenger')
TARGETCOLUMNS ('gender','fare')
ACCUMULATE ('survived')
INCLUDENULLS('true')

```

```
INPUTTYPES('true')
)AS dt ORDER BY 2, 1;
```

**Output 4**

passenger	AttributeName	AttributeValue_Num	AttributeValue_Char	survived
2	fare	7.12833000000000E 001	?	1
4	fare	5.31000000000000E 001	?	1
7	fare	5.18625000000000E 001	?	0
10	fare	3.00708000000000E 001	?	1
16	fare	1.60000000000000E 001	?	1
21	fare	2.60000000000000E 001	?	0
40	fare	1.12417000000000E 001	?	1
61	fare	7.22920000000000E 000	?	0
1000	fare	1.00500000000000E 002	?	1
2	gender	?	female	1
4	gender	?	female	1
7	gender	?	male	0
10	gender	?	female	1
16	gender	?	female	1
21	gender	?	male	0
40	gender	?	female	1
61	gender	?	male	0
1000	gender	?	?	1

**SQL Call 5**

```
SELECT * FROM TD_UNPIVOTING(
ON unpivoting_titanic_dataset AS InputTable PARTITION BY ANY
USING
IDCOLUMN('passenger')
TARGETCOLUMNS ('gender','fare')
ATTRIBUTECLNAME('Attribute')
VALUECLNAME('value')
ACCUMULATE ('survived')
INCLUDENULLS('true')
INPUTTYPES('true')
)AS dt ORDER BY 2, 1;
```

**Output 5**

passenger	Attribute	Value_Num	Value_Char	survived
2	fare	7.12833000000000E 001	?	1
4	fare	5.31000000000000E 001	?	1
7	fare	5.18625000000000E 001	?	0
10	fare	3.00708000000000E 001	?	1
16	fare	1.60000000000000E 001	?	1
21	fare	2.60000000000000E 001	?	0
40	fare	1.12417000000000E 001	?	1
61	fare	7.22920000000000E 000	?	0
1000	fare	1.00500000000000E 002	?	1
2	gender	?	female	1
4	gender	?	female	1
7	gender	?	male	0
10	gender	?	female	1
16	gender	?	female	1
21	gender	?	male	0
40	gender	?	female	1
61	gender	?	male	0
1000	gender	?	?	1

# Feature Engineering Utility Functions

## TD\_FillRowID

Adds a column of unique row identifiers to the input table.

## TD\_NumApply

Applies a specified numeric operator to the specified input table columns.

## TD\_RoundColumns

Rounds the values of each specified input table column to a specified number of decimal places.

## TD\_StrApply

Applies a specified string operator to the specified input table columns.

## **TD\_FillRowID**

TD\_FillRowID adds a column of unique row identifiers to an input table.

---

### Note:

This function may not return the same RowIds if the function is run multiple times.

---

Row IDs are used in tables to uniquely identify each row within the table. They serve to ensure that each row in the table has a unique identifier that can be easily retrieved or updated.

Row IDs are typically assigned automatically by the database management system (DBMS) when a new row is added to the table, although they can also be assigned manually by a user.

Row IDs are useful for a variety of reasons, including:

- Ensuring data integrity: With a unique identifier for each row, it is easier to ensure that there are no duplicate rows in the table, which can cause data inconsistencies and errors.
- Enabling efficient searches: When searching for specific rows within a table, using the row ID as a search criterion can be faster and more efficient than searching based on other attributes.
- Supporting table relationships: In relational databases, row IDs are often used as foreign keys in related tables, which allows for linking between different tables in the database.

Overall, row IDs are a fundamental component of table design and are essential for the proper functioning of many database applications.

### **Function Information**

- [TD\\_FillRowID Syntax](#)

- [Required Syntax Elements for TD\\_FillRowID](#)
- [Optional Syntax Elements for TD\\_FillRowID](#)
- [TD\\_FillRowID Input](#)
- [TD\\_FillRowID Output](#)
- [Example: Using TD\\_FillRowID to Generate Row Identifiers](#)

## TD\_FillRowID Syntax

```
TD_FillRowID (
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY [ ORDER BY
order_column ] ]
    USING
    [ RowIDColumnName ('row_id_column') ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_FillRowID

### ON clause

Accept the PARTITION BY and ORDER BY clauses.

- The PARTITION BY clause specifies that a table is partitioned by one or more partitioning levels. A partitioning level can be defined using a single COLUMN keyword, a partitioning expression, or a combination of both.
- The ORDER BY clause specifies how result sets are sorted. If it is not used, the result rows are returned unsorted.

## Optional Syntax Elements for TD\_FillRowID

### RowIDColumnName

Specify a name for the output column of row identifiers.

Default: row\_id

## TD\_FillRowID Input

### InputTable Schema

InputTable can have any schema.

## TD\_FillRowID Output

### Output Table Schema

Column	Data Type	Description
<i>row_id_column</i>	BIGINT	Column of row identifiers.
<i>input_column</i>	Same as in InputTable	Column copied from InputTable.

## Example: Using TD\_FillRowID to Generate Row Identifiers

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_FillRowID InputTable: fillrowid\_input

Survived	Pclass	Name	Age
0	3	Mrs. Jacques Heath	35
0	3	Mr. Owen Harris	22
1	3	Mrs. Laina	26
1	1	Mrs. John Bradley	38

### TD\_FillRowID SQL Call

```
SELECT * FROM TD_FillRowID (
    ON fillrowid_input AS InputTable
    PARTITION BY Pclass
    ORDER BY Age
    USING
    RowIDColumnName ('PassengerId')
) AS dt;
```

### TD\_FillRowID Output

The output shows the generated IDs in the PassengerId column.

Survived	Pclass	Name	Age	PassengerId
1	1	Mrs. John Bradley	38	4
0	3	Mr. Owen Harris	22	4
1	3	Mrs. Laina	26	8
0	3	Mrs. Jacques Heath	35	12

## TD\_NumApply

TD\_NumApply applies a specified numeric operator to specified input table columns.

Use the TD\_NumApply function to apply a user-defined, numeric operator to a specified column or a set of columns in a Teradata database table. You can specify one of the following operators:

- Exponential function
- Logarithmic function
- Sigmoid function
- Inverse of Sin function
- Hyperbolic Tangent function

The TD\_NumApply function is useful for a range of data analysis and manipulation tasks, such as performing complex calculations. You can use the function in data analysis to identify relationships between multiple variables and columns or measure the impact of different factors on a given outcome.

Hence, the TD\_NumApply function is a powerful tool for performing numerical calculations and analysis on large datasets in Teradata databases. Its efficient parallel processing capabilities make it a valuable asset for businesses and organizations looking to gain insights from their data quickly and efficiently.

### Function Information

- [TD\\_NumApply Syntax](#)
- [Required Syntax Elements for TD\\_NumApply](#)
- [Optional Syntax Elements for TD\\_NumApply](#)
- [TD\\_NumApply Input](#)
- [TD\\_NumApply Output](#)
- [TD\\_NumApply Usage Notes](#)

- [Example: How to Use TD\\_NumApply](#)

**Related Information:**

[TD\\_StrApply](#)  
[TD\\_FunctionFit](#)  
[TD\\_FunctionTransform](#)

## TD\_NumApply Syntax

```
TD_NumApply (
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY [ ORDER BY
order_column ] ]
    USING
        TargetColumns ({ 'target_column' | 'target_column_range' }[,...])
        [ OutputColumns ('output_column' [,...]) ]
        [ Accumulate ({ 'accumulate_column' | 'accumulate_column_range' }[,...]) ]
        ApplyMethod ('num_operator')
        [ SigmoidStyle ({ 'logit' | 'modifiedlogit' | 'tanh' })]
        [ InPlace ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
)
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_NumApply

**ON clause**

Specifies the table name, view name or query as an InputTable.

**TargetColumns**

Specifies the names of the InputTable columns to apply the numeric operator.

**ApplyMethod**

Main base attribute for this function that specifies the operator that you want to apply to the given columns. Accepts one of these numeric operators:

num_operator	Description
EXP	Raises e (base of natural logarithms) to power of value, where e = 2.71828182845905.
LOG	Computes base 10 logarithm of value.
SIGMOID	Applies sigmoid function to value.
SININV	Computes inverse hyperbolic sine of value.
TANH	Computes hyperbolic tangent of value.

## Optional Syntax Elements for TD\_NumApply

### OutputColumns

[Ignored with Inplace ('true'), otherwise optional.] Specifies the names for the output columns. An output\_column cannot exceed 128 characters.

Default: With InPlace ('false'), target\_column\_operator; otherwise target\_column.

#### Note:

If any target\_column\_operator exceeds 128 characters, specify an output\_column for each target\_column.

### Accumulate

Specifies the names of the InputTable columns to copy to the output table.

With InPlace ('true'), no target\_column can be an accumulate\_column.

### SigmoidStyle

Specifies the sigmoid style.

#### Note:

Required with ApplyMethod ('sigmoid'), otherwise ignored.

Default: logit

### InPlace

Specifies whether the output columns have the same names as the target columns.

InPlace ('true') effectively replaces each value in each target column with the result of applying num\_operator to it.

InPlace ('false') copies the target columns to the output table and adds output columns whose values are the result of applying num\_operator to each value.

With InPlace ('true'), no target\_column can be an accumulate\_column.

Default: true

## TD\_NumApply Input

### InputTable Schema

Column	Data Type	Description
target_column	NUMERIC	Column to which to apply num_operator.
accumulate_column	Any	Column to copy to output table.

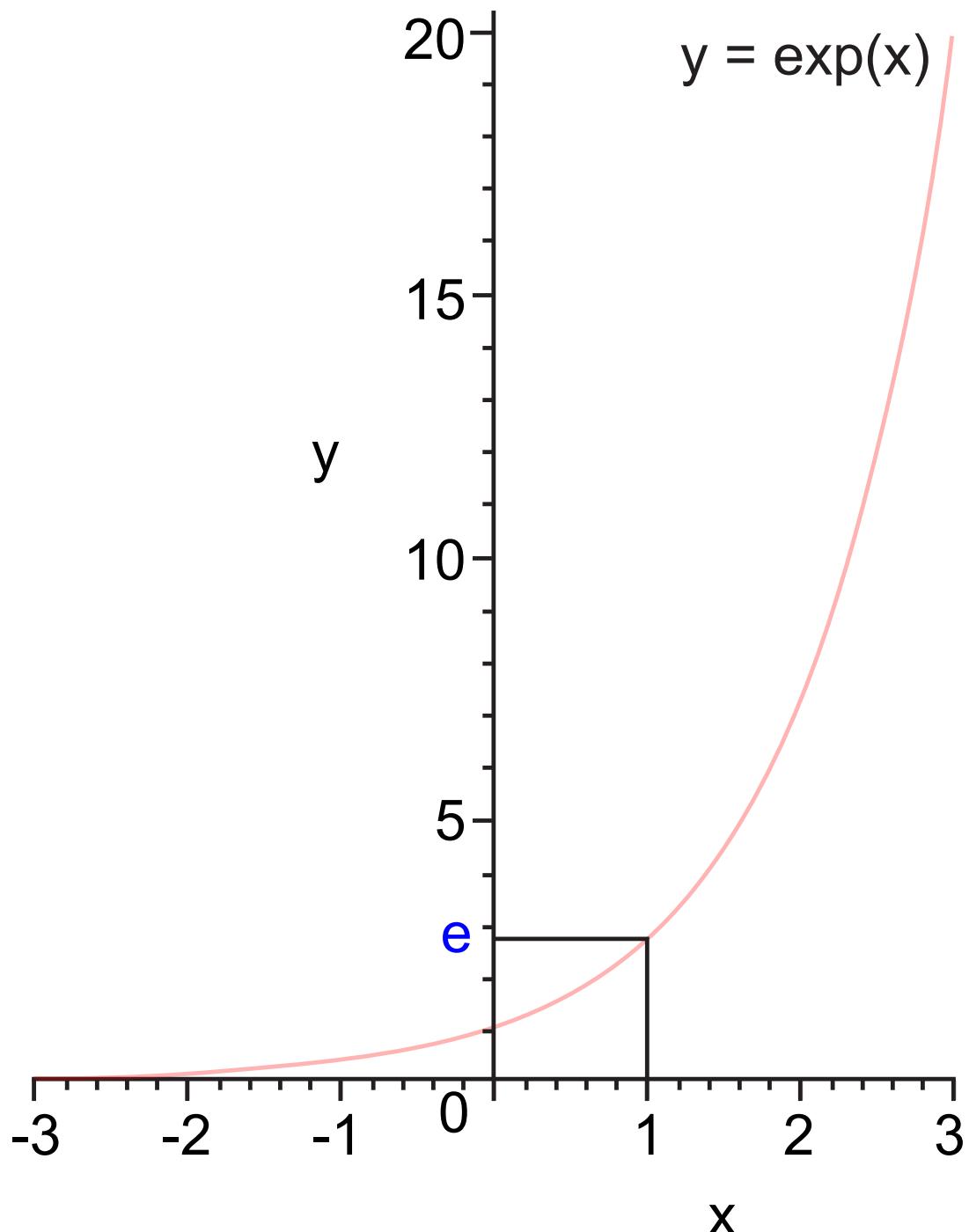
## TD\_NumApply Output

### Output Table Schema

Column	Data Type	Description
accumulate_column	Same as in InputTable	Column copied from InputTable.
output_column	Same as in InputTable	Column to that num_operator was applied. With InPlace ('true'), output_column is target_column. With InPlace ('false'), OutputColumns determines output_column.

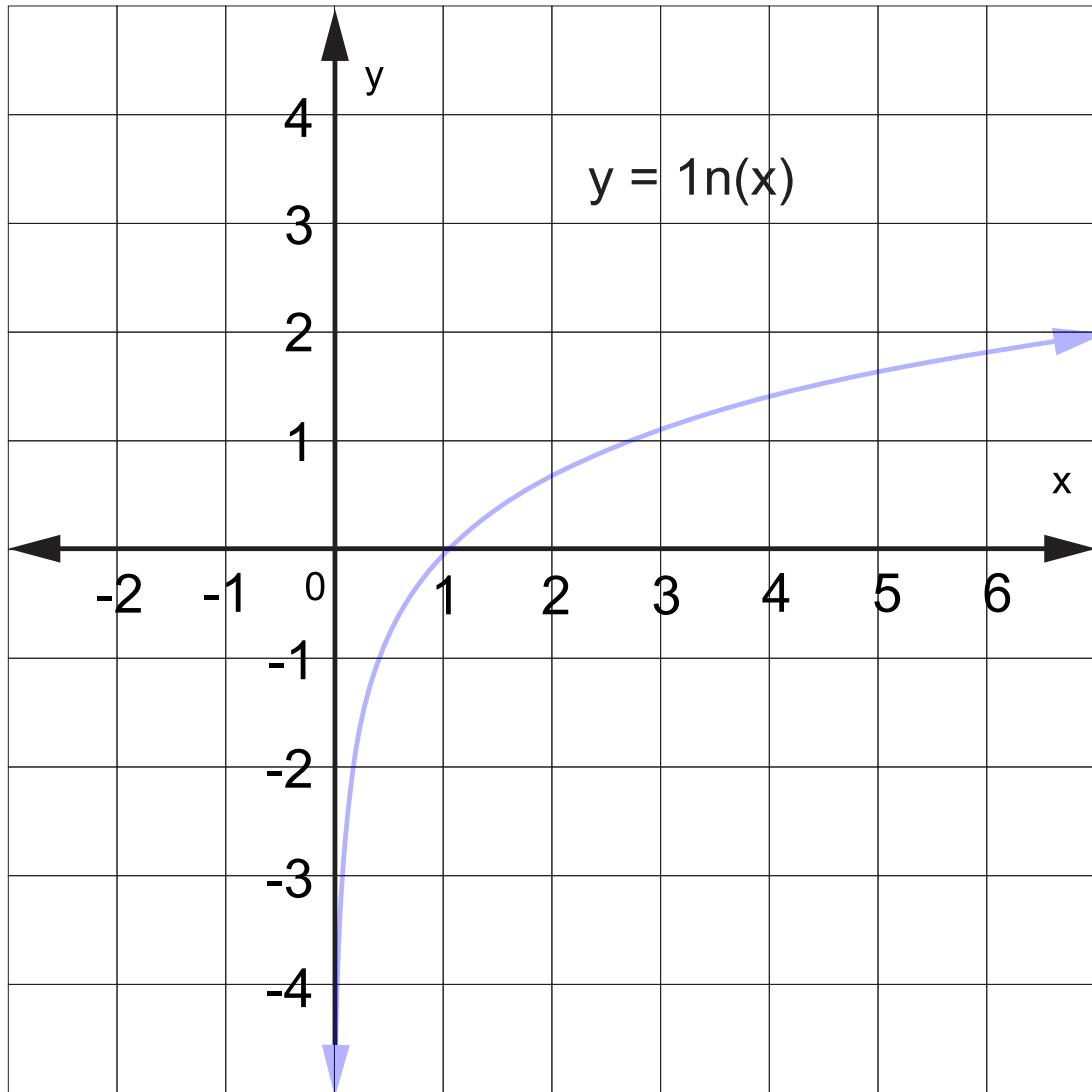
## TD\_NumApply Usage Notes

As TD\_NumApply applies a specified numeric operator to the specified input table columns. You can apply the following numeric operators:



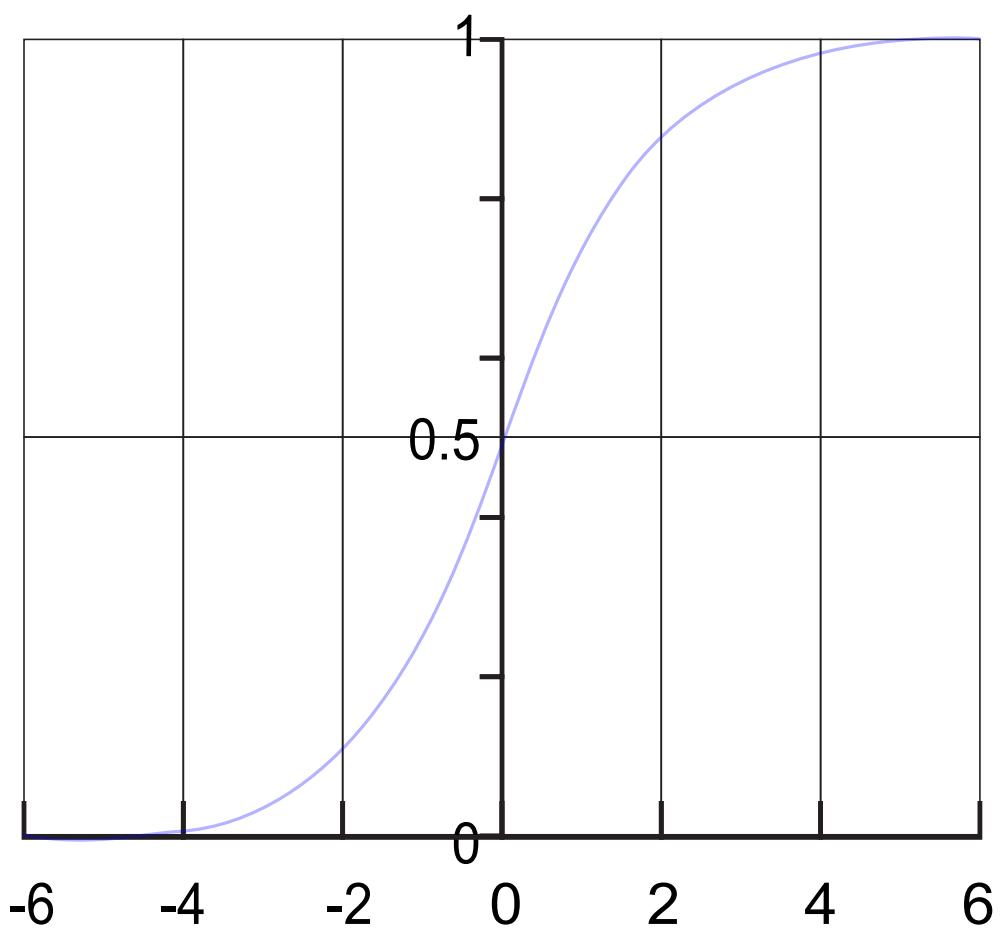
**EXP:** The exponential function is a mathematical function denoted by  $f(x) = \exp(x) = e^x$  (where the argument  $x$  is written as an exponent). The term generally refers to the positive-valued function of a real variable, although it can be extended to the complex numbers or generalized to other mathematical objects like matrixes or Lie algebras. You can use an exponential function to calculate the exponential growth or decay of a given set of data. For example, you can use exponential functions

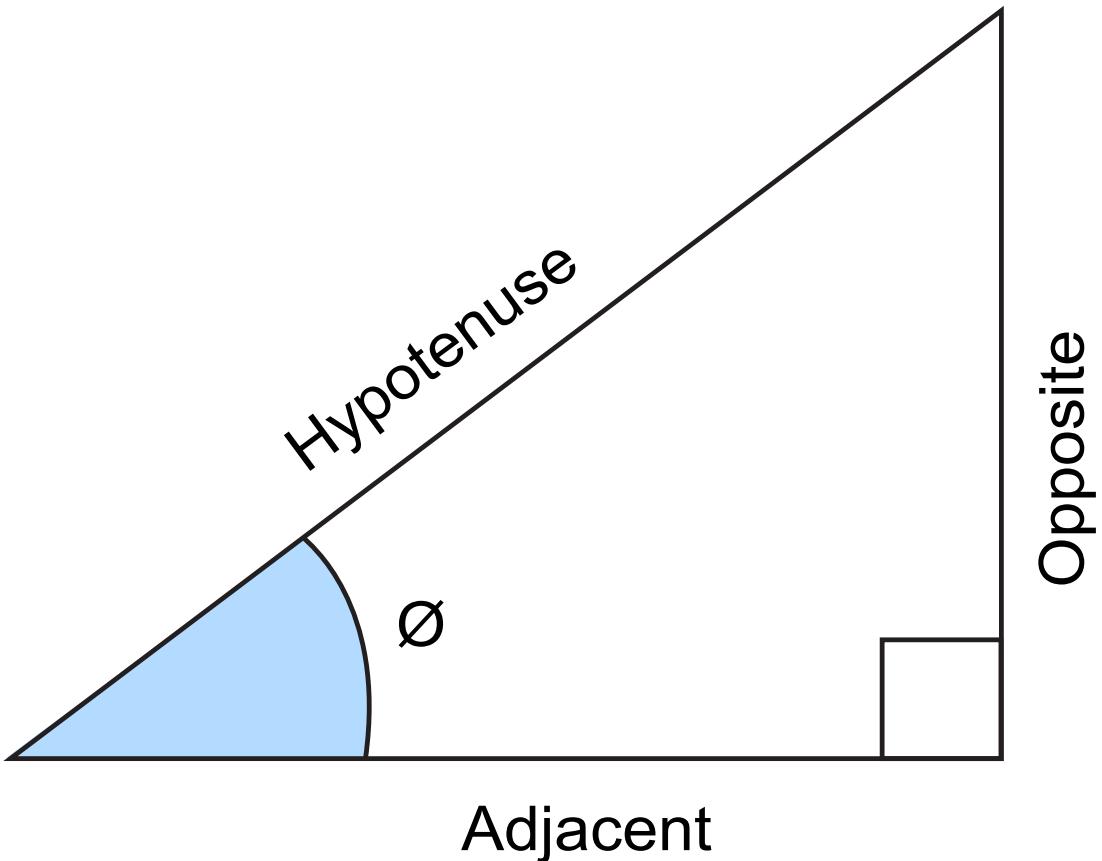
to calculate changes in population, loan interest charges, bacterial growth, radioactive decay or the spread of disease.



**LOG:** Using logarithms makes the values onto a comparable scale and brings in “linearity” to the values. This function is denoted by  $f(x)=\ln(x)$  which is a strictly increasing function. Two main reasons to use logarithmic scales in charts and graphs follow:

- To respond to skewness towards large values; specifically, cases in which one or a few points are much larger than the bulk of the data.
- To show percent change or multiplicative factors.

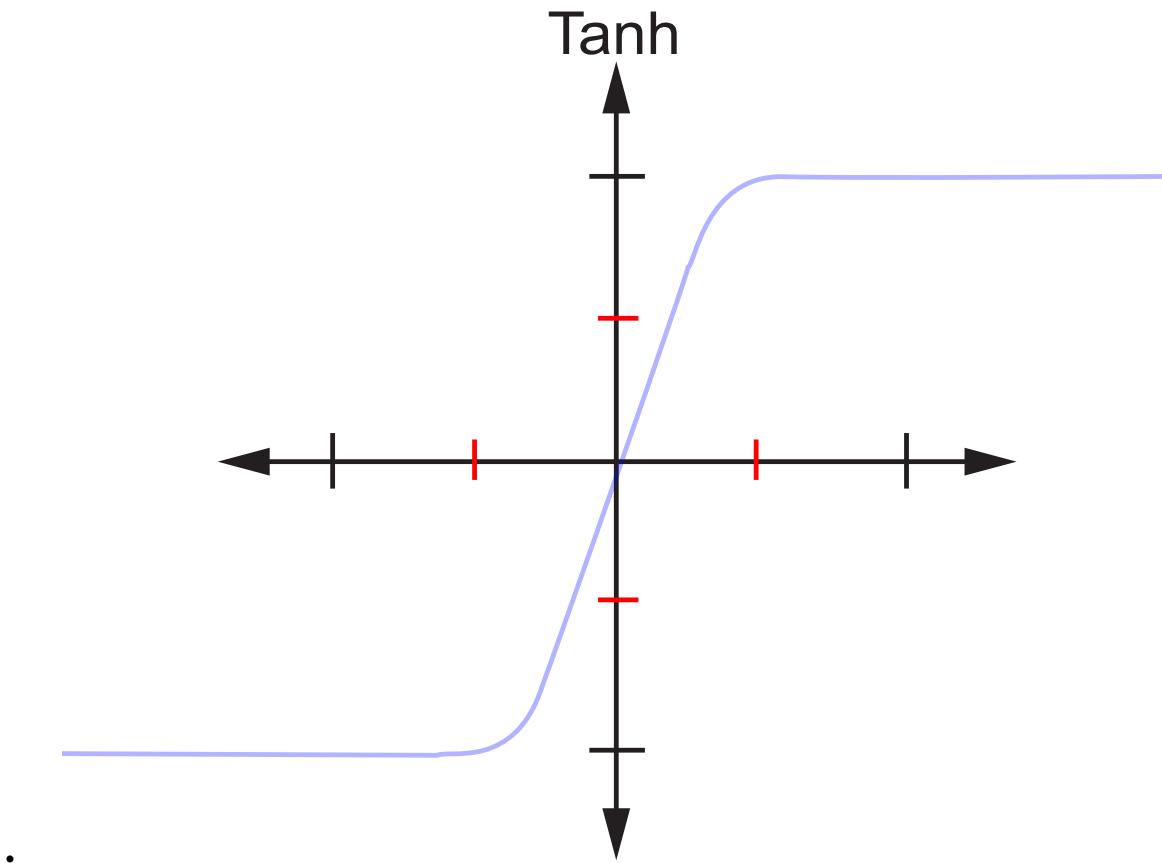




**SIGMOID:** A sigmoid function is a bounded, differentiable, real function that is defined for all real input values and has a non-negative derivative at each point and exactly one inflection point. It is denoted

by  $f(x) = \frac{1}{1+e^{-x}}$ . The sigmoid function's ability to transform any real number to one between 0 and 1 is advantageous in data science and many other fields such as in deep learning as a non-linear activation function within neurons, and in artificial neural networks to allow the network to learn non-linear relationships between the data.

- **SININV:** Sine inverse or arcsine is the inverse of sine function. It is denoted by  $f(x) = \sin^{-1}(x)$ .



**TANH:** Tanh is the hyperbolic tangent function denoted by  $f(x) = \tanh(x)$ . The tanh function is mainly used for classification between two classes. Both tanh and logistic sigmoid activation functions are used in feed-forward nets. Use the hyperbolic tangent sigmoid function as transfer function for hidden neurons and the linear transfer function for output layer.

## Example: How to Use TD\_NumApply

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_NumApply InputTable: input\_table

passenger	survived	pclass	gender	age	sibsp	parch	fare	cabin	embarked
1	0	3	male	22	1	0	7.250000000	null	S
2	1	1	female	38	1	0	71.280000000	C85	C
3	1	3	female	26	0	0	7.930000000	null	S
4	1	1	female	35	1	0	53.100000000	C123	S
5	0	3	male	35	0	0	8.050000000	null	S

## Example: TD\_NumApply Using LOG Method

```
SELECT * FROM TD_NumApply (
    ON numApply_input AS InputTable PARTITION BY ANY
    USING
        TargetColumns ('Age', 'Fare')
        ApplyMethod ('log')
        Accumulate ('Passenger')
        InPlace ('true')
) AS dt;
```

## TD\_NumApply Output

passenger	age	fare
1	3.091042453	1.981001469
2	3.637586160	4.266615783
3	3.258096538	2.070653036
4	3.555348061	3.972176928
5	3.555348061	2.085672091

## TD\_RoundColumns

TD\_RoundColumns rounds the values of each specified input table column to a specified number of decimal places.

Rounding input table columns to a specified number of decimal places in data engineering can have several benefits:

- Data consistency: Ensure that the data remains consistent across different sources and applications that use the data. This can prevent errors and inconsistencies that can arise from rounding differences between different systems.
- Data reduction: Reduce the size of the dataset, making it easier to process and analyze. This can be especially important in big data applications where the size of a dataset can be very large.
- Improved accuracy: Improve accuracy by reducing the impact of minor variations in numeric values that may not be relevant to the analysis being performed. For example, if you are working with financial data and only need to analyze amounts to the nearest dollar, rounding the values to the nearest dollar could make it easier to identify patterns and trends in the data.
- Improved visualization: Visualize the data. For example, if the data is being plotted on a graph, rounding the input values can reduce the number of data points and make the graph easier to read.
- Improved performance: Improve the performance of certain algorithms and machine learning models that are computationally expensive. Rounding can reduce the computational complexity of these algorithms and models, making them faster and more efficient.

## Function Information

- [TD\\_RoundColumns Syntax](#)
- [Required Syntax Elements for TD\\_RoundColumns](#)
- [Optional Syntax Elements for TD\\_RoundColumns](#)
- [TD\\_RoundColumns Input](#)
- [TD\\_RoundColumns Output](#)
- [Example: How to Use TD\\_RoundColumns](#)

## TD\_RoundColumns Syntax

```
TD_RoundColumns (
    ON { table | view | (query) } AS InputTable
    USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        [ PrecisionDigit (precision) ]
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_RoundColumns

### ON clause

Accept the *InputTable* clause.

### TargetColumns

Specify the names of the *InputTable* columns in which to round every value to *precision* digits.

## Optional Syntax Elements for TD\_RoundColumns

### PrecisionDigit

Specify the number of decimal places to which to round values.

If *precision* is positive, the function rounds values to the right of the decimal point.

If *precision* is negative, the function rounds values to the left of the decimal point.

Default: If the PrecisionDigit value is not provided, the function rounds the column values to 0 places.

---

**Note:**

If the column values have the DECIMAL/NUMERIC data type with a precision less than 38, then the function increases the precision by 1. For example, when a DECIMAL (4,2) value of 99.99 is rounded to 0 places, the function returns a DECIMAL (5,2) value, 100.00. However, if the precision is 38, then the function only reduces the scale value by 1 unless the scale is 0. For example, the function returns a DECIMAL (38, 36) value of 99.999999999 as a DECIMAL (38, 35) value, 100.00.

---

## Accumulate

Specify the names of the InputTable columns to copy to the output table.

## TD\_RoundColumns Input

### InputTable Schema

Column	Data Type	Description
target_column	NUMERIC	Column in which to round every value to <i>precision</i> digits.
accumulate_column	Any	Column to copy to output table.

## TD\_RoundColumns Output

### Output Table Schema

Column	Data Type	Description
target_column	NUMERIC	Column in that every value is rounded to <i>precision</i> digits.
accumulate_column	Any	Column copied from InputTable.

## Example: How to Use TD\_RoundColumns

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

**TD\_RoundColumns InputTable: titanic**

passenger	pclass	fare	survived
1	3	7.25	0
2	1	71.2833	1
3	3	7.925	1
4	1	53.100	1
5	3	8.050	0

**TD\_RoundColumns SQL Call**

```
SELECT * FROM TD_RoundColumns (
    ON titanic AS InputTable
    USING
        TargetColumns ('Fare')
        PrecisionDigit (1)
        Accumulate ('[0:1]', 'Survived')
) AS dt;
```

**TD\_RoundColumns Output**

passenger	pclass	survived	fare
1	3	0	7.30
2	1	1	71.30
3	3	1	7.90
4	1	1	53.10
5	3	0	8.10

**TD\_RoundColumns Column Examples**

You can also input multiple columns in the TargetColumns parameter to have these statistics calculated for all those columns at the same time. If you want stats for all columns, a shorter way to do this would be to add [:] in the TargetColumns parameter.

**titanic2: Input Table**

The following input table is used for these examples.

passenger	p_class	survived	fare	height_cms
5	3	0	8.10	137.90
3	3	1	7.90	155.10

passenger	p_class	survived	fare	height_cms
4	1	1	53.10	130.70
1	3	0	7.30	145.30
2	1	1	71.30	134.70

### Multiple Columns SQL Call

```
SELECT * FROM TD_RoundColumns (
    ON titanic2 AS InputTable
    USING
        TargetColumns ('fare','height_cms')
        PrecisionDigit (1)
        Accumulate ('[0:1]','survived')
) AS dt;
```

### Multiple Columns Output

passenger	p_class	survived	fare	height_cms
5	3	0	8.10	137.90
3	3	1	7.90	155.10
4	1	1	53.10	130.70
1	3	0	7.30	145.30
2	1	1	71.30	134.70

### All Columns SQL Call

```
SELECT * FROM TD_RoundColumns (
    ON titanic2 AS InputTable
    USING
        TargetColumns ('[:]')
        PrecisionDigit (1)
) AS dt;
```

### All Columns Output

passenger	p_class	fare	height_cms	survived
5	3	8.10	137.90	0
3	3	7.90	155.10	1

passenger	p_class	fare	height_cms	survived
4	1	53.10	130.70	1
1	3	7.30	145.30	0
2	1	71.30	134.70	1

## TD\_StrApply

TD\_StrApply is a string manipulation function that applies a regular expression pattern to a string and returns a new string with the matches replaced by a specified replacement string. This function supports several string operators such as Trimming, Padding, Upper, and so on.

For example, suppose you have a table called employees with a column called email, which contains email addresses in the format firstname.lastname@company.com. You want to replace ‘.’ character with a ‘\_’ character in all of the email addresses. You use the TD\_StrApply function to achieve this.

### Function Information

- [TD\\_StrApply Syntax](#)
- [Required Syntax Elements for TD\\_StrApply](#)
- [Optional Syntax Elements for TD\\_StrApply](#)
- [TD\\_StrApply Input](#)
- [TD\\_StrApply Output](#)
- [Example: How to Use TD\\_StrApply](#)

### Related Information:

[TD\\_NumApply](#)

## TD\_StrApply Syntax

```
TD_strApply (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    USING
        TargetColumns ({ 'target_column' | target_column_range }[,...])
        [ OutputColumns ('output_column' [,...]) ]
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
        StringOperation (str_operator)
        [ String ('string') ]
        [ StringLength ('length') ]
        [ OperatingSide ({ 'Left' | 'Right' })]
        [ IsCaseSpecific ({'true'|'t'||'yes'||'y'||'1'||'false'||'f'||'no'||'n'||'0'}) ]
        [ EscapeString ('escape_string') ]
```

```
[ IgnoreTrailingBlank ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ StartIndex ('start_index')]
[ InPlace ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_StrApply

### ON clause

Accepts the InputTable clause.

### TargetColumns

Specifies the name of the input table columns to apply the string operator.

### StringOperation

Specifies a string operator (str\_operator) from the following table. If str\_operator requires *string*, *length*, or *start\_index*, specify that value with String, StringLength, or StartIndex.

str_operator	Description
CHARTOHEXINT	Converts value to its hexadecimal representation.
GETNCHARS	Returns <i>length</i> characters from value. See <a href="#">Optional Syntax Elements for TD_StrApply</a> (OperatingSide).
INITCAP	Capitalizes first letter of value.
STRINGCON	Concatenates <i>string</i> to value.
STRINGINDEX	Returns index of first character of <i>string</i> in value. See <a href="#">Optional Syntax Elements for TD_StrApply</a> (IsCaseSpecific).
STRINGLIKE	Returns first string that matches specified pattern if one exists in value. See <a href="#">Optional Syntax Elements for TD_StrApply</a> (EscapeString, IsCaseSpecific, IgnoreTrailingBlank).
STRINGPAD	Pads value with <i>string</i> to <i>length</i> . See <a href="#">Optional Syntax Elements for TD_StrApply</a> (OperatingSide).
STRINGREVERSE	Reverses order of characters in value.

<i>str_operator</i>	Description
STRINGTRIM	If value contains <i>string</i> , trim <i>string</i> from value. See <a href="#">Optional Syntax Elements for TD_StrApply</a> (OperatingSide)..
SUBSTRING	Returns substring starting at <i>start_index</i> with <i>length</i> from value.
TOLOWER	Replaces uppercase letters in value with lowercase equivalents.
TOUPPER	Replaces lowercase letters in value with uppercase equivalents.
TRIMSPACES	Trims leading and trailing space characters from value.
UNICODESTRING	Converts LATIN value to UNICODE.

## Optional Syntax Elements for TD\_StrApply

### OutputColumns

[Ignored with Inplace ('true'), otherwise optional.] Specifies the name for the output columns.  
An *output\_column* cannot exceed 128 characters.

Default: With InPlace ('false'), *target\_column\_operator*; otherwise *target\_column*

### Note:

If any *target\_column\_operator* exceeds 128 characters, specify an *output\_column* for each *target\_column*.

### Accumulate

Specifies the names of the input table columns to copy to the output table.

With InPlace ('true'), no *target\_column* can be an *accumulate\_column*.

### String

[Required when *str\_operator* needs *string* argument, ignored otherwise.] Specifies the *string* argument for *str\_operator*.

<i>str_operator</i>	<i>string</i>
STRINGCON	String to concatenate to value.
STRINGINDEX	String for which to return index of its first character in value.
STRINGLIKE	Pattern that describes string to find in value.
STRINGPAD	String with which to pad value.

<i>str_operator</i>	<i>string</i>
STRINGTRIM	String to trim from value.

**StringLength**

[Required when *str\_operator* needs *length* argument, ignored otherwise.] Specifies the *length* argument for *str\_operator*:

<i>str_operator</i>	<i>length</i>
GETNCHARS	Number of characters to return from value.
STRINGPAD	Length to which to pad value.
SUBSTRING	Substring length.

**OperatingSide**

Applies only when *str\_operator* is GETNCHARS, STRINGPAD, or STRINGTRIM. Specifies side of value on which to apply *str\_operator*.

Default: left.

**IsCaseSpecific**

Applies only when *str\_operator* is STRINGINDEX or STRINGLIKE. Specify whether search for *string* is case-specific.

Default: true (search is case-specific).

**EscapeString**

Applies only when *str\_operator* is STRINGLIKE. Specify the escape characters.

**IgnoreTrailingBlank**

Applies only when *str\_operator* is STRINGLIKE. Specify whether to ignore trailing space characters.

**StartIndex**

Applies only when *str\_operator* is SUBSTRING. Specify the index of the character at which the substring starts.

**InPlace**

Specifies whether the output columns have the same names as the target columns.

InPlace ('true') effectively replaces each value in each target column with the result of applying *str\_operator* to it.

InPlace ('false') copies the target columns to the output table and adds output columns whose values are the result of applying *str\_operator* to each value.

With InPlace ('true'), no *target\_column* can be an *accumulate\_column*.

Default: true.

## TD\_StrApply Input

### Input Table Schema

Column	Data Type	Description
target_column	CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE)	Column to which to apply str_operator.
accumulate_column	Any	Column to copy to output table.

## TD\_StrApply Output

### Output Table Schema

Column	Data Type	Description
accumulate_column	Same as input table.	Column copied from input table.
output_column	Same as input table.	Column to which str_operator was applied. With InPlace ('true'), output_column is target_column. With InPlace ('false'), OutputColumns determines output_column.

## Example: How to Use TD\_StrApply

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_StrApply Input Table: input\_table

passenger	survived	pclass	name	gender	age	sibsp	parch	ticket	fare	cabin	embarked
5	0	3	Allan; Mr. William Henry	male	35	0	0	373450	8.050000000	null	S
4	1	1	Futrelle; Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.100000000	C123	S
3	1	3	Heikkinen; Miss. Laina	female	26	0	0	STON/O2. 3101282	7.925000000	null	S
1	0	3	Braund; Mr. Owen Harris	male	22	1	0	A/5 21171	7.250000000	null	S
2	1	1	Cumings; Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.283300000	C85	C

**Example: TD\_StrApply SQL Call**

```
SELECT * FROM TD_strApply (
ON strApply_input_table as InputTable PARTITION BY ANY
USING
TargetColumns ('Gender')
stringOperation ('toUpper')
Accumulate('Passenger')
InPlace('True')
) as dt order by 1;
```

**TD\_StrApply Output**

passenger	gender
1	MALE
2	FEMALE
3	FEMALE
4	FEMALE
5	MALE

# Model Training Functions

- [TD\\_DecisionForest](#)
- [TD\\_GLM](#)
- [TD\\_KMeans](#)
- [TD\\_KNN](#)
- [TD\\_NaiveBayes](#)
- [TD\\_OneClassSVM](#)
- [TD\\_SVM](#)
- [TD\\_VectorDistance](#)
- [TD\\_XGBoost](#)

## TD\_DecisionForest

Decision forest functions create predictive models based on the algorithm for decision tree training and prediction.

TD\_DecisionForest function is an ensemble algorithm used for classification and regression predictive modeling problems. It is an extension of bootstrap aggregation (bagging) of decision trees. The function supports regression, binary, and multiclass classification.

Constructing a decision tree involves evaluating the value for each input variable in the data to select a split point. The function reduces the variables to a random subset that can be considered at each split point. The algorithm can force each decision tree in the forest to be different to improve prediction accuracy.

Each node in the tree represents a decision based on the value of a single variable, and the tree is grown by iteratively splitting the data into smaller and smaller subsets based on these decisions. It repeats this process until it finds the best variable to split the data at a given level of a tree, and repeats it at each level until the stopping criterion is met.

Consider the following points when using TD\_DecisionForest function:

- All input variables are numeric. Convert the categorical columns to numerical columns as preprocessing step.
- For classification, class labels (ResponseColumn values) can only be integers. Supports a maximum of 500 classes for classification.
- The function skips any observation with a missing value in an input column and is not used for training. Use TD\_SimpleImpute function to assign missing values.

TD\_DecisionForest has several parameters that you can tune to optimize performance, including the number of trees, the maximum depth of each tree, and the minimum number of samples required to split a node. TD\_DecisionForest constructs the trees in parallel by all the AMPs, which have a non-empty partition of data.

- When you specify the NumTrees value, TD\_DecisionForest adjusts the number of trees built as:

```
Number_of_trees = Num_AMPs_with_data * (NumTrees/Num_AMPs_with_data)
```

- For Num\_AMPs\_with\_data value, use the SQL command `SELECT HASHAMP() + 1;`.
- When you do not specify the NumTrees value, TD\_DecisionForest calculates the number of trees built by an AMP as:

```
Number_of_AMP_trees = CoverageFactor * Num_Rows_AMP / TreeSize
```

The number of trees built by the function is the sum of Number\_of\_AMP\_trees.

**Note:**

When a data set is small, best practice is to distribute the data to one AMP. To do this, create an identifier column as a primary index, and use the same value for each row.

- The TreeSize value determines the sample size used to build a tree in the forest and depends on the memory available to the AMP. By default, TD\_DecisionForest computes internally this value. TD\_DecisionForest reserves approximately 40% of its available memory to store the input sample, while the rest is used to build the tree.

**Note:**

The number of trees controls the processing time and complexity of the trees. For example, changing CoverageFactor from 1.0 to 2.0 doubles the number of trees and increases processing time of the query.

TD\_DecisionForest uses a training dataset to create a predictive model. TD\_DecisionForestPredict function uses the model created by TD\_DecisionForest function for making predictions.

See [TD\\_DecisionForestPredict](#).

The following is an example of how to use TD\_DecisionForest:

- Convert the categorical columns to numerical columns.
- Determine the parameters to use with the function, such as tree depth, model type, and number of trees.
- Use TD\_DecisionForest on a training dataset to create a predictive model.
- Use [TD\\_DecisionForestPredict](#) function on the model created by the TD\_DecisionForest function to make predictions.

## Function Information

- [TD\\_DecisionForest Syntax](#)
- [Required Syntax Elements for TD\\_DecisionForest](#)
- [Optional Syntax Elements for TD\\_DecisionForest](#)
- [TD\\_DecisionForest Input](#)
- [TD\\_DecisionForest Output](#)

- [TD\\_DecisionForest Usage Notes](#)
- [TD\\_DecisionForest Examples](#)

## TD\_DecisionForest Syntax

```
TD_DecisionForest (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  USING
    InputColumns ({'input_column' | input_column_range }[,...])
    ResponseColumn('response_column')
    [ MaxDepth (maxdepth) ]
    [ MinNodeSize (minnodesize) ]
    [ NumTrees (numtrees) ]
    [ ModelType ('classification'|'regression') ]
    [ TreeSize (treesize) ]
    [ CoverageFactor (coveragefactor) ]
    [ Seed (seed) ]
    [ Mtry (mtry) ]
    [ MtrySeed (mtryseed) ]
    [ MinImpurity (minimpurity) ]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_DecisionForest

The following elements are required when using TD\_DecisionForest:

### ON clause

Specify the table name, view name or query as an InputTable.

### InputColumns

Specify the input table column names for training the model (predictors, features, or independent variables).

---

**Note:**

Input column names with double quotation marks are not allowed for this function.

---

**ResponseColumn**

Specify the column name that contains the classification label or target value (dependent variable) for regression.

## Optional Syntax Elements for TD\_DecisionForest

The following elements are optional when using TD\_DecisionForest:

**MaxDepth**

Specify the maximum depth of a tree. The algorithm stops splitting a node beyond this depth. Decision trees can grow to  $2^{(\max\_depth+1)} - 1$  nodes. You must specify a non-negative integer value.

Default value: 5

**MinNodeSize**

Specify the minimum number of observations in a tree node. The algorithm stops splitting a node if the number of observations in the node is equal to or smaller than this value. You must specify a non-negative integer value.

Default value: 1.

**NumTrees**

Specify the number of trees for the forest model. You must specify a value greater than or equal to the number of data AMPs. By default, the function builds the minimum number of trees that provides the specified coverage level in the CoverageFactor argument for the input dataset.

---

**Note:**

Maximum number of supported trees is 65536.

---

Default value: -1.

**ModelType**

Specify whether the analysis is a regression (continuous response variable) or a multiple-class classification (predicting result from the number of classes).

Allowed values: Regression or Classification.

When using classification, the TD\_DecisionForest function generates tree models only when more than one class is distributed to the AMP. If the AMP only has one class, no tree will be built.

---

**Note:**

A maximum of 500 classes is supported for classification.

---

Default value: Regression.

**TreeSize**

Specify the number of rows that each tree uses as its input dataset. The function builds a tree using either the number of rows on an AMP, the number of rows that fit into the AMP's memory (whichever is less), or the number of rows given by the TreeSize argument. By default, this value is the minimum number of rows on an AMP and the number of rows that fit into the AMP's memory.

Default value: -1.

**CoverageFactor**

Specify the level of coverage for the dataset in the forest. The value is specified in percentage. The default coverage value is 1.0 (100%).

**Seed**

Specify the random seed the algorithm uses for repeatable results.

Default value: 1.

**Mtry**

Specify the number of features from input columns for evaluating the best split of a node. A higher value improves the splitting and performance of a tree. A smaller value improves the robustness of the forest and prevents it from overfitting. When the value is -1, all variables are used for each split.

Default value: -1.

**MtrySeed**

Specify the random seed that the algorithm uses for the Mtry argument.

Default value: 1.

**MinImpurity**

Specify the minimum impurity of a tree node. The algorithm stops splitting a node if the value is equal to or smaller than the specified value.

Default value: 0.0.

## TD\_DecisionForest Input

Column Name	Data Type	Description
input_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, or NUMBER	The columns that the function uses to train the DecisionForest model.
response_column	INTEGER, SMALLINT, BYTEINT, FLOAT, DECIMAL, or NUMBER	The column that contains the response value for an observation. For regression, all numeric data types are supported. For classification, INTEGER, SMALLINT, and BYTEINT data types are supported.

## TD\_DecisionForest Output

The function produces a model and a JSON representation of the decision tree. The model output is as follows:

Column Name	Data Type	Description
task_index	SMALLINT	The AMP that produces the decision tree.
tree_num	SMALLINT	The identified decision tree within an AMP.
regression_tree or classification_tree	VARCHAR	The JSON representation of decision tree. If the JSON string is larger than 32000 bytes, then it is truncated and displayed in several rows associated with a tree_order id.

The JSON representation of the decision tree has the following elements:

JSON Type	Description
id_	The unique identifier of the node.
sum_	[Regression trees] The sum of response variable values in the node.
sumSq_	[Regression trees] The sum of squared values of the response variable in the node.
responseCounts_	[Classification trees] The number of observations in each class of the node.
size_	The total number of observations in the node.
maxDepth_	The maximum possible depth of the tree starting from the current node. For the root node, the value is max_depth. For leaf nodes, the value is 0. For other nodes, the value is the maximum possible depth of the tree, starting from that node.
split_	The start of JSON item that describes a split in the node.
score_	The GINI score of the node.

JSON Type	Description
attr_	The attribute on which the algorithm splits the node.
type_	Type of tree and split. Possible values: <ul style="list-style-type: none"><li>• CLASSIFICATION_NUMERIC_SPLIT</li><li>• REGRESSION_NUMERIC_SPLIT</li></ul>
leftNodeSize_	The number of observations assigned to the left node of the split.
rightNodeSize_	The number of observations assigned to the right node of the split.
leftChild_	The start of the JSON item that describes the left child of the node.
rightChild_	The start of the JSON item that describes the right child of the node.
nodeType_	The node type. Possible values: <ul style="list-style-type: none"><li>• CLASSIFICATION_NODE</li><li>• CLASSIFICATION_LEAF</li><li>• REGRESSION_NODE</li><li>• REGRESSION_LEAF</li></ul>

## TD\_DecisionForest Usage Notes

A decision forest is a machine learning model that is composed of multiple decision trees. A decision tree is a hierarchical structure that makes decisions by recursively splitting the data into subsets based on the values of input variables.

The ways a tree splitting a node dependent on a target variable:

Continuous target variable: reduction in variance.

1. For each split, individually calculate the variance of each child node
2. Calculate the variance of each split as the weighted average variance of child nodes
3. Select the split with the lowest variance
4. Repeat steps 1-3 until you achieve homogeneous nodes

smaller the variance, the better the split.

Categorical target variable: information gain

1. For each split, individually calculate the entropy of each child node
2. Calculate the entropy of each split as the weighted average entropy of child nodes
3. Select the split with the lowest entropy or highest information gain
4. Repeat steps 1-3 until you achieve homogeneous nodes

The closer to 1 information gain is, the better the split.

The decision forest algorithm builds an ensemble of decision trees, where the algorithm trains each tree on a randomly sampled subset of the training data and a randomly selected subset of the variables. This randomness helps to prevent overfitting and improve the generalization of the model.

The decision forest algorithm makes the final prediction by aggregating the predictions of all the trees in the forest. You can do this in different ways, such as taking the average prediction for regression problems or taking the majority vote for classification problems.

You can represent the decision function of a decision forest as follows:

where:

- T is the number of trees in the forest
- $f_i(x)$  is the prediction of the  $i$ th tree for the input data point  $x$
- $f(x)$  is the final prediction of the forest for the input data point  $x$

You can represent each tree in the forest by a set of IF, THEN rules that recursively split the input space into subsets based on the values of the input variables.

You can represent the decision rule for a node  $j$  in the  $i$ th tree as follows:

Where:

- $x_{i,k}$  is the value of the  $k$ th variable of the input data point
- $t_{j,k}$  is the threshold value for the  $k$ th variable at node  $j$
- The left and right children of node  $j$  correspond to the subsets of the input space where  $x_{i,k} < t_{j,k}$  and  $x_{i,k} \geq t_{j,k}$ , respectively

You can represent the prediction function for a leaf node  $j$  in the  $i$ th tree as follows:

where  $c_j$  is the class label or target value associated with leaf node  $j$

Some of the advantages of decision forest includes:

- Robustness and Generalization: Decision forests can be less prone to overfitting and can generalize well to new data. This is because they combine multiple decision trees, which can help to reduce variance and improve stability of the predictions.
- Handles high dimensional data: Decision forests can handle high dimensional data, which is common in real world applications such as image recognition, natural language processing, and bioinformatics.
- Scalability: Decision forests are relatively fast to train and can handle large datasets, making them suitable for big data applications.
- Nonlinear relationships: Decision forests can model nonlinear relationships between the input features and the target variable, which can be useful when there are complex interactions and dependencies in the data.
- Parallelism: Decision forests can be trained in parallel, which can significantly reduce the training time and improve efficiency.

Some of the disadvantages of decision forest Includes:

- Complexity and Interpretability: Decision forests can be complex and difficult to interpret and explain due to the large number of trees and the complexity of their interactions. This can be a disadvantage in situations where interpretability is important, such as in medical diagnosis or legal decision making.
- Hyperparameter Tuning: Decision forests have several hyperparameters that you need to tune, such as the number of trees, the depth of the trees, and the number of variables to consider at each split. Tuning these hyperparameters can be time consuming and requires domain expertise.

- Bias-Variance Trade-off: Decision forests can still be prone to overfitting if the number of trees is too small or underfitting if the number of trees is too large. Finding the right balance between bias and variance can be challenging and requires careful tuning of the hyperparameters.
- Memory Usage: Decision forests can require significant memory to store and maintain the multiple trees and their associated data structures. This can be a limitation in memory constrained environments such as mobile devices or embedded systems.

In summary, a decision forest is a machine learning model that combines multiple decision trees to make predictions. The decision function of the forest is a weighted average of the predictions of all the trees, and each tree is represented by a set of IF, THEN rules that recursively split the input space into subsets based on the values of the input variables.

## TD\_DecisionForest Examples

### Example: TD\_DecisionForest Classification

Input table

encoded	ROW_I	attribute_1	attribute_2	attribute_3	...	attribute_49	sample_id
0	99	-0.0664	-0.0999	-0.0949	...	-0.0942	2
0	101	-0.0603	-0.0938	-0.0900	...	-0.0935	2
1	114	0.0000	0.0001	0.0001	...	0.0001	2
1	115	0.0001	0.0001	0.0001	...	0.0001	2
...	...	...	...	...	...	...	...

### TD\_DecisionForest Call for Classification

```
CREATE VOLATILE TABLE DecisionForestOutput AS (
SELECT * FROM TD_DecisionForest(
    ON DT_Input AS PARTITION BY ANY
USING
    ResponseColumn('encoded')
    InputColumns('[2:12]')
    TreeType('CLASSIFICATION')
) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;
```

### TD\_DecisionForest Output for Classification

```
SELECT * FROM DecisionForestOutput
```

task_index	tree_num	tree_order	tree
24	0	0	{"id":1,"size":75,"maxDepth":5,"label":"1","responseCounts":{"1":75},"nodeType":"CLASSIFICATION_LEAF"}
35	0	0	{"id":1,"size":82,"maxDepth":5,"label":0,"responseCounts":{"0":82},"nodeType":"CLASSIFICATION_LEAF"}

### Example: TD\_DecisionForest Regression

The following is a sample of housing data taken from Boston housing dataset.

CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	medv
0.05188	0.0	4.49	0.0	0.449	6.015	45.1	4.4272	3.0	247	18.5	396.99	12.86	22.5
0.30347	0.0	7.38	0.0	0.493	6.312	28.9	5.4159	5.0	287	19.6	396.90	6.15	23.0
0.6147	0.0	6.20	0.0	0.507	6.618	80.8	3.2721	8.0	307	17.4	396.90	7.6	30.1
0.04527	0.0	11.93	0.0	0.537	6.120	76.7	2.2875	1.0	273	21.0	396.90	9.08	20.6
0.12816	12.5	6.07	0.0	0.409	5.885	33	6.4890	4.0	345	18.9	396.90	8.79	20.9
...	...	...	...	...	...	...	...	...	...	...	...	...	...

### TD\_DecisionForest SQL Call for Regression

```

SELECT * FROM TD_DecisionForest (
ON housing_sample AS inputtable PARTITION BY ANY
USING
    ResponseColumn('medv')
    InputColumns('[0:12]')
    MaxDepth(12)
    MinNodeSize(1)
    NumTrees(4)
    ModelType('REGRESSION')
    Seed(1)
    Mtry(3)
    MtrySeed(1)
) AS dt;

```

## TD\_DecisionForest Output for Regression

task_index	tree_num	regression_tree
0	0	{"id_":1,"sum_":201.700000,"sumSq_":6781.890000,"size_":6,"maxDepth_":12,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":7.091500,"attr_":"rm","type_":"REGRESSION_NUMERIC_SPLIT","score_":32984.915253,"scoreImprove_":32984.915253,"leftNodeSize_":5,"rightNodeSize_":1},"leftChild_":{"id_":2,"sum_":167.000000,"sumSq_":5577.800000,"size_":5,"maxDepth_":11,"value_":33.400000,"nodeType_":"REGRESSION_LEAF"},"rightChild_":{"id_":3,"sum_":34.700000,"sumSq_":1204.090000,"size_":1,"maxDepth_":11,"value_":34.700000,"nodeType_":"REGRESSION_LEAF"}}
2	0	{"id_":1,"sum_":208.800000,"sumSq_":4905.980000,"size_":9,"maxDepth_":12,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":6.465000,"attr_":"rm","type_":"REGRESSION_NUMERIC_SPLIT","score_":37076.368050,"scoreImprove_":37076.368050,"leftNodeSize_":8,"rightNodeSize_":1},"leftChild_":{"id_":2,"sum_":178.700000,"sumSq_":3999.970000,"size_":8,"maxDepth_":11,"value_":22.337500,"nodeType_":"REGRESSION_LEAF"},"rightChild_":{"id_":3,"sum_":30.100000,"sumSq_":906.010000,"size_":1,"maxDepth_":11,"value_":30.100000,"nodeType_":"REGRESSION_LEAF"}}
3	0	{"id_":1,"sum_":93.600000,"sumSq_":2194.560000,"size_":4,"maxDepth_":12,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":7.060000,"attr_":"lstat","type_":"REGRESSION_NUMERIC_SPLIT","score_":6272.052528,"scoreImprove_":6272.052528,"leftNodeSize_":3,"rightNodeSize_":1},"leftChild_":{"id_":2,"sum_":72.000000,"sumSq_":1728.000000,"size_":3,"maxDepth_":11,"value_":24.000000,"nodeType_":"REGRESSION_LEAF"},"rightChild_":{"id_":3,"sum_":21.600000,"sumSq_":466.560000,"size_":1,"maxDepth_":11,"value_":21.600000,"nodeType_":"REGRESSION_LEAF"}}

## TD\_GLM

TD\_GLM function is a generalized linear model (GLM) that performs regression and classification analysis on data sets, where the response follows an exponential family distribution and supports the following models:

- Regression (Gaussian family): The loss function is squared error.
- Binary Classification (Binomial family): The loss function is logistic and implements logistic regression. The response values are 0 or 1.

GLMs are a flexible class of statistical models that extend the linear regression framework to accommodate a wide range of response variables, including binary, count, and continuous data. GLMs assume the response variable has a probability distribution from an exponential family of distributions, which includes commonly-used distributions such as the normal, binomial, and Poisson distributions.

GLMs consist of the following key components:

- Linear predictor: A predictor variables and their coefficients, similar to linear regression. It uses predictor variables  $X$ , and their coefficients  $\beta$ , and  $\eta = X\beta$ .

- Link function: The relationship of the linear predictor to the mean of the response variable, allowing for non-linear relationships between the predictors and response. It uses the link function  $g$  for  $g(\mu) = \eta$ .
- Probability distribution: The variability of the response variable, and is chosen based on the nature of the data. The variance is calculated as  $\text{Var}(Y) = \varphi V(\mu)$ , where  $\varphi$  is a scale parameter, and  $V(\mu)$  is the variance function.

GLMs are fitted using maximum likelihood estimation, which involves finding the parameter values that maximize the likelihood of observing the data given the model. Model fit can be assessed using various goodness-of-fit measures, such as deviance or Pearson chi-squared statistics.

By specifying the appropriate link and variance functions, GLMs can be used to model a wide range of response variables. For example, the logistic regression model for binary data has the following components:

- Probability distribution: Bernoulli distribution
- Linear predictor:  $\eta = X\beta$
- Link function: logit ( $g(\mu) = \text{logit}(\mu) = \log(\mu/(1-\mu))$ )

Similarly, the Poisson regression model for count data has the following components:

- Probability distribution: Poisson distribution
- Linear predictor:  $\eta = X\beta$
- Link function: log ( $g(\mu) = \log(\mu)$ )
- Variance function:  $\text{Var}(Y) = \mu$

`TD_GLM` uses the Minibatch Stochastic Gradient Descent (SGD) algorithm. The algorithm estimates the gradient of loss in minibatches, which is defined by the `BatchSize` argument and updates the model with a learning rate using the `LearningRate` argument.

The function also supports the following approaches:

- L1, L2, and Elastic Net Regularization for shrinking model parameters
- Accelerated learning using Momentum and Nesterov approaches

`TD_GLM` uses a combination of `IterNumNoChange` and `Tolerance` arguments to define the convergence criterion and runs multiple iterations (up to the specified value in the `MaxIterNum` argument) until the algorithm meets the criterion. The function also supports `LocalSGD`, a variant of SGD, that uses `LocalSGDIterations` on each AMP to run multiple batch iterations locally followed by a global iteration. The weights from all mappers are aggregated in a reduce phase to compute the gradient and loss in the next iteration. `LocalSGD` lowers communication costs and can result in faster learning and convergence in fewer iterations, especially when there is a large cluster size and many features.

Due to gradient-based learning, `TD_GLM` is highly sensitive to feature scaling. Before using the features in the function, you must standardize the Input features using [TD\\_ScaleFit](#) and [TD\\_ScaleTransform](#).

`TD_GLM` only accepts numeric features. Therefore, before training, you must convert the categorical features to numeric values. The function skips the rows with missing (null) values during training.

The function output is a trained GLM model that is used as input to the [TD\\_GLMPredict](#) function. The output contains model statistics of MSE, Loglikelihood, AIC, and BIC. You can use [TD\\_RegressionEvaluator](#), [TD\\_ClassificationEvaluator](#), and [TD\\_ROC](#) functions to perform model evaluation as a post-processing step.

The function skips the rows with missing (null) values during training. You can use an imputation functions, such as TD\_SimpleImputeFit and TD\_SimpleImputeTransform to do imputation of missing values.

The TD\_GLM function is used to train the whole data set as one model or each data partition as a single mode. To train the whole data set, specify partition by any function in the ON clause. To train each data partition, specify partition by key function in the ON clause.

A model generated from partition-by-any can be matched with a model from partition-by-key if all input data for partition-by-any is on a single AMP and the BatchSize value is greater than or equal to the number of rows in input.

GLMs are used for regression analysis, classification, and survival analysis. They have applications in fields such as medicine, biology, economics, and engineering.

## Function Information

- [TD\\_GLM Syntax](#)
- [Required Syntax Elements for TD\\_GLM](#)
- [Optional TD\\_GLM Syntax Elements](#)
- [TD\\_GLM Input](#)
- [TD\\_GLM Output](#)
- [Examples: How to Use TD\\_GLM](#)

## Using Partition by Any

When it is used as partition-by-any function to train the whole data sets as a single model, the function also supports LocalSGD, a variant of SGD, that uses LocalSGDIterations on each AMP to run multiple batch iterations locally followed by a global iteration.

The weights from all mappers are aggregated in a reduce phase and are used to compute the gradient and loss in the next iteration. LocalSGD lowers communication costs and can result in faster learning and convergence in fewer iterations, especially when there is a large cluster size and many features.

The function output is trained GLM models that is used as input to the TD\_GLMPredict function. The output contains model statistics of MSE, Loglikelihood, AIC, and BIC. You can use [TD\\_RegressionEvaluator](#), [TD\\_ClassificationEvaluator](#), and [TD\\_ROC](#) functions to perform model evaluation as a post processing step.

## Using Partition by Key (Micromodeling)

When it is used as partition-by-key function to train each partition as a single model, the function accepts the following inputs:

- InputTable is required, containing an input dataset to be used for training GLM per segment model.

- AttributeTable is optional, defining a subset of features to be used with respect to each partition.
- ParameterTable is optional, defining a subset of parameters to be used with respect to each partition.

The ORDER BY clause can be optionally applied to the InputTable to guarantee the result in each run is deterministic. The situation of indeterministic result in a partition can occur if the BatchSize argument is less than the number of rows in the partition. Adding the ORDER BY clause ensures the result is deterministic regardless of the BatchSize settings. The ORDER BY clause can affect performance.

## Function Information

- [TD\\_GLM Syntax](#)
- [Required Syntax Elements for TD\\_GLM](#)
- [Optional TD\\_GLM Syntax Elements](#)
- [TD\\_GLM Input](#)
- [TD\\_GLM Output](#)
- [Examples: How to Use TD\\_GLM](#)

## TD\_GLM Syntax

### TD\_GLM Syntax Using Partition by Any

---

#### Important:

In Analytics Database Release 17.20.03.14 and later, the AS InputTable alias is mandatory. If your scripts does not use an alias, use AS InputTable alias for the input table.

```
TD_GLM (
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY ]
    [ OUT TABLE MetaInformationTable (meta_table) ]
    USING
        InputColumns ({'input_column' | 'input_column_range' }[,...])
        ResponseColumn('response_column')
        [ Family ('Gaussian' | 'Binomial') ]
        [ BatchSize (batchsize) ]
        [ MaxIterNum (max_iter) ]
        [ RegularizationLambda (lambda) ]
        [ Alpha (alpha) ]
        [ IterNumNoChange (n_iter_no_change) ]
        [ Tolerance (tolerance) ]
        [ Intercept ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
        [ ClassWeights ('class:weight,...') ]
        [ LearningRate ('constant' | 'optimal' | 'invtime' | 'adaptive' ) ]
        [ InitialEta (eta0) ]
        [ DecayRate (gamma) ]
```

```
[ DecaySteps (decay_steps) ]
[ Momentum (momentum) ]
[ Nesterov ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ LocalSGDIterations(local_iterations) ]
[ StepwiseDirection ('forward'|'backward'|'both' | 'bidirectional')
[ MaxStepsNum (max_stepwise_steps) ]
[ InitialStepwiseColumns ({initialstate_columns |
initialstate_column_range }[,...])]
)
```

## TD\_GLM Syntax Using Partition by Key

```
TD_GLM (
ON { table | view | (query) } AS InputTable PARTITION BY partition_by_column [ ORDER
BY id_column ]
[ ON { table | view | (query) } AS AttributeTable PARTITION BY partition_by_column ]
[ ON { table | view | (query) } AS ParameterTable PARTITION BY partition_by_column ]
USING
InputColumns({'input_column'| input_column_range}{...})
ResponseColumn (response_column)
[ PartitionColumn ('partition_column') ]
[ Family ('Gaussian' | 'Binomial') ]
[ BatchSize (batchsize) ]
[ MaxIterNum (max_iter) ]
[ RegularizationLambda (lambda) ]
[ Alpha (alpha) ]
[ IterNumNoChange (n_iter_no_change) ]
[ Tolerance (tolerance) ]
[ Intercept ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ ClassWeights ('class:weight,...') ]
[ LearningRate ('constant'|'optimal'|'invtime'|'adaptive') ]
[ InitialEta (eta0) ]
[ DecayRate (gamma) ]
[ DecaySteps (decay_steps) ]
[ Momentum (momentum) ]
[ Nesterov ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
[ IterationMode ('batch'|'epoch')]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_GLM

**ON clause**

Accepts the PARTITION BY ANY clause.

**OUT clause**

Accepts the TABLE clause.

**InputColumns**

Specify the input table column names for training the model (predictors, features or independent variables).

**ResponseColumn**

Specify the column name containing the class label for classification or target value (dependent variable) for regression.

## Optional TD\_GLM Syntax Elements

**PartitionColumn**

Name of the InputTable columns on which to partition the input. The name must be consistent with the partition\_by\_column in the ON clause. If the partition\_by\_column is unicode with foreign language characters, then it is necessary to specify PartitionColumn argument.

**Family**

Specify the distribution exponential family. Options are Gaussian and Binomial. Default value is Gaussian.

**BatchSize**

Specify the number of observations (training samples) processed in a single minibatch per AMP. A value of 0 or higher than the number of rows on an AMP processes all rows on the AMP, such that the entire dataset is processed in a single iteration, and the algorithm becomes Gradient Descent. Specify a non-negative integer value. The default value is 10.

**MaxIterNum**

Specify the maximum number of iterations (minibatches) over the training data batches. Value is a positive integer less than 10,000,000. Default value is 300.

**RegularizationLambda**

Specify the regularization amount. The higher the value, stronger the regularization. It is also used to compute learning rate when learning rate is set to optimal. Must be a non-negative float value. A value of 0 means no regularization. Default value is 0.02.

**Alpha**

Specify the Elasticnet parameter for penalty computation. It is only effective when RegularizationLambda is greater than 0. The value represents the contribution ratio of L1 in the penalty. A value of 1.0 indicates L1 (LASSO) only, a value of 0 indicates L2 (Ridge) only, and a value between is a combination of L1 and L2. Value is a float value between 0 and 1. The default value is 0.15 (15% L1, 85% L2).

**IterNumNoChange**

Specify the number of iterations (minibatches) with no improvement in loss including the tolerance to stop training. A value of 0 indicates no early stopping and the algorithm continues until MaxIterNum iterations are reached. Specify a non-negative integer value. The default value is 50.

**Tolerance**

Specify the stopping criteria in terms of loss function improvement. Applicable when IterNumNoChange is greater than 0. Specify a positive integer value. The default value is 0.001.

**Intercept**

Specify whether to estimate intercept based on whether the data is already centered. The default value is true.

**ClassWeights**

Specify weights associated with classes. Only applicable for Binomial Family. The format is 0:weight,1:weight. For example, 0:1.0,1:0.5 gives twice the weight to each observation in class 0. If the weight of a class is omitted, it is assumed to be 1.0. The default value is 0:1.0,1:1.0.

**LearningRate**

Specify one of the learning rate algorithms:

- Constant
- InvTime

- Optimal
- Adaptive

The default value is invtime for Gaussian, and optimal for Binomial.

#### **InitialEta**

Specify the initial learning rate eta value. If you specify the learning rate as constant, the eta value is applicable for all iterations. The default value is 0.05.

#### **DecayRate**

Specify the decay rate for the learning rate. Only applicable for invtime and adaptive learning rates. The default value is 0.25.

#### **DecaySteps**

Specify the number of iterations without decay for the adaptive learning rate. The learning rate changes by decay rate after the specified number of iterations are completed. The default value is 5.

#### **Momentum**

Specify the value to use for momentum learning rate optimizer. A larger value indicates higher momentum contribution. A value of 0 means momentum optimizer is disabled. For a good momentum contribution, a value between 0.6-0.95 is recommended. Value is a non-negative float between 0 and 1. The default value is 0.

#### **Nesterov**

Specify whether to use Nesterov optimization for the Momentum optimizer. Only applicable when the Momentum optimizer value is greater than 0. The default value is True.

#### **LocalSGDIterations**

Specify the number of local iterations for the Local SGD algorithm. A value of 0 implies that the algorithm is disabled. A value greater than 0 enables the algorithm and specifies the number of iterations for the algorithm. The recommended values for the arguments are as follows:

- LocalSGDIterations: 10
- MaxIterNum: 100
- BatchSize: 50
- IterNumNoChange: 5

The default value is 0.

**StepwiseDirection**

Specify the type of algorithm to be used. The algorithm stops when the addition or deletion of variables in the model does not improve the score. The following arguments are available:

- forward

Forward selection starts with an empty model and adds features one at a time, keeping the variable that provides the best model. Once a feature has been added to the model it cannot be removed.

- backward

Backward elimination starts with a full model with all the features to be tested. In each step, a single feature is removed from the model keeping only those that improve the selected criterion. Once a feature has been removed it cannot be added to the model.

- both or bidirectional

Both and bidirectional behave the same way. The function starts from a given model, which can be full, empty, or any combination of variables. The features inside the model are deleted and those outside the model are added one at a time, selecting in each step the combination that improves the most the score of the selected criterion.

**MaxStepsNum**

Specify the maximum number of steps to be used for the StepwiseDirection algorithm. If the value is set to zero, then the algorithm runs until convergence. Default is 5.

**InitialStepwiseColumns**

Specify the names of the initial state model columns that need to be used as starting point for the stepwise regression algorithm, such as predictors, features or independent variables.

**IterationMode**

The iteration mode. Values are Batch and Epoch. Default is Batch.

- Batch: One iteration per batch. After processing rows in a batch, update the weight of the parameters and proceed to the next iteration.
- Epoch: One iteration per epoch. After processing all rows in a partition (with one or more batches), update the weight of the parameters and proceed to the next epoch. See BatchSize argument for batch size.

**TD\_GLM Input**

Column	Data Type	Description
partition_by_column	CHARACTER, VARCHAR, INTEGER, BIGINT, SMALLINT, BYTEINT	Unique model identifier.

Column	Data Type	Description
id_column	CHARACTER, VARCHAR, INTEGER, BIGINT, SMALLINT, BYTEINT	Unique identifier for each row.
input_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	The input table columns used to train the GLM model.
response_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	The column that contains the response value for an observation.

**AttributeTable Schema**

Column	Data Type	Description
partition_by_column	CHARACTER, VARCHAR, INTEGER, BIGINT, SMALLINT, BYTEINT	Unique model identifier.
attribute_column	VARCHAR	<p>Names of the target columns in the input table.</p> <p><b>Note:</b> If the column contains duplicate values, then the function displays an error message. The column name must be attribute_column.</p>

**ParameterTable Schema**

Column	Data Type	Description
partition_by_column	CHARACTER, VARCHAR, INTEGER, BIGINT, SMALLINT, BYTEINT	Unique model identifier.
parameter_column	VARCHAR	<p>One of the following syntax elements:</p> <ul style="list-style-type: none"> <li>• Family</li> <li>• BatchSize</li> <li>• MaxIterNum</li> <li>• RegularizationLambda</li> <li>• Alpha</li> <li>• IterNumNoChange</li> <li>• Tolerance</li> <li>• Intercept</li> <li>• ClassWeights</li> <li>• LearningRate</li> <li>• InitialEta</li> </ul>

Column	Data Type	Description
		<ul style="list-style-type: none"> <li>• DecayRate</li> <li>• DecaySteps</li> <li>• Momentum</li> <li>• Nesterov</li> </ul> The column name must be parameter_column.
value_column	VARCHAR	Value to use for syntax elements listed in parameter_column.

## TD\_GLM Output

TD\_GLM produces the following outputs:

- Model (Primary output): Contains the trained model with model statistics. The following model statistics are stored in the model:
  - Loss Function
  - MSE (Gaussian)
  - Loglikelihood (Logistic)
  - Number of Observations
  - AIC
  - BIC
  - Number of Iterations
  - Regularization
  - Alpha (L1/L2/Elasticnet)
  - Learning Rate (initial)
  - Learning Rate (Final)
  - Momentum
  - Nesterov
  - LocalSGD Iterations (Partition by Any only)
- [Optional for PARTITION BY ANY] MetaInformationTable (Secondary Output): Contains training progress information for each iteration. When the StepwiseDirection parameter is specified, the secondary output table contains information for the Stepwise Regression algorithm.

The model output schema for partition by any is as follows:

**Output Schema for PARTITION BY ANY**

Column	Data Type	Description
attribute	SMALLINT	Numeric index of predictor and model metrics. Intercept is specified using index 0, and the rest of the predictors take positive values. Model metrics take negative indices.
predictor	VARCHAR	Name of the predictor or model metric.

Column	Data Type	Description
estimate	FLOAT	Predictor weights and numeric-based metric values.
value	VARCHAR	String-based metric values such as SQUARED_ERROR for LossFunction, L2 for Regularization, and so on.

[Optional for PARTITION BY ANY] The MetaInformationTable output schema is as follows:

#### MetaInformationTable Output Schema for PARTITION BY ANY

Column	Data Type	Description
iteration	INTEGER	Iteration number.
num_rows	BIGINT	Total number of rows processed.
eta	FLOAT	Learning rate for the iteration.
loss	FLOAT	Loss in the iteration.
best_loss	FLOAT	Best loss until the specified iteration.
Step	INTEGER	[StepwiseDirection only] Iteration number (step number).
SubStep	INTEGER	[StepwiseDirection only] Feature number to be added or deleted. Non-feature numbers are given to different algorithm stages for sorting purposes.
Description	VARCHAR	[StepwiseDirection only] Description of the stages of each step. Added features are preceded by a plus sign, and deleted features by a minus sign.
Score	FLOAT	[StepwiseDirection only] Score of a given model tested in each substep, as well as the best score in each step and the best overall score, as indicated by the Description column.
Model	VARCHAR	[StepwiseDirection only] List of variable names that are contained in the model at each step.

The output schema for partition by key is as follows:

#### Output Schema for Partition by Key

Column	Data Type	Description
partition_by_column	CHARACTER, VARCHAR, INTEGER, BIGINT, SMALLINT, BYTEINT	Data type is the same as the original column in the input table.
attribute	SMALLINT	Numeric index of predictor and model metrics. Intercept is specified using index 0, and the rest of the predictors take positive values. Model metrics take negative indices.
predictor	VARCHAR	Name of the predictor or model metric.
estimate	FLOAT	Predictor weights and numeric-based metric values.

Column	Data Type	Description
value	VARCHAR	String-based metric values, such as SQUARED_ERROR for LossFunction, L2 for Regularization, and so on.

## Examples: How to Use TD\_GLM

### Input Table for train\_dataset

encoded	ROW_I	attribute_1	attribute_2	attribute_3	....	attribute_49	sample_id
0	99	-0.0664	-0.0999	-0.0949	....	-0.0942	2
0	101	-0.0603	-0.0938	-0.0900	....	-0.0935	2
1	114	0.0000	0.0001	0.0001	....	0.0001	2
1	115	0.0001	0.0001	0.0001	....	0.0001	2
....	....	....	....	....	....	....	....

### Example: TD\_GLM Using Credit Data Set

The following credit data set is used in this example:

ID	A1	A2	A7	A10	A13	A14	A0_b	A0_a	A3_Y	A3_u	A4_p	A4_g
61	0. 218228	2.17724	0.142986	1. 25309	0.238997	-0. 130455	1	0	0	1	0	1
297	-0. 453325	-0. 0247979	-0. 404925	-0. 77231	0.886307	-0. 225685	0	1	0	1	0	1
631	-1. 13175	-0. 77234	-0. 199458	-0. 547266	-0. 761392-0. 643699	-0. 225817	0	1	0	1	0	1
122	-0. 657541	1.20223	0. 00600835	0. 802998	-0.54366	-0. 120063	1	0	0	1	0	1
...	...	...	...	...	...	...	...	...	...	...	..	...

### TD\_GLM Call for Credit Data

```
CREATE VOLATILE TABLE td_glm_output_credit_ex AS (
  SELECT * FROM td_glm (
    ON credit_ex_merged AS InputTable
    USING
      InputColumns('a1', 'a2', 'a7', 'a10', 'a13', 'a14', 'a0_b', 'a0_a',
```

```

'a3_y', 'a3_u', 'a4_p', 'a4_g', 'a5_k', 'a5_cc', 'a5_d', 'a5_c', 'a5_aa',
'a5_m', 'a5_q', 'a5_w', 'a5_e', 'a5_ff', 'a5_j', 'a5_x', 'a5_i', 'a6_v',
'a6_h', 'a6_bb', 'a6_z', 'a6_ff', 'a6_j', 'a8_t', 'a8_f', 'a9_t', 'a9_f',
'a11_t', 'a11_f', 'a12_g', 'a12_s')
    ResponseColumn('Outcome')
    Family('Binomial')
    BatchSize(10)
    MaxIterNum(300)
    RegularizationLambda(0.02)
    Alpha(0.15)
    IterNumNoChange(50)
    Tolerance(0.001)
    Intercept('true')
    LearningRate('optimal')
    InitialEta(0.001)
    Momentum(0.0)
    LocalSGDIterations(0)
) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS
;

```

### TD\_GLM Output for Credit Data

Attribute	Predictor	Estimate	Value
-13	LocaleSGD Iterations	0	
-12	Nesterov		FALSE
-11	Momentum	0	
-10	Learning Rate (Final)	0.287682	
-9	Learning Rate (Initial)	0.001	
-8	Number of Iterations	156	CONVERGED
-7	Alpha	0.15	Elasticnet
-6	Regularization	0.02	ENABLED
-5	BIC	151.787	
-4	AIC	80.4189	
-3	Number of Observations	44	
-2	Loglik	-0.209461	
-1	Loss Function		LOG

Attribute	Predictor	Estimate	Value
0	(Intercept)	0.146566	
1	A1	0.732289	
2	A2	0	
3	A7	0.717899	
4	A10	0.682358	
5	A13	0.822302	
6	A14	0.176791	
7	A0_b	0.172178	
8	A0_a	-0.12165	
9	A3_y	-0.285135	
10	A3_u	0.335663	
11	A4_p	-0.285135	
12	A4_g	0.335663	
13	A5_k	0.0358046	
14	A5_cc	0	
15	A5_d	0.0480538	
16	A5_c	0.430725	
17	A5_aa	0	
18	A5_m	-0.332389	
19	A5_q	0.524153	
20	A5_w	-0.599829	
21	A5_e	0.0257252	
22	A5_ff	-0.359011	
23	A5_j	-0.119432	
24	A5_x	0.0494795	
25	A5_i	0	
26	A6_v	-0.387493	
27	A6_h	0.0415697	
28	A6_bb	0.0604108	

Attribute	Predictor	Estimate	Value
29	A6_z	0	
30	A6_ff	-0.372217	
31	A6_j	0.538139	
32	A8_t	0.9259	
33	A8-f	-0.875372	
34	A9_t	0	
35	A9_f	0	
36	A11_t	0.197957	
37	A11_f	-0.146928	
38	A12_g	-0.221414	
39	A12_s	0.272506	

## Example: TD\_GLM Using Housing Data

This example takes raw housing data, and does the following:

1. Uses [TD\\_ScaleFit](#) to standardize the data.
2. Uses [TD\\_ScaleTransform](#) to transform the data.
3. Uses TD\_GLM to get a model.

## Raw Housing Data

ID	MedInc	HousAge	AveRoom	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
2833	1.3527	30	2.24754	0.742574	169	1.67327	35.39	-119.02	0.6
5328	2.7679	23	3.03868	1.06446	2031	1.63658	34.04	-118.45	2.775
5300	1.583	19	3.14815	1.04548	3751	2.4373	34.07	-118.45	3.5
12433	1.7344	24	3.2984	1.05856	4042	4.4963	33.51	-118.01	0.664
...	...	...	...	...	...	...	...	...	...

## TD\_ScaleFit Call for Housing Data

```
SELECT * FROM TD_ScaleFit(
    ON cal_housing_ex_raw AS InputTable
    OUT VOLATILE TABLE OutputTable(scaleFitOut_cal_ex)
    USING
        TargetColumns('medinc', 'houseage', 'averooms', 'avebedrms', 'population',
        'aveoccup', 'latitude', 'longitude')
        ScaleMethod('STD')
) AS dt2;
```

## TD\_ScaleFit Output for Housing Data

TD_STATTYPE_SQLPIT	Medinc	HousAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
min	15472	7	2.24752	0.742574	47	1.63658	32.64	-123.35

TD_STATTYPE_SCPLIT		Medinc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
max		10.7721	52	8.89005	2.56522	4145	5.46536	40.99	-116.01
sum		201.401	2134	349.412	74.0563	91566	200.562	2459.39	-8256.05
count		69	69	69	69	69	69	69	69
null		0	0	0	0	0	0	0	0
avg		3.78842	30.9275	1.07328	1.07328	1327.04	2.90714	-119.653	
variance		6.40242	142.803	1.50533	0.0458083	86163	0.618171	0.618171	4.73921
std		2.30741	11.8631	1.23694	0.213472	921.491	0.780521	0.780521	2.16114
var		2.32431	11.95	1.246	0.214029	929.242	0.786239	0.786239	2.17667
stddev		1	1	1	1	1	1	1	1
multiplier		0	0	0	0	0	0	0	0
intercept		3.78842	30.9275	1.06394	1.07328	1327.04	2.90714	-119.653	
location		2.30741	11.8631	1.23694	0.213472	921.491	0.780521	0.780521	2.16114
scale		nan	nan	nan	nan	nan	nan	nan	nan
globalScale, false		3	3	3	3	3	3	3	3
ScaleMethodNumberMapping: [0 mean, 1 sum, 2 std, 3 std, 4 range, 5 mrange, 6 maxabs, 7 rescale]		nan	nan	nan	nan	nan	nan	nan	nan
missvalue, keep		nan	nan	nan	nan	nan	nan	nan	nan

## TD\_ScaleTransform Call for Housing Data

```
CREATE MULTISET TABLE cal_housing_ex_scaled AS (
    SELECT * FROM TD_ScaleTransform(
        ON cal_housing_ex_raw AS InputTable
        ON scaleFitOut_cal_ex AS FitTable DIMENSION
        USING
        accumulate('id', 'MedHouseVal')
    ) AS dt1
) WITH data;
```

## TD\_ScaleTransform Output for Housing Data

ID	MedHouseVal	Medinc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
244	4.117	0.405796	1.10294	0.460347	0.456686	1.00221	1.04105	0.046201	-1.18745
670	1.923	-0.0308521	0.427982	0.199999	0.539672	0.761956	0.21356	0.006345	-1.18008
686	1.576	-0.152084	-0.5781865	0.655426	0.513581	0.685902	0.533101	1.01705	-1.14157
1754	1.651	-0.0263147	0.596172	0.454207	0.0272726	0.0893203	0.0827654	1.01705	-1.23412
--	--	--	--	--	--	--	--	--	--

## TD\_GLM Call for Housing Data

```
CREATE VOLATILE TABLE td_glm_cal_ex AS (
    SELECT * from TD_GLM (
        ON cal_housing_ex_scaled AS InputTable
        USING
        InputColumns('medinc', 'houseage', 'averooms', 'avebedrms',
        'population', 'aveoccup', 'latitude', 'longitude')
        ResponseColumn('MedHouseVal')
        Family('Gaussian')
        BatchSize(10)
        MaxIterNum(300)
        RegularizationLambda(0.02)
        Alpha(0.15)
        IterNumNoChange(50)
        Intercept('true')
        LearningRate('invtime')
        InitialEta(0.05)
        Momentum(0)
        Nesterov('false')
        LocalSGDIterations(0)
    )
)
```

```

) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;

```

### **TD\_GLM Output for Housing Data**

Attribute	Predictor	Estimate	Value
-13	LocalSGD Iterations	0	
-12	Nesterov		
-11	Momentum	0	
-10	Learning Rate (Final)	0.0133974	
-9	Learning Rate (Initial)	0.05	
-8	Number of Iterations	194	CONVERGED
-7	Alpha	0.15	Elasticnet
-6	Regularization	0.02	ENABLE
-5	BIC	-67.6236	
-4	AIC	-87.7305	
-3	Number of Observations	69	
-2	MSE	0.216033	
-1	Loss Function		SQUARED_ERROR
0	(Intercept)	2.07174	
1	MedInc	0.782883	
2	HouseAge	0.231914	
3	AveRooms	0.0619822	
4	AveBedrms	-0.113656	
5	Population	0.211336	
6	AveOccup	-0.388201	
7	Latitude	-0.195511	
8	Longitude	-0.193884	

### Example: TD\_GLM Call of Housing Data Using Forward Selection

```
SELECT * FROM TD_GLM (
ON cal_housing_ex_raw AS InputTable
USING
    InputColumns('[1:4]', '[7:10]')
    ResponseColumn('price')
    Family('Gaussian')
    LearningRate('optimal')
    RegularizationLambda(0.02)
    Alpha(0.33)
    BatchSize(10)
    MaxIterNum (36)
    IterNumNoChange(100)
    Tolerance(0.0001)
    InitialEta(0.02)
    StepwiseDirection('forward')
    MaxStepsNum(10)
) as dt
```

### TD\_GLM Output of Housing Data Using Forward Selection

Attribute	Predictor	Estimate	Value
1	RAD	-3.778	
0	(Intercept)	23.16449	
-1	Loss Function	SQUARED_ERROR	
-2	MSE	47.409	
-3	Number of Observations	360	
-4	AIC	1393.174	
-5	BIC	1400.946	
-6	Regularization	0.02	ENABLED
-7	Alpha	0.33	Elasticnet
-8	Number of Iterations	36	NOT CONVERGED
-9	Learning Rate (Initial)	0.02	
-10	Learning Rate (Final)	0.929316	
-11	Momentum	0	
-12	Nesterov		FALSE

Attribute	Predictor	Estimate	Value
-13	LocalSGD Iterations	0	

### Example: TD\_GLM Call of Housing Data Using Backward Deletion

```

SELECT * FROM TD_GLM (
ON cal_housing_ex_raw AS InputTable
OUT TABLE MetaInformationTable(logtable)
USING
    InputColumns('[3:6]')
    ResponseColumn('price')
    Family('Gaussian')
    LearningRate('optimal')
    RegularizationLambda(0.02)
    Alpha(0.33)
    BatchSize(10)
    MaxIterNum (36)
    IterNumNoChange(100)
    Tolerance(0.0001)
    InitialEta(0.02)
    StepwiseDirection('backward')
    MaxStepsNum(10)
) as dt;

```

### TD\_GLM Output of Housing Data Using Backward Deletion

Attribute	Predictor	Estimate	Value
0	(Intercept)	23.14352	
1	NOX	-1.85727	
2	RM	8.010339	
-1	Loss Function	SQUARED_ERROR	
-2	MSE	11.67791	
-3	Number of Observations	360	
-4	AIC	890.7715	
-5	BIC	902.4298	
-6	Regularization	0.02	ENABLED
-7	Alpha	0.33	Elasticnet

Attribute	Predictor	Estimate	Value
-8	Number of Iterations	36	NOT CONVERGED
-9	Learning Rate (Initial)	0.02	
-10	Learning Rate (Final)	0.929316	
-11	Momentum	0	
-12	Nesterov		FALSE
-13	LocalSGD Iterations	0	

### Example: Regression Using One Table

The following examples use this input table. Table name is housing\_train\_segment.

sn	price	lotsize	bedrooms	bathrms	stories	driveway	recroom	fullbase	gashw	airco	ga
1	42000	5850	3	1	2	1	0	1	0	0	1
2	38500	4000	2	1	1	1	0	0	0	0	0
3	49500	3060	3	1	1	1	0	0	0	0	0
4	60500	6650	3	1	2	1	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...

### TD\_GLM Call with Family: Gaussian for Input Table

```
SELECT * FROM TD_GLM (
    ON housing_train_segment AS InputTable PARTITION BY partition_id ORDER BY sn
    USING
        Family('Gaussian')
        InputColumns('[3:10]')
        ResponseColumn('price')
        BatchSize(10)
        MaxIterNum (1000)
        IsDebug('true')
) AS dt;
```

### TD\_GLM Output with Family: Gaussian for Input Table

partition_id	attribute	predictor	estimate	value
31	0	(intercept)	4777.491933	-
31	1	bedrooms	11243.33906	-

partition_id	attribute	predictor	estimate	value
31	2	bathrms	7426.162028	-
31	3	stories	6897.217893	-
...	...	...	...	...

### Example: Using an Attribute Table

Attribute Table

partition_id	Attribute Column
5	bedrooms1
31	gashw
37	bathrms1
37	recroom1

### TD\_GLM Call with Family: Gaussian for Input Table and Attribute Table

```
SELECT * FROM TD_GLM (
    ON housing_train_segment AS InputTable PARTITION BY partition_id
    ORDER BY sn
    ON housing_train_attribute AS AttributeTable PARTITION BY partition_id
    USING
        Family('Gaussian')
        InputColumns('[3:10]')
        ResponseColumn('price')
        MaxIterNum (100)
) AS dt;
```

### TD\_GLM Output for Attribute Table

partition_id	Attribute	Predictor	Estimate	Value
31	0	(intercept)	48060.8685	-
31	1	gashw	2532.68146	-
31	-1	Loss Function	-	SQUARED
31	-3	Number of Observations	115	-
31	-2	MSE	39290745.1	-
31	-4	AIC	2014.94745	-

partition_id	Attribute	Predictor	Estimate	Value
31	-5	BIC	2020.43731	-
31	-6	Regularization	0.02	ENABLED
31	-7	Alpha	0.15	Elasticnet
31	-8	Number of Iterations	100	NOT CONVERGED
31	--9	Learning Rate (initial)	0.05	-
31	-10	Learning Rate (final)	0.01581139	-
31	-11	Momentum	0	-
31	-12	Nesterov	-	FALSE

### Example: Using a Parameter Table

Parameter Table

Partition_id	Parameter_column	Value_column
31	Alpha1	0.5

### TD\_GLM Call with Family: Gaussian for Input Table and Parameter Table

```
SELECT * FROM TD_GLM (
    ON housing_train_segment AS InputTable PARTITION BY partition_id
    ORDER BY sn
    ON housing_train_parameter AS ParameterTable PARTITION BY partition_id
    USING
        Family('Gaussian')
        InputColumns('[3:10]')
        ResponseColumn('price')
        MaxIterNum (100)
) AS dt;
```

### TD\_GLM Output for Parameter Table

Partition_id	Attribute	Predictor	Estimate	Value
31	-	-	-	Invalid parameter in parameter table. Found: Alpha1 in partition(31)
32	0	(intercept)	4830.20162	-
32	1	bedrooms	8167.57498	-
...	...	...	...	...

## Example: Classification Using TD\_GLM Call with Family: Binomial for Input Table and Attribute Table

```
SELECT * FROM TD_GLM (
    ON housing_train_segment AS InputTable PARTITION BY partition_id
    ON housing_train_attribute AS AttributeTable PARTITION BY partition_id
    USING
        Family('Binomial')
        InputColumns('[3:10]')
        ResponseColumn('homestyle')
        MaxIterNum (100)
) AS dt;
```

## TD\_GLM Output with Family: Binomial for Input Table and Attribute Table

partition_id	attribute	predictor	estimate	value
31	0	(intercept)	-0.308286246	-
31	1	gashw	0.07012385	-
31	-1	Loss Function	-	LOG
31	-3	Number of Observations	134	-
...	...	...	...	...

## TD\_KMeans

The k-means algorithm groups a set of observations into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid). This algorithm minimizes the objective function, that is, the total Euclidean distance of all data points from the center of the cluster as follows:

1. Specify or randomly select k initial cluster centroids.
2. Assign each data point to the cluster that has the closest centroid.
3. Recalculate the positions of the k centroids.
4. Repeat steps 2 and 3 until the centroids no longer move.

The algorithm does not necessarily find the optimal configuration, as it depends significantly on the initial randomly selected cluster centers. You can run TD\_KMeans multiple times to reduce the effect of this limitation.

You can also select initial centroids using the 'KMeans++' algorithm to overcome this limitation. The 'KMeans++' algorithm is a smarter way of choosing initial centroids for the KMeans clustering algorithm. The main idea is to select the initial centroids far away from each other. It reduces the possibility of initial centroids being chosen from the same cluster. 'KMeans++' improves the overall quality of clustering, and in some cases, can also speed up the convergence of the KMeans algorithm.

## InputTable Usage Considerations

TD\_KMeans also returns the within-cluster-squared-sum, which you can use to determine an optimal number of clusters using the Elbow method.

- This function does not consider the InputTable and InitialCentroidsTable Input rows that have a NULL entry in the specified TargetColumns.
- The function can produce deterministic output across different machine configurations if you provide the InitialCentroidsTable in the query.
- The function randomly samples the initial centroids from the InputTable, if you do not provide the InitialCentroidsTable in the query. In this case, you can use the Seed element to make the function output deterministic on a machine with an assigned configuration. However, using the Seed argument does not guarantee deterministic output across machines with different configurations.

### Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).  
For information about PTCs, see *International Character Set Support*, B035-1125.
- This function does not support KanjiSJIS or Graphic data types.
- SELECT TOP gives non-deterministic results. Therefore, identical queries including this instruction may produce different results.

## Function Information

- [TD\\_KMeans Syntax](#)
- [Required Syntax Elements for TD\\_KMeans](#)
- [Optional Syntax Elements for TD\\_KMeans](#)
- [TD\\_KMeans Input](#)
- [TD\\_KMeans Output](#)
- [TD\\_KMeans Usage Notes](#)
- [Examples: How to Use TD\\_KMeans](#)

## TD\_KMeans Syntax

```
TD_KMeans (
  ON {table | view | (query)} AS InputTable
  [ ON {table | view | (query)} AS InitialCentroidsTable DIMENSION ]
  [ OUT [ PERMANENT | VOLATILE ] TABLE ModelTable(model_output_table_name) ]
  USING
  IdColumn('id_column')
  TargetColumns({'target_column' | 'target_column_range'}[,...])
```

```
[ InitialCentroidsMethod({'random' | 'kmeans++'}) ]
[ NumClusters(number_of_clusters) ]
[ Seed(seed_value) ]
[ StopThreshold(threshold_value) ]
[ MaxIterNum(number_of_iterations) ]
[ NumInit(num_init) ]
[ OutputClusterAssignment({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_KMeans

### ON clause

Accepts the InputTable clause.

The input table can have no partition at all, or have the following combinations of PARTITION BY ANY/ORDER BY clauses:

- PARTITION BY ANY ORDER BY
- PARTITION BY ANY

If you specify PARTITION BY column for InputTable, an error is reported.

NumClusters argument and InitialCentroidsTable cannot be provided together, otherwise, an error is reported.

### **IdColumn**

Specifies the input table column name that has the unique identifier for each input table row.

### **TargetColumns**

Specifies the input table columns for clustering.

## Optional Syntax Elements for TD\_KMeans

The following are the hyperparameters that you tune when using the TD\_KMeans function.

**ON clause**

Accepts the InitialCentroidsTable clause.

**OUT clause**

Accepts the ModelTable clause.

**ModelTable**

Specifies the ModelTable name to save the clustering data model. If specified, then a model containing centroids of clusters is saved in the specified ModelTable name.

**InitialCentroidsMethod**

Specifies the initialization method to be used for selecting initial set of centroids. Not required, if the InitialCentroidsTable is specified. Allowed values: [random, kmeans++].

'random': The initial set of centroids are selected randomly.

'kmeans++': The initial set of centroids are selected using the KMeans++ algorithm.

Default: 'random'

**NumClusters**

Specifies the number of clusters to create from the clustering data. Not required, if the InitialCentroidsTable is specified.

**Seed**

Specifies a non-negative integer value to randomly select the initial cluster centroid positions from the input table rows. Not required, if the InitialCentroidsTable is specified.

**StopThreshold**

The algorithm converges if the distance between the centroids from the previous iteration and the current iteration is less than the specified value.

Default: 0.0395

**MaxIterNum**

Specifies the maximum number of iterations for the K-means algorithm. The algorithm stops after performing the specified number of iterations even if the convergence criterion is not met.

Default: 10

**NumInit**

Specifies the number of times to repeat clustering with different initial centroid seeds. The function returns the model having the least value of Total Within Cluster Squared Sum.

Not required, if the InitialCentroidsTable is specified.

Default: 1

### **OutputClusterAssignment**

Specifies whether to return the Cluster Assignment information.

Default: False

## **TD\_KMeans Input**

### **InputTable Schema**

Column	Data Type	Description
IdColumn	Any	The InputTable column name that has the unique identifier for each input table row.
TargetColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, FLOAT, REAL, DOUBLE PRECISION	The InputTable columns for clustering.

### **InitialCentroids Table Schema**

Column	Data Type	Description
Initial_Clusterid_Column	BYTEINT, SMALLINT, INTEGER, BIGINT	The column that contains the unique identifiers of initial centroids.
TargetColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, FLOAT, REAL, DOUBLE PRECISION	The columns that contain the initial centroid values.

## **TD\_KMeans Output**

### **Output Table Schema**

If the OutputClusterAssignment value is set to False:

Column	Data Type	Description
TD_CLUSTERID_KMEANS	BIGINT	The unique identifier of the cluster.
TargetColumns	REAL	The columns that contain the centroid value for each feature.

Column	Data Type	Description
TD_SIZE_KMEANS	BIGINT	The number of points in the cluster.
TD_WITHINSS_KMEANS	REAL	The within-cluster-sum-of-squares, that is, the sum of squared differences of each point from its cluster centroid.
Id_Column	BYTEINT	The unique identifier column name copied from the InputTable. This column contains only NULL values in the output.
TD_MODELINFO_KMEANS	VARCHAR(128) CHARACTER SET LATIN	The following information related to the model is saved: <ul style="list-style-type: none"> <li>• Converged: True or False</li> <li>• Number of Iterations: The number of iterations performed by the function.</li> <li>• Number of Clusters: The number of clusters produced.</li> <li>• Total_WorldSS: The total within cluster sum of squares.</li> <li>• Between_SS: Between sum of squares, that is, the sum of squared distances of centroids to global mean, where squared distance of each mean to global mean is multiplied by the number of data points it represents.</li> <li>• Method for InitialCentroids: 'Random' or 'KMeans++' or 'Externally supplied InitialCentroidsTable'.</li> </ul>

If the OutputClusterAssignment value is set to True:

Column	Data Type	Description
Id_Column	Any	The unique identifier of input rows copied from the input table.
TD_CLUSTERID_KMEANS	BIGINT	The ClusterId assigned to the input row.

## TD\_KMeans Usage Notes

K-means is an algorithm that does unsupervised clustering. This method groups a set of n observations into k clusters based on their proximity to the cluster centers. The objective of the algorithm is to minimize the within-cluster variance, meaning that similar observations belong to the same cluster. The goal is to assign all n points to their respective clusters.

The k-means algorithm calculates the distance between a point and each cluster center, the algorithm assigns the point to the nearest cluster. The k-means algorithm assumes that data points that are close together are similar.

The number of clusters or k is a crucial hyperparameter. If this value is not known, the algorithm uses a method to determine the optimal value of k.

Applications such as Market Segmentation, Document Clustering, Image Segmentation, and Image Compression uses k-means algorithm. Although the algorithm is simple and achieves good performance,

the algorithm is sensitive to outliers that can affect the cluster centers. K-means can also become slow when processing larger datasets because the algorithm needs to compare data points.

## Why Use K-Means

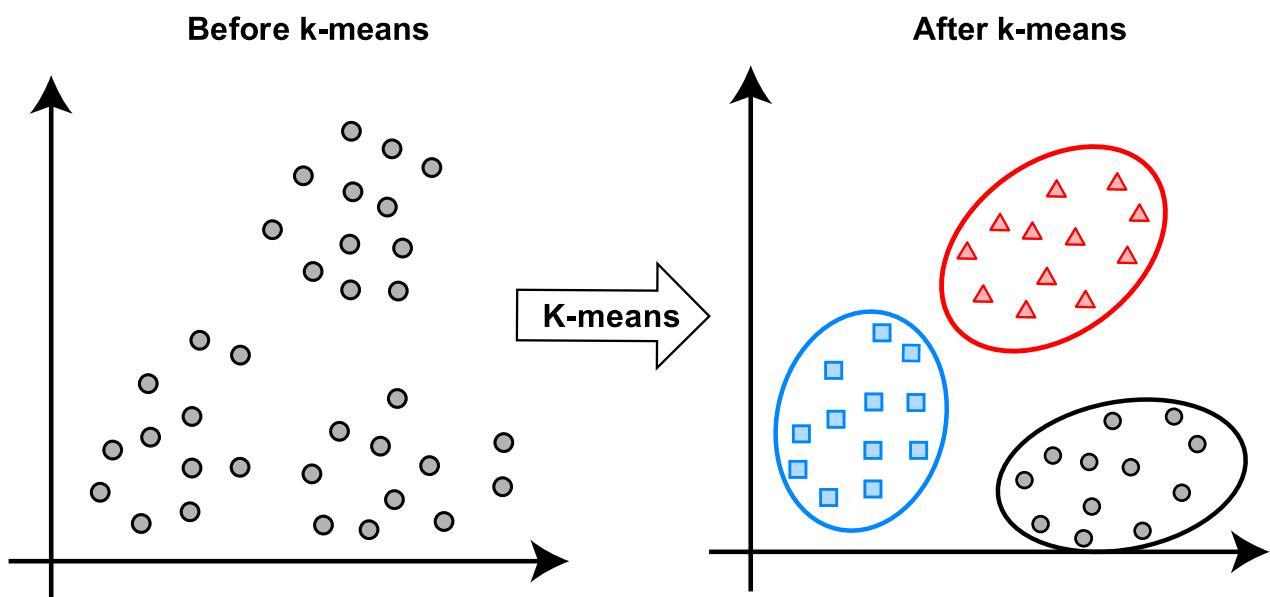
K-means clustering is an unsupervised learning algorithm that separates an unlabeled dataset into different clusters. The value  $k$  determines the number of pre-defined clusters to create, with  $k=2$  resulting in two clusters,  $k=3$  resulting in three clusters, and so on.

This algorithm enables you to group data into different categories and identify these groups in an unlabeled dataset without any training. K-means is a centroid-based algorithm where each cluster is associated with a centroid. The primary goal of the algorithm is to minimize the total distance between data points and their respective clusters.

To implement the algorithm, the input is an unlabeled dataset that is divided into  $k$  clusters, and the process repeats until the best clusters are found. You need to predetermine the value of  $k$ .

The k-means algorithm performs two main tasks: iteratively determining the best value for  $k$  center points or centroids and assigning each data point to its nearest centroid to create a cluster. Each cluster contains data points with similarities that distinguish the data points from other clusters.

The following diagram illustrates how k-means algorithm works.



K-means is a simple, fast, and versatile algorithm that has been applied to a wide range of problems. However, its performance can be sensitive to the:

- Choice of seed or the order of data points.
- Outliers.
- Scale of different variables.
- Large datasets (Computation Cost).

K-means upsides:

- Simple to implement.
- Faster than other clustering algorithms (like hierarchical clustering).
- Guarantees convergence.
- Generalizes to clusters of different shapes and sizes, such as elliptical clusters.

In conclusion, k-means is a simple, yet powerful algorithm that has different applications in machine learning and computer science. These include Anomaly Detection, Image Segmentation, and Recommendation Engines to name a few. Although you need to provide a value of k (the number of clusters), you can find the number of clusters using other methods like Elbow or Silhouette method.

## Examples: How to Use TD\_KMeans

### **InputTable**

```
create table kmeans_input_table (id int, c1 int, c2 int);
insert into kmeans_input_table values(1,1,1);
insert into kmeans_input_table values(2,2,2);
insert into kmeans_input_table values(3,8,8);
insert into kmeans_input_table values(4,9,9);
```

id	C1	C2
1	1	1
2	2	2
3	8	8
4	9	9

### **InitialCentroidsTable**

```
create table kmeans_initial_centroids_table as (select * from
kmeans_input_table where id in (2,4)) with data;
```

id	C1	C2
2	2	2
4	9	9

```
2 2 2
4 9 9
```

### **Example 1: TD\_KMeans SQL Call: InitialCentroidsTable is Not Provided**

```
SELECT * FROM TD_KMeans (
ON kmeans_input_table AS InputTable
USING
IdColumn('id')
TargetColumns('c1','c2')
NumClusters(2)
Seed(0)
StopThreshold(0.0395)
MaxIterNum(3)
)AS dt;
```

### **Output: InitialCentroidsTable is Not Provided**

td_clusterid_kmeans	C1	C2	td_size_kmeans	td_withinss_kmeans	id	td_modelinfo_kmeans
0	1.5	1.5	2	1	NULL	NULL
1	8.5	8.5	2	1	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	Converged : True
NULL	NULL	NULL	NULL	NULL	NULL	Number of Iterations : 2
NULL	NULL	NULL	NULL	NULL	NULL	Number of Clusters : 2
NULL	NULL	NULL	NULL	NULL	NULL	Total_WithinSS 2.0000000000000E+00
NULL	NULL	NULL	NULL	NULL	NULL	Between_SS : 9.8000000000000E+01
NULL	NULL	NULL	NULL	NULL	NULL	Method For InitialCentroids : Random

### **Example 2: TD\_KMeans SQL Call: InitialCentroidsTable is Provided**

```
SELECT * FROM TD_KMeans (
ON kmeans_input_table AS InputTable
ON kmeans_initial_centroids_table AS InitialCentroidsTable DIMENSION
USING
IdColumn('id')
TargetColumns('c1','c2')
StopThreshold(0.0395)
MaxIterNum(3)
) AS dt;
```

### **Output: InitialCentroidsTable is Provided**

td_clusterid_kmeans	C1	C2	td_size_kmeans	td_withinss_kmeans	id	td_modelinfo_kmeans
0	1.5	1.5	2	1	NULL	NULL
1	8.5	8.5	2	1	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	Converged : True
NULL	NULL	NULL	NULL	NULL	NULL	Number of Iterations : 2
NULL	NULL	NULL	NULL	NULL	NULL	Number of Clusters : 2
NULL	NULL	NULL	NULL	NULL	NULL	Total_WithinSS : 2.0000000000000E+00
NULL	NULL	NULL	NULL	NULL	NULL	Between_SS : 9.8000000000000E+01
NULL	NULL	NULL	NULL	NULL	NULL	Method For InitialCentroids : Externally supplied InitialCentroidsTable

**Example 3: TD\_KMeans SQL Call: TD\_KMeans Call with OutputClusterAssignment Set to true**

```
SELECT * FROM TD_KMeans (
ON kmeans_input_table AS InputTable
ON kmeans_initial_centroids_table AS InitialCentroidsTable DIMENSION
USING
IdColumn('id')
TargetColumns('c1','c2')
StopThreshold(0.0395)
MaxIterNum(3)
OutputClusterAssignment('true')
)AS dt;
```

**Output: TD\_KMeans Call with OutputClusterAssignment Set to true**

id	td_clusterid_kmeans
1	0
2	0
3	1
4	1

**Example 4: TD\_KMeans SQL Call: with OutputClusterAssignment Set to true and ModelTable [Out Table Clause] is Provided**

```
SELECT * FROM TD_KMeans (
ON kmeans_input_table AS InputTable
ON kmeans_initial_centroids_table AS InitialCentroidsTable DIMENSION
OUT TABLE ModelTable(kmeans_model)
USING
IdColumn('id')
TargetColumns('c1','c2')
StopThreshold(0.0395)
MaxIterNum(3)
OutputClusterAssignment('true')
)AS dt;
```

**Output: TD\_KMeans Call with OutputClusterAssignment Set to true and ModelTable [Out Table Clause] is Provided**

id	td_clusterid_kmeans
1	0

2	0
3	1
4	1

# KMeans ModelTable Output

```

td_clusterid_kmeans c1 c2 td_size_kmeans td_withinss_kmeans id td_modelinfo_kmeans
-----+-----+-----+-----+-----+-----+-----+-----+
Null Null Null Null Null Null Converged : True
Null Null Null Null Null Null Number of Clusters : 2
Null Null Null Null Null Null Total_WithinSS : 2.00000000000000E+00
Null Null Null Null Null Null Between_SS : 9.80000000000000E+01
Null Null Null Null Null Null Number of Iterations : 2
0 1.500000000 1.500000000 2 1.000000000 Null Null
1 8.500000000 8.500000000 2 1.000000000 Null Null
Null Null Null Null Null Null Method for InitialCentroids : Externally
supplied InitialCentroidsTable

```

**Input [InitialCentroidsMethod : 'kmeans++']**

```
create table kmeans_input_table_2 (id INTEGER, c1 FLOAT, c2 FLOAT);
insert into kmeans_input_table_2 values(1,18,18);
insert into kmeans_input_table_2 values(2,19,19);
insert into kmeans_input_table_2 values(3,20,20);
insert into kmeans_input_table_2 values(4,55,55);
insert into kmeans_input_table_2 values(5,56,56);
insert into kmeans_input_table_2 values(6,57,57);
insert into kmeans_input_table_2 values(7,88,88);
insert into kmeans_input_table_2 values(8,89,89);
insert into kmeans_input_table_2 values(9,90,90);
```

## **kmeans\_input\_table\_2**

<b>id</b>	<b>c1</b>	<b>c2</b>
1	18	18
2	19	19
3	20	20
4	55	55
5	56	56
6	57	57
7	88	88
8	89	89
9	90	90

**Example 5: TD\_KMeans SQL Call [InitialCentroidsMethod : 'kmeans++']**

```
SELECT * FROM TD_KMeans (
ON kmeans_input_table_2 AS InputTable
USING
```

```

IdColumn('id')
TargetColumns('c1','c2')
InitialCentroidsMethod('kmeans++')
NumClusters(3)
Seed(10)
StopThreshold(0.0395)
MaxIterNum(3)
) AS dt;

```

### **Output [InitialCentroidsMethod : 'kmeans++']**

td_clusterid_kmeans	c1	c2	td_size_kmeans	td_withinss_kmeans	id	td_modelinfo_kmeans
0	89	89	3	4	NULL	NULL
1	19	19	3	4	NULL	NULL
2	56	56	3	4	NULL	NULL
NULL	NULL	NULL	NULL	NULL	Converged : True	
NULL	NULL	NULL	NULL	NULL	Number of Iterations : 2	
NULL	NULL	NULL	NULL	NULL	Number of Clusters : 3	
NULL	NULL	NULL	NULL	NULL	Total_WithinSS :1.200000000000E+01	
NULL	NULL	NULL	NULL	NULL	Between_SS :1.471600000000E+04	
NULL	NULL	NULL	NULL	NULL	Method For InitialCentroids: KMeans++	

## **TD\_KNN**

K-nearest Neighbors (k-NN) is a supervised learning technique that predicts the test data by computing nearest neighbors from training data based on a similarity (distance) metric. The algorithm does not construct a model from the training set, instead, it predicts the test data directly based on similarity with training data.

KNN uses a distance metric, such as Euclidean or Manhattan distance, to determine the similarity between data points. During the prediction phase, the algorithm calculates the distance between the new data point and all training examples, selects the K closest neighbors, and makes a prediction based on the majority class or average value of these neighbors. KNN is simple and easy to implement, but it can be computationally expensive and sensitive to irrelevant features.

The choice of K and the distance metric are important factors that affect the performance of KNN.

- If K is too small, the algorithm may be too sensitive to outliers.
- If K is too large, the algorithm may not be able to capture the underlying patterns in the data.

TD\_KNN supports classification, regression, and neighbors model types. For classification, the category of the test data is based on a majority vote among the k nearest neighbors. For regression, the test data is assigned a score based on the mean similarity from its neighbors. For neighbors, the function returns the nearest neighbors.

The function supports up to 2018 features and 1000 labels for the classification model type.

The function internally calls the TD\_VectorDistance function, where the computation complexity is  $O(N^2)$ , where N is the number of rows. Therefore, the query may run significantly longer as the number of rows increases in either the training table or the test table. In such cases, alternative algorithms such as decision trees, random forests, or neural networks may be more appropriate.

**Note:**

Because the training table for TD\_KNN is a DIMENSION input, it is copied to the spool for each AMP before processing. Due to this reason, the size/scalability of this input is limited by the user spool in the database.

**Function Information**

- [TD\\_KNN Usage Notes](#)
- [TD\\_KNN Syntax](#)
- [Required Syntax Elements for TD\\_KNN](#)
- [Optional Syntax Elements for TD\\_KNN](#)
- [TD\\_KNN Input](#)
- [TD\\_KNN Output](#)
- [Examples: How to Use TD\\_KNN](#)

**TD\_KNN Usage Notes**

K-Nearest Neighbors (KNN) is a supervised learning algorithm that is commonly used for classification and regression problems. It's a non-parametric, instance-based, and lazy learning algorithm that operates under the principle of similarity-based classification, where a sample is classified based on the class labels of its nearest neighbors in the feature space.

Given a set of training samples  $X = \{x_1, x_2, \dots, x_n\}$ , where each  $x_i$  is a d-dimensional feature vector and its corresponding class label  $y_i$ , the KNN algorithm works as follows:

For a new sample  $x'$ , the algorithm computes the distance between  $x'$  and each training sample  $x_i$  in the feature space. A common choice of distance metric is the Euclidean distance:

$$d(x_i, x') = \sqrt{\sum (x_{ij} - x'j)^2}$$

where  $x_{ij}$  is the j-th feature of  $x_i$  and  $x'j$  is the j-th feature of  $x'$

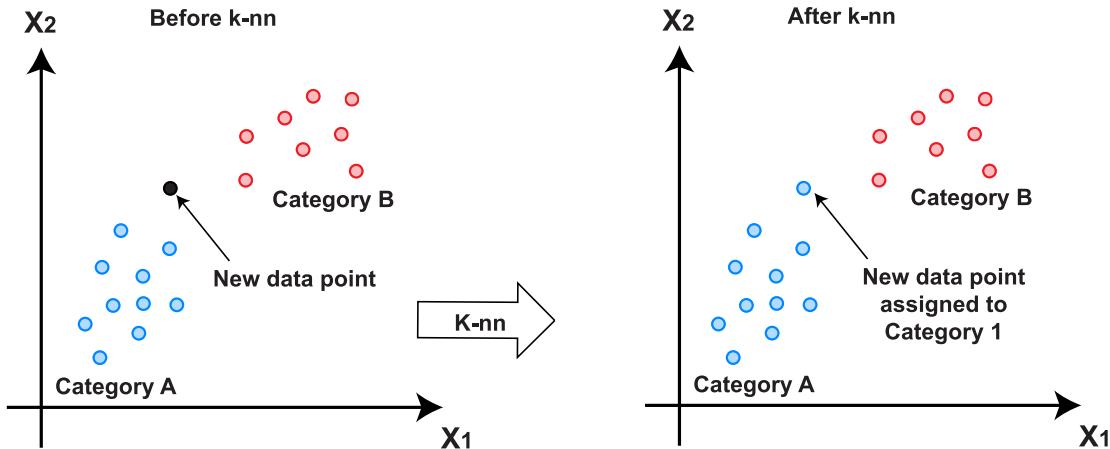
The "k" nearest neighbors to  $x'$  are then selected, where "k" is a user-defined constant integer number.

The class label of  $x'$  is determined by the majority vote of the class labels of its k nearest neighbors.

Mathematically, the predicted class label  $\hat{y}$  of  $x'$  can be expressed as:

$$\hat{y} = \operatorname{argmax}(c_j) (1/k) \sum [y_i = c_j]$$

where  $c_j$  is a class label,  $y_i$  is the class label of the i-th nearest neighbor, and the sum is over the k nearest neighbors.



KNN is a simple, fast, and versatile algorithm that has been applied to a wide range of problems. However, its performance can be sensitive to:

- Choice of distance metric
- The number of nearest neighbors
- The presence of irrelevant or noisy features
- Large datasets (Computation Cost)

KNN has the following benefits:

- Is easy to implement
- Has no training phase
- Can be used for classification, regression, and anomaly detection problems

In conclusion, KNN is a simple and powerful algorithm that has been widely used for various tasks in machine learning and computer science. Its strengths include its ability to handle non-linear relationships, multi-class problems, and its ease of implementation. However, its performance can be sensitive to the choice of "k", the distance metric, and the presence of irrelevant or noisy features, and it's important to carefully evaluate its performance for each specific problem.

## **TD\_KNN Syntax**

```
TD_KNN(
    ON { table | view | (query) } AS TestTable PARTITION BY ANY
    ON { table | view | (query) } AS TrainingTable DIMENSION
    USING
        IDColumn('id_col_name')
        InputColumns({'target_column'| target_column_range}[,...])
        [ ModelType('classification'|'regression'|'neighbors') ]
        [ K(k) ]
        [ Accumulate({'accumulate_column'|accumulate_column_range}[,...]) ]
```

```
[ ResponseColumn(['response_column']) ]
[ VotingWeight(voting_weight) ]
[ Tolerance(tolerance) ]
[ OutputProb('true' | 'false' | 't' | 'yes' | 'y' | '1' | 'f' | 'no' | 'n' | '0') ]
[ Responses('response_list') ]
[ EmitNeighbors('true' | 'false' | 't' | 'yes' | 'y' | '1' | 'f' | 'no' | 'n' | '0') ]
[ EmitDistances('true' | 'false' | 't' | 'yes' | 'y' | '1' | 'f' | 'no' | 'n' | '0') ]

)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_KNN

### **ON clause**

Accepts the TestTable and TrainingTable clauses.

### **IDColumn**

Specifies the column name that uniquely identifies a data object in the training and the test table. Each entry in the column must be unique, otherwise an error appears.

### **InputColumns**

Specifies the training table column names that the function uses to compute the distance between a test object and a training object. The test table must also have the same training table column names.

## Optional Syntax Elements for TD\_KNN

### **ModelType**

Specifies the model type for the KNN function.

Accepted Values: Regression, Classification, or Neighbors.

Default: Classification

**K**

Specifies the number of nearest neighbors to use in the algorithm. Select any positive integer value greater than 0 or less than or equal to 100.

Default: 5

**Accumulate**

Specifies the test table column names to copy to the output table.

**ResponseColumn**

[For the Regression or Classification model type] Specifies the training table column name that contains the numeric response variable values used for prediction in KNN-based regression or classification.

The class labels under the Response column must be numeric.

Invalid for the Neighbors model type.

**VotingWeight**

[For the Regression or Classification model type] Specifies the voting weight of the training object to determine the test object class as a function of the distance between the training and the test object.

The weighted voting score is calculated as  $w=1/\text{POWER}(\text{distance}, \text{voting\_weight})$ , where distance is the distance between the test object and the training object. The voting\_weight value must be a non-negative real number.

Default: 0

**Tolerance**

Specifies the tolerance value to define the smallest distance.

The weight ( $w=1/\text{POWER}(\text{distance}, \text{voting\_weight})$ ) is undefined if the distance is zero and a non-zero voting\_weight is used.

For any distance less than the specified tolerance, the weight is calculated as  $w=1/\text{POWER}(\text{tolerance}, \text{voting\_weight})$ .

Default: 0.0000001

**OutputProb**

[For the Classification model type] Specifies whether you want to return the probability of each response specified in the Responses argument. If the Responses argument is not specified, then the function returns the probability of the predicted response when it is set to True.

Default: False

Invalid for the Regression or the Neighbors model type

### Responses

[When the OutputProb argument is true] Specifies the class labels for returning probabilities.

The class labels are transformed to integer values only.

The maximum value is 1000.

### EmitNeighbors

[For the Classification or Regression model type] Specifies whether to display neighbors in the output. The default value is true for the Neighbors model type, or you can manually set the value to true. However, you cannot set the value to false for the Neighbors model type.

The default value for the Classification or the Regression model type is false.

### EmitDistances

Specifies whether to display the neighbor distances in the output.

Default: False

## TD\_KNN Input

The function accepts only two ON clauses for input tables, where the Test table uses the PARTITION BY ANY clause, and the training table uses the dimension clause. Both the test and training tables are required.

### TrainingTable Schema

Column	Data Type	Description
IDColumn	BYTEINT,SMALLINT, INTEGER,BIGINT	The unique identifier of the training table.
InputColumns	NUMERIC	The columns that the function uses to compute the distance between test data points and the training data points. The test table must have corresponding columns in the training table with name and datatype.
ResponseColumn	NUMERIC (regression), INTEGER (classification)	The numeric value of the target variable used for prediction in KNN-based regression or classification.

## TestTable Schema

Column	Data Type	Description
IDColumn	BYTEINT, SMALLINT, INTEGER,BIGINT	The unique identifier of the test data object.
InputColumns	NUMERIC	The columns that the function uses to compute the distance between test data points and the training data points. The training table must have corresponding columns in the test table with name and datatype.

## TD\_KNN Output

### Output Table Schema

Column	Data Type	Description
id_column	Same as IDColumn in test table	The unique identifier from the test table.
prediction	Same as ResponseColumn in training table	The predicted response value calculated by the KNN regression or classification model type.
prob	Double Precision	The probability of the predicted class. Only displays if the OutputProb element value is true and the Responses element value is not specified.
prob_k	Double Precision	The probability of the k <sup>th</sup> response specified by the Responses argument. Only displays if the Responses argument is specified.
neighbor_idk	Same as the id_column in the training Table	The unique identifier of the test data object for the neighbor k.
neighbor_distk	Double Precision	The Euclidean distance of the neighbor_idk value from the test data point.
accumulate_column	Data type of the column being copied	The specified column names in the Accumulate element is copied to the output.

## Examples: How to Use TD\_KNN

- [Example: Using TD\\_KNN To Predict Weight Based on Height and Age](#)
- [Example: Using TD\\_KNN with Classification](#)
- [Example: Using TD\\_KNN with Neighbors](#)

## Example: Using TD\_KNN To Predict Weight Based on Height and Age

### TD\_KNN Input Table for Regression Example

**Person\_train**

<b>id</b>	<b>height</b>	<b>age</b>	<b>weight</b>
0	5	32	67
1	5.11	45	98
2	5.9	46	78
3	4.8	35	86
4	5.8	22	70
...	...	...	...

### Example: TD\_KNN SQL Call for Regression

```
SELECT * FROM TD_KNN(
ON person_test AS TestTable PARTITION BY ANY
ON person_train AS TrainingTable DIMENSION
USING
K(2)
ResponseColumn('weight')
InputColumns('height','age')
IDColumn('id')
ModelType('regression')
EmitNeighbors('true')
EmitDistances('true')
) AS dt ;
```

### TD\_KNN Output Table for Regression

<b>id</b>	<b>prediction</b>	<b>neighbor_id1</b>	<b>neighbor_dist1</b>	<b>neighbor_id2</b>	<b>neighbor_dist2</b>
11	88	4	0.7	2	1.004
12	80	8	20.00	4	41.00

## Example: Using TD\_KNN with Classification

### TD\_KNN Input Table for Classification

encoded	ROW_I	attribute_1	attribute_2	attribute_3	...	attribute_49	sample_id
0	99	-0.0664	-0.0999	-0.0949	...	-0.0942	2
0	101	-0.603	-0.0938	-0.0900	...	-0.0935	2
1	114	0.0000	0.0001	0.0001	...	0.0001	2
1	115	0.0001	0.0001	0.0001	...	0.0001	2
...	...	...	...	...	...	...	...

### Example: TD\_KNN SQL Call for Classification

```

CREATE VOLATILE TABLE KNN AS (
SELECT * FROM TD_KNN (
    ON test_dataset AS TestTable PARTITION BY ANY
    ON train_dataset AS TrainingTable DIMENSION
    USING
        K(3)
        ResponseColumn('encoded')
        InputColumns('[2:7]')
        IDColumn('Row_I')
        Accumulate ('encoded')
        ModelType('Classification')
        OutputProb('true')
        EmitNeighbors('true')
        Responses('0', '1')
) AS dt
)WITH DATA
ON COMMIT PRESERVE ROWS;

```

### TD\_KNN Output Table for Classification

```
SELECT * FROM KNN;
```

ROW_I	prediction	prob_0	prob_1	neighbor_id1	neighbor_id2	neighbor_id3	encoded
43	1	0.3333	0.6666	146	42	145	0
101	0	1.0	0.0	100	102	103	0

150	0	0.6666	0.3333	48	47	128	1
192	1	0.0	1.0	191	193	190	1

## Example: Using TD\_KNN with Neighbors

### TD\_KNN Input Table for Neighbors

Id	MSSubClass	MSZoning	LotFrontage	LotArea	...	SaleCondition	SalePrice
1	60	RL	65	8450	...	Normal	208500
2	20	RL	80	9600	...	Normal	181500
3	60	RL	68	11250	...	Normal	22350
4	70	RL	60	9550	...	Abnormal	140000
5	60	RL	84	14260	...	Normal	250000
...	...	...	...	...	...	...	...

### Example: TD\_KNN SQL Call for Neighbors

```
SELECT * FROM TD_KNN(
ON housing_train AS TestTable PARTITION BY ANY
ON housing_train AS TrainingTable DIMENSION
USING
k(5)
InputColumns('[1:5]')
IDColumn('sn')
ModelType('Neighbors')
) AS dt;
```

### TD\_KNN Output Table for Neighbors

sn	neighbor_id1	neighbor_id2	neighbor_id3	neighbor_id4	neighbor_id5
1	106	28	27	18	1
2	23	55	2	24	15
3	3	83	134	81	33
4	4	5	66	35	87
5	35	4	5	87	66

## TD\_NaiveBayes

TD\_NaiveBayes classification algorithm takes a training data set with known discrete outcomes and either discrete or continuous numeric input variables and categorical variables and generates a model that you can use to predict the outcome of future observations based on their input variable values. The model assumes that, given the outcome, the input variables are independent of each other.

The Naive Bayes algorithm is a supervised learning algorithm, which is based on the Bayes theorem and used for solving classification problems. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

### Function Information

- [TD\\_NaiveBayes Syntax](#)
- [Required Syntax Elements for TD\\_NaiveBayes](#)
- [Optional Syntax Elements for TD\\_NaiveBayes](#)
- [TD\\_NaiveBayes Input](#)
- [TD\\_NaiveBayes Output](#)
- [Example: How to Use TD\\_NaiveBayes](#)

## TD\_NaiveBayes Syntax

```
TD_NaiveBayes(
  ON { table | view | (query) } AS InputTable
  USING
  ResponseColumn('response_column')
  { for_dense_input | for_sparse_input }
)
```

### for\_dense\_input

```
[ NumericInputs ({ 'numeric_column' |
  'numeric_column_range' }[,...] ) ]
[ CategoricalInputs ({ 'categorical_column' |
  'categorical_column_range' }[,...] ) ]
```

### for\_sparse\_input

```
AttributeNameColumn('attribute_name_column')
AttributeValueColumn('attribute_value_column')
{
  [ NumericAttributes('numeric_attribute' [,...]) ]
  [ CategoricalAttributes('categorical_attribute' [,...]) ]
```

```

| AttributeType('{ALLNUMERIC | ALLCATEGORICAL}')
}

```

## Required Syntax Elements for TD\_NaiveBayes

### ON clause

Specifies the table name, view name, or query as InputTable.

### ResponseColumn

Specifies the column from the input table which contains the response values.

## Optional Syntax Elements for TD\_NaiveBayes

### NumericInputs

Specify columns from the input table which contains the numeric attributes values.

---

#### Note:

Required only when input is in dense format and if you omit *CategoricalInputs*.

---

### CategoricalInputs

Specify columns from the input table which contains the categorical attributes values.

---

#### Note:

Required only when input is in dense format and if you omit *NumericInputs*.

---

### AttributeNameColumn

Specifies the column from the input table which contains the attributes.

---

#### Note:

Required only when input is in sparse format.

---

### AttributeValueColumn

Specifies the column from the input table which contains the attribute values.

**Note:**

Required only when input is in sparse format.

**NumericAttributes**

Specify the attributes names which are numeric.

**Note:**

Required only when input is in sparse format and if you omit *AttributeType* and *CategoricalAttributes*.

**CategoricalAttributes**

Specify the attributes names which are categorical.

**Note:**

Required only when input is in sparse format and if you omit *AttributeType* and *NumericAttributes*.

**AttributeType**

Specifies ALLNUMERIC if all the attributes are numeric type.

Specifies ALLCATEGORICAL if all the attributes are categorical type.

**Note:**

Required only when input is in sparse format and if you omit both *NumericAttributes* and *CategoricalAttributes*.

## TD\_NaiveBayes Input

### InputTable Schema for Dense Input

Column Name	Data Type	Description
response_column	BYTEINT, SMALLINT, INT, BIGINT, CHAR, VARCHAR	Column containing the response values.
numeric_input_column	BYTEINT, SMALLINT, INT, BIGINT, REAL, DECIMAL, NUMBER	Column containing the numeric attribute values.
categorical_input_column	CHAR, VARCHAR	Column containing the categorical attribute values.

**InputTable Schema for Sparse Input**

Column Name	Data Type	Description
response_column	BYTEINT, SMALLINT, INT, BIGINT, CHAR, VARCHAR	Column containing the response values.
attribute_name_column	CHAR, VARCHAR	Column containing the attributes names.
attribute_value_column	BYTEINT, SMALLINT, INT, BIGINT, REAL, DECIMAL, NUMBER, CHAR, VARCHAR	Column containing the attributes values.

**TD\_NaiveBayes Output****OutputTable Schema**

Column Name	Data Type	Description
class	VARCHAR	Column containing the response values.
variable	VARCHAR	Column containing the attributes.
type	VARCHAR	Input variable types ('NUMERIC' or 'CATEGORICAL').
category	VARCHAR	If type is 'NUMERIC', then NULL. If type is 'CATEGORICAL', then Category of categorical variable.
cnt	BIGINT	Count of observations with this class, variable, and category.
sum	REAL	If type is 'NUMERIC', then Sum of variable values for observations with this class, variable, and category. If type is 'CATEGORICAL', then NULL.
sumsq	REAL	If type is 'NUMERIC', then Sum of square of variable values for observations with this class, variable, and category. If type is 'CATEGORICAL', then NULL.
totalcnt	BIGINT	Total count of observations.
smoothingfactor	REAL	If type is 'NUMERIC', then NULL. If type is 'CATEGORICAL', then smoothing factor with this class, variable.

**Example: How to Use TD\_NaiveBayes**

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

This section shows the input table, SQL query, and output tables of an example using TD\_NaiveBayes.

The following table contains a subset of housing dataset.

**Note:**

Only part of the dataset is shown in this example.

```
DROP TABLE housing_train;
CREATE TABLE housing_train (sn integer, price real, lotsize real, bedrooms
integer, bathrms integer, stories integer, driveway varchar(10), recroom
varchar(10), fullbase varchar(10), gashw varchar(10), airco varchar(10), garagepl
integer, prefarea varchar(10), homestyle varchar(10)) PRIMARY INDEX(sn);
Insert into housing_train
values(1,42000,5850,3,1,2,'yes','no','yes','no','no',1,'no','Classic');
Insert into housing_train
values(2,38500,4000,2,1,1,'yes','no','no','no','no',0,'no','Classic');
Insert into housing_train
values(3,49500,3060,3,1,1,'yes','no','no','no','no',0,'no','Classic');
Insert into housing_train
values(4,60500,6650,3,1,2,'yes','yes','no','no','no',0,'no','Eclectic');
Insert into housing_train
values(5,61000,6360,2,1,1,'yes','no','no','no','no',0,'no','Eclectic');
...
SELECT * FROM housing_train ORDER BY 1;
```

### TD\_NaiveBayes SQL Call

```
SELECT * FROM TD_NaiveBayes(
ON housing_train AS InputTable
USING
ResponseColumn('homestyle')
NumericInputs('price','lotsize','bedrooms','bathrms','stories','garagepl')
CategoricalInputs('driveway','recroom','fullbase','gashw','airco','prefarea')
) AS dt ORDER BY 1,2;
```

### TD\_NaiveBayes Output Table

class	variable	type	category	cnt	sum	sumsq	totalcnt	smoothingfactor
bungalow	airco	CATEGORICAL	no	14	?	?	392	1. 72413793103448E-002
bungalow	airco	CATEGORICAL	yes	42	?	?	392	1. 72413793103448E-002
bungalow	bathrms	NUMERIC	?	56	1. 050000000000000E 002	2. 190000000000000E 002	392	?
...	...	...	...	...	...	...	...	...

class	variable	type	category	cnt	sum	sumsq	totalcnt	smoothingfactor
Classic	recroom	CATEGORICAL	no	133	?	?	980	7. 04225352112676E-003
Classic	recroom	CATEGORICAL	yes	7	?	?	980	7. 04225352112676E-003
Classic	stories	NUMERIC	?	140	2. 070000000000000E 002	3. 490000000000000E 002	980	?
...	...	...	...	...	...	...	...	...
Eclectic	recroom	CATEGORICAL	no	233	?	?	2072	3. 35570469798658E-003
Eclectic	recroom	CATEGORICAL	yes	63	?	?	2072	3. 35570469798658E-003
Eclectic	stories	NUMERIC	?	296	5. 310000000000000E 002	1. 153000000000000E 003	2072	?

As the data is in dense format, you need to use TD\_Unpivoting function to change the data to sparse format.

```
DROP TABLE housing_train_sparse;
CREATE MULTISET TABLE housing_train_sparse AS(
SELECT * FROM TD_UNPIVOTING(
ON housing_train AS InputTable
USING
IDColumn('sn')
TargetColumns('price','lotsize','bedrooms','bathrms','stories','garagepl','drive
way','recroom','fullbase','gashw','airco','prefarea')
AttributeName('AttributeName')
ValueColumnName('AttributeValue')
Accumulate('homestyle')
) AS dt) WITH data;
```

## TD\_NaiveBayes SQL Call

```
SELECT * FROM TD_NaiveBayes(
ON housing_train_sparse AS InputTable
USING
ResponseColumn('homestyle')
AttributeNameColumn('AttributeName')
AttributeValueColumn('AttributeValue')
NumericAttributes('price','lotsize','bedrooms','bathrms','stories','garagepl')
CategoricalAttributes('driveway','recroom','fullbase','gashw','airco','prefarea'
)
) AS dt Order by 1,2, 3, 4;
```

**TD\_NaiveBayes Output Table**

class	variable	type	category	cnt	sum	sumsq	totalcnt	smoothingfactor
bungalow	airco	CATEGORICAL	no	14	?	?	392	1. 72413793103448E-002
bungalow	airco	CATEGORICAL	yes	42	?	?	392	1. 72413793103448E-002
bungalow	bathrms	NUMERIC	?	56	1. 050000000000000E 002	2. 190000000000000E 002	392	?
...	...	...	...	...	...	...	...	...
Classic	recroom	CATEGORICAL	no	133	?	?	980	7. 04225352112676E-003
Classic	recroom	CATEGORICAL	yes	7	?	?	980	7. 04225352112676E-003
Classic	stories	NUMERIC	?	140	2. 070000000000000E 002	3. 490000000000000E 002	980	?
...	...	...	...	...	...	...	...	...
Eclectic	recroom	CATEGORICAL	no	233	?	?	2072	3. 35570469798658E-003
Eclectic	recroom	CATEGORICAL	yes	63	?	?	2072	3. 35570469798658E-003
Eclectic	stories	NUMERIC	?	296	5. 310000000000000E 002	1. 153000000000000E 003	2072	?

## TD\_OneClassSVM

TD\_OneClassSVM is a linear support vector machine (SVM) that performs classification analysis on data sets to identify outliers or novelty in the data.

This function supports the Classification (loss: hinge) model. During the training, all the data is assumed to belong to a single class (value 1), therefore ResponseColumn is not needed by the model. For TD\_OneClassSVMPredict, output values are 0 or 1. A value of 0 corresponds to an outlier, and 1 to a normal observation or instance.

TD\_OneClassSVM is implemented using Minibatch Stochastic Gradient Descent (SGD) algorithm, which is highly scalable for large datasets. See [TD\\_GLM](#) for information about SGD.

The function output is a trained one-class SVM model, which can be input to TD\_OneClassSVMPredict for prediction. The model also contains model statistics of MSE, Loglikelihood, AIC, and BIC.

One-class SVM is a machine learning algorithm used for anomaly detection to identify data points that deviate significantly from the norm. It is a variant of Support Vector Machines (SVM) that is designed to work with only one class of data.

One-class SVM works by first mapping the input data to a high-dimensional feature space and then finding the hyperplane that separates the data from the origin with the largest margin. The hyperplane is then used to classify new data points as either belonging to the same class as the training data or not.

One of the main advantages of TD\_OneClassSVM is that it can be used with unlabeled data, meaning it can detect anomalies without requiring labeled data for training. This makes it particularly useful for detecting fraud, intrusion detection, and other types of security applications where anomalous behavior is often the most important signal.

However, TD\_OneClassSVM has some limitations, such as being sensitive to the choice of kernel function, and not being suitable for datasets with high levels of noise or a significant overlap between classes.

One-class SVM remains a powerful tool for anomaly detection and is widely used in industry and research.

The key mathematical concepts that underlie TD\_OneClassSVM include:

- Kernel Functions: Mathematical functions that are used to measure the similarity between two data points in a higher-dimensional feature space. In TD\_OneClassSVM, the kernel function is used to map the data points into the feature space. Common kernel functions used in TD\_OneClassSVM include the Gaussian, polynomial, and sigmoid kernels.
- Support Vectors: Data points that lie closest to the decision boundary. They are used to define the hyperplane that separates the normal data points from the anomalies. The weights and biases of the hyperplane are calculated based on the support vectors.
- Lagrange Multipliers: Multipliers to optimize the problem of finding the hyperplane that maximizes the margin. The Lagrange multipliers are used to constrain the solution to satisfy certain conditions, such as the requirement that the weights of the hyperplane sum to zero.
- Quadratic Programming: Mathematical optimization technique that is used to solve this problem efficiently. The problem of finding the hyperplane that maximizes the margin can be formulated as a quadratic programming problem.
- Convex Optimization: Branch of mathematics that studies the optimization of convex functions, which are functions that have a non-negative second derivative and are always either increasing or decreasing. The objective function is convex and has a unique global minimum.

## Function Information

- [TD\\_OneClassSVM Syntax](#)
- [Required Syntax Elements for TD\\_OneClassSVM](#)
- [Optional Syntax Elements for TD\\_OneClassSVM](#)
- [TD\\_OneClassSVM Input](#)
- [TD\\_OneClassSVM Output](#)
- [TD\\_OneClassSVM Usage Notes](#)
- [Example: How to Use TD\\_OneClassSVM](#)

## TD\_OneClassSVM Syntax

### Important:

In Analytics Database Release 17.20.03.14 and later, the AS InputTable alias is mandatory. If your scripts does not use an alias, use AS InputTable alias for the input table.

```

TD_OneClassSVM (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    [ OUT TABLE MetaInformationTable (meta_table) ]
    USING
        InputColumns ({{'input_column'|input_column_range }[,...]})
        [ BatchSize (batchsize) ]
        [ MaxIterNum (max_iter) ]
        [ RegularizationLambda (lamda) ]
        [ Alpha (alpha) ]
        [ IterNumNoChange (n_iter_no_change) ]
        [ Tolerance (tolerance) ]
        [ Intercept ('true'||'t'||'yes'||'y'||'1'||'false'||'f'||'no'||'n'||'0') ]
        [ LearningRate ('constant'||'optimal'||'invtime'||'adaptive') ]
        [ InitialEta (eta0) ]
        [ DecayRate (gamma) ]
        [ DecaySteps (decay steps) ]
        [ Momentum (momentum) ]
        [ Nesterov ('true'||'t'||'yes'||'y'||'1'||'false'||'f'||'no'||'n'||'0') ]
        [ LocalSGDIterations(local_iterations) ]
)

```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_OneClassSVM

### ON clause

Accepts the PARTITION BY ANY clause.

### InputColumns

Specify the names of the input table columns that need to be used for training the model (predictors, features or independent variables).

## Optional Syntax Elements for TD\_OneClassSVM

### **OUT clause**

Accepts the MetaInformationTable clause.

### **MaxIterNum**

Specify the maximum number of iterations (minibatches) over the training data batches. Value is a positive integer less than 10,000,000. Default value is 300.

### **BatchSize**

Specify the number of observations (training samples) processed in a single minibatch per AMP. A value of 0 or higher than the number of rows on an AMP processes all rows on the AMP, such that the entire dataset is processed in a single iteration, and the algorithm becomes Gradient Descent. You must specify a non-negative integer. Default value is 10.

### **RegularizationLambda**

Specify the amount of regularization to be added. The higher the value, stronger the regularization. It is also used to compute learning rate when learning rate is set to optimal. Must be a non-negative float value. A value of 0 means no regularization. Default is 0.02.

### **Alpha**

Specify the Elasticnet parameter for penalty computation. It is only effective when RegularizationLambda is greater than 0. The value represents the contribution ratio of L1 in the penalty. A value of 1.0 indicates L1 (LASSO) only, a value of 0 indicates L2 (Ridge) only, and a value between is a combination of L1 and L2. Value is a float value between 0 and 1. Default is 0.15.

### **IterNumNoChange**

Specify the number of iterations (minibatches) with no improvement in loss including the tolerance to stop training. A value of 0 indicates no early stopping and the algorithm continues until MaxIterNum iterations are reached. Value is a non-negative integer. Default is 50.

### **Tolerance**

Specify the stopping criteria in terms of loss function improvement. Applicable when IterNumNoChange is greater than 0. Value is a positive integer. Default is 0.001.

### **Intercept**

Specify whether to estimate the intercept based on whether data is already centered. Default is true.

**LearningRate**

Specify the learning rate algorithm. Learning rates are:

- Constant
- InvTime
- Optimal
- Adaptive

Default is invtime for Gaussian, optimal for Binomial.

**InitialEta**

Specify the initial value of eta for learning rate. For LearningRate set to constant, this value is the learning rate for all iterations. Value is numeric. Default is 0.05.

**DecayRate**

Specify the decay rate for learning rate. Only applicable for learning rates invtime and adaptive. Value is numeric. Default is 0.25.

**DecaySteps**

Specify the number of iterations without decay for the adaptive learning rate. The learning rate changes by decay rate after this many iterations. Value is integer. Default is 5.

**Momentum**

Specify the value to use for momentum learning rate optimizer. A larger value indicates higher momentum contribution. A value of 0 means momentum optimizer is disabled. For a good momentum contribution, a value between 0.6-0.95 is recommended. Value is a non-negative float between 0 and 1. Default is 0.

**Nesterov**

Specify whether to apply Nesterov optimization to Momentum optimizer. Only applicable when Momentum is greater than 0. Default is false.

**LocalSGDIterations**

Specify the number of local iterations to be used for Local SGD algorithm. A value of 0 implies Local SGD is disabled. A value higher than 0 enables Local SGD and multiple, equal to the value supplied by the user. With Local SGD algorithm, the recommended values for arguments are:

- LocalSGDIterations: 10
- MaxIterNum: 100
- BatchSize: 50
- IterNumNoChange: 5

Value is a positive integer. Default is 0.

## TD\_OneClassSVM Input

TD\_OneClassSVM accepts one input table containing all input columns. No response column is needed.

Column Name	Data Type	Description
input_column	INTEGER, BIGINT, SMALLINT, BYTEINT, DOUBLE PRECISION	Column that this function uses to train the SVM model.

## TD\_OneClassSVM Output

TD\_OneClassSVM produces two outputs:

- Model (Primary output): Contains the trained model along with model statistics. These model statistics will be stored in the model:
  - Loss Function
  - MSE
  - Number of Observations
  - Loglik
  - AIC
  - BIC
  - Regularization
  - Alpha (L1/L2/Elasticnet)
  - Number of Iterations
  - Learning Rate (Initial)
  - Learning Rate (Final)
  - Momentum
  - Nesterov
  - LocalSGD Iterations
- MetaInformationTable (Secondary Output - Optional): Contains training progress information for each epoch.

### Model Output Schema

Column	Data Type	Description
attribute	SMALLINT	Contains a numeric index of predictor and model metrics. Intercept is specified using index 0 and rest of the predictors take positive values. Model metrics take negative indices.
predictor	VARCHAR	Contains the name of the predictor or Model metric.

Column	Data Type	Description
estimate	FLOAT	Contains the predictor weights and numeric-based metric values.
value	VARCHAR	Contains the string-based metric values (for example., HINGE for LossFunction, L2 for Regularization, and so on).

### MetaInformationTable Output Schema

Column	Data Type	Description
iteration	INTEGER	Contains iteration number (epoch number).
num_rows	BIGINT	Contains total number of rows processed so far.
eta	FLOAT	Learning rate for the iteration.
loss	FLOAT	Contains loss in this iteration.
best_loss	FLOAT	Contains best loss till this iteration.

### TD\_OneClassSVM Usage Notes

- The categorical columns are converted to numerical columns as preprocessing step (for example, using TD\_OneHotEncoding, TD\_OrdinalEncoding, TD\_TargetEncoding). TD\_SVM takes all features as numeric input.
- For a good model, standardize the dataset before feeding to TD\_OneClassSVM as a preprocessing step (for example, using TD\_Scale).
- The rows with missing values are ignored during training and prediction of TD\_OneClassSVM and TD\_OneClassSVMPredict. Consider filling up those rows using imputation (TD\_SimpleImpute) or other mechanism to train on rows with missing values.
- The function supports linear SVMs only.
- A maximum of 2046 features are supported due to the limitation imposed by the maximum number of columns (2048) in a database table for TD\_OneClassSVM.

### Example: How to Use TD\_OneClassSVM

This is a sample of starting diabetes information.

ID	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
5	1	139	80	23	140	27.1	1.441
1	5	116	74	0	0	25.6	.0201
8	2	197	70	45	45	30.5	0.158
2	2	87	58	16	16	32.7	0.166

ID	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
4	0	128	68	19	19	30.5	0.121
6	4	130	70	0	0	34.2	0.652
7	10	115	0	0	0	35.3	0.134
3	7	129	68	49	49	38.5	0.439
10	4	110	92	0	0	37.6	0.191
9	0	125	96	0	0	22.5	0.262

### Example: TD\_OneClassSVM Using LearningRate Constant

```
SELECT * FROM TD_OneClassSVM(
    ON diabetes_train_scaled AS InputTable
    USING
        InputColumns('[1:8]')
        Tolerance(1e-7)
        BatchSize(30)
        LearningRate('constant')
        InitialEta (0.01)
        RegularizationLambda(0.1)
        Alpha(0)
        Momentum (0.0)
        Nesterov ('false')
        MaxIterNum (100)
) AS dt
```

### TD\_OneClassSVM Output

Attribute	Predictor	Estimate	Value
0	(Intercept)	-0.002	
1	Pregnancies	0.0007820357	
2	Glucose	-0.0009073822	
3	BloodPressure	-0.0005117763	
4	SkinThickness	-0.0001402478	
5	Insulin	0.0013394897	
6	BMI	0.0004039902	
7	DiabetesPedigreeFunction	-0.0004613734	

Attribute	Predictor	Estimate	Value
8	Age	-0.0011434892	
-1	Loss Function		HINGE
-2	Number of Observations	537	
-3	MSE	-0.0001951275	
-4	AIC	18.0003902551	
-5	BIC	56.5743731056	
-6	Regularization	0.1	ENABLED
-7	Alpha	0	L2
-8	Number of Iterations	59	CONVERGED
-9	Learning Rate (Initial)	0.01	
-10	Learning Rate (Final)	0.01	
-11	Momentum	0	
-12	Nesterov		FALSE
-13	LocalSGD Iterations	0	
-15	Intercept Scaling	1	
-16	Sparse		FALSE
-17	Kernel		LINEAR
-18	OneClass SVM		TRUE

## TD\_SVM

TD\_SVM function is a linear support vector machine (SVM) that performs classification and regression analysis on datasets.

This function supports these models:

- Regression (loss: epsilon\_insensitive).
- Classification (loss: hinge). Only supports binary classification. The only response values are 0 or 1.

TD\_SVM is implemented using Minibatch Stochastic Gradient Descent (SGD) algorithm, which is highly scalable for large datasets. See [TD\\_GLM](#) for details on SGD.

Support Vector Machines (SVM) is a type of supervised machine learning algorithm used for classification and regression analysis. The goal of SVM is to find a hyperplane that best separates the data points into different classes while maximizing the margin, such as the distance between the hyperplane and the nearest data points from both classes.

SVM is a machine learning algorithm used for classification and regression analysis that has been proven to be effective in solving various real-world problems. Its ability to handle high-dimensional data, non-linearly separable data, and outliers make it a popular choice for many applications.

The following procedure is an example of how to use TD\_SVM:

1. Standardize the Input features using [TD\\_ScaleFit](#) and [TD\\_ScaleTransform](#) functions. The function only accepts numeric features.
2. Before training, convert the categorical features to numeric values. The function skips the rows with missing (null) values during training.

The function output is a trained SVM model, which can be used as input to [TD\\_SVMPredict](#) for prediction. The model also contains model statistics of mean squared error (MSE), Loglikelihood, Akaike information criterion (AIC), and Bayesian information criterion (BIC).

3. Perform model evaluation as a post-processing step using functions such as [TD\\_RegressionEvaluator](#), [TD\\_ClassificationEvaluator](#), and [TD\\_ROC](#).

The optimization problem of SVM can be formulated as:

$$\text{minimize: } \frac{1}{2} * \|w\|^2 + C * \sum(x_i)$$

where:

- $\|w\|$  is the Euclidean norm of the weight vector  $w$
- $C$  is a parameter that controls the trade-off between maximizing the margin and minimizing the classification error
- $x_i$  is the slack variable that measures the degree of misclassification
- $\sum(x_i)$  is the total misclassification error

subject to:  $y_i(w^*x_i+b) \geq 1 - x_i$  where:

- $y_i$  is the class label (either -1 or 1)
- $x_i$  is the slack variable
- $w$  is the weight vector
- $b$  is the bias term
- $(x_i + b)$  is the decision boundary

In the case of linearly-separable data, the optimal hyperplane can be found by solving the optimization problem. However, in the case of non-linearly separable data, a kernel function can be used to map the data into a higher-dimensional space where a hyperplane can be used to separate the classes. The most commonly used kernel functions are:

- Linear kernel:

$$K(x_i, x_j) = x_i^* x_j$$

- Polynomial kernel:

$$K(x_i, x_j) = (\gamma x_i^* x_j + r)^d$$

where  $\gamma$ ,  $r$ , and  $d$  are parameters.

- Radial basis function (RBF) kernel:

$$K(x_i, x_j) = \exp(-\gamma |(x_i - x_j)|^2)$$

where  $\gamma$  is a parameter.

Once the data is mapped into a higher-dimensional space, the optimization problem can be solved to find the optimal hyperplane. The decision boundary is then given by:

$$w^T \phi(x_i) + b = 0$$

where  $\phi(x_i)$  is the feature vector in the higher-dimensional space.

SVM can be trained using the Minibatch Stochastic Gradient Descent (SGD) algorithm, which is a popular optimization algorithm for large-scale machine learning problems. Minibatch SGD updates the weight vector incrementally using small batches of data, making it more computationally efficient than batch SGD.

The objective function for SVM using Minibatch SGD can be formulated as follows:

minimize:

$$\frac{1}{2} \|w\|^2 + C * \frac{1}{b} \sum \max(0, 1 - y_i(w^T x_i + b))$$

where

- $\|w\|$  is the Euclidean norm of the weight vector  $w$
- $C$  is a parameter that controls the trade-off between maximizing the margin and minimizing the classification error
- $y_i$  is the class label (either -1 or 1)
- $x_i$  is the feature vector
- $b$  is the bias term
- $b$  is the size of the minibatch

At each iteration, a minibatch of data points is randomly selected from the training set, and the weight vector is updated using the following equation:

$$w = w - \eta * (\lambda * w - \frac{1}{b} \sum \max(0, 1 - y_i(w^T x_i + b)) * y_i * x_i)$$

where

- $\eta$  is the learning rate
- $\lambda$  is the regularization parameter
- $x_i$  is the feature vector in the minibatch

The weight vector is then updated iteratively until convergence.

Minibatch SGD has several advantages over batch SGD, including faster convergence and better generalization performance. However, it also has some disadvantages, such as sensitivity to the choice of the learning rate and the minibatch size.

## Function Information

- [TD\\_SVM Syntax](#)
- [Required Syntax Elements for TD\\_SVM](#)

- [Optional Syntax Elements for TD\\_SVM](#)
- [TD\\_SVM Input](#)
- [TD\\_SVM Output](#)
- [Examples: How to Use TD\\_SVM](#)

## TD\_SVM Syntax

---

### Important:

In Analytics Database Release 17.20.03.14 and later, the AS InputTable alias is mandatory. If your scripts does not use an alias, use AS InputTable alias for the input table.

---

```
TD_SVM (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    [ OUT TABLE MetaInformationTable (meta_table) ]
    USING
        InputColumns ({{'input_column'|input_column_range }[,...]})
        ResponseColumn('response_column')
        [ ModelType ('Classification' | 'Regression') ]
        [ BatchSize (batchsize) ]
        [ MaxIterNum (max_iter) ]
        [ Epsilon (epsilon) ]
        [ RegularizationLambda (lambda) ]
        [ Alpha (alpha) ]
        [ IterNumNoChange (n_iter_no_change) ]
        [ Tolerance (tolerance) ]
        [ Intercept ('true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0') ]
        [ ClassWeights ('class:weight,...') ]
        [ LearningRate ('constant'|'optimal'|'invtime'|'adaptive') ]
        [ InitialEta (eta0) ]
        [ DecayRate (gamma) ]
        [ DecaySteps (decay_steps) ]
        [ Momentum (momentum) ]
        [ Nesterov ('true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0') ]
        [ LocalSGDIterations(local_iterations) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_SVM

**ON clause**

Accepts the InputTable clause.

**InputColumns**

Specifies the name of the input table columns that need to be used for training the model (predictors, features, or independent variables).

**ResponseColumn**

Specifies the name of the column that contains the class label for classification or target value (dependent variable) for regression.

## Optional Syntax Elements for TD\_SVM

**OUT clause**

Accepts the MetaInformationTable clause.

**ModelType**

Specifies the type of the analysis. Acceptable values are Regression, Classification.

Default: Classification

**MaxIterNum**

Specifies the maximum number of iterations (minibatches) over the training data batches. The value is a positive integer less than 10,000,000.

Default: 300

**Epsilon**

Specifies the epsilon threshold for Regression (the value of epsilon for epsilon\_insensitive loss). Any difference between the current prediction and the correct label is ignored within this threshold. The value is numeric.

Epsilon is a numeric value that sets a threshold for the epsilon-insensitive loss, which means that any difference between the predicted value and the actual label within this threshold is ignored.

This means that the SVM algorithm allows for a margin of error, or "slack" when making predictions for regression problems. The value of epsilon determines the size of this margin of error and affects the tradeoff between the accuracy of the model and its ability to generalize to new data.

Default: 0.1

### **BatchSize**

Specifies the number of observations (training samples) processed in a single minibatch per AMP. A value of 0 or higher than the number of rows on an AMP processes all rows on the AMP, such that the entire dataset is processed in a single iteration, and the algorithm becomes Gradient Descent. The value is a positive integer.

Default: 10

### **RegularizationLambda**

Specifies the amount of regularization to be added. The higher the value, stronger the regularization. It must be a positive float value. A value of 0 means no regularization.

RegularizationLambda is a positive float value that specifies the amount of regularization to be added to the model. The higher the value of RegularizationLambda, the stronger the regularization, which means that the algorithm places a greater emphasis on preventing overfitting to the training data.

In addition to its use in regularization, RegularizationLambda is also used to compute the learning rate when learning rate is set to optimal. Learning rate is a parameter that controls the step size at each iteration of the optimization algorithm and setting it to optimal means that it is automatically determined based on the value of RegularizationLambda.

Default: 0.02

### **Alpha**

Specifies the Elasticnet parameter for penalty computation. It is only effective when RegularizationLambda is greater than 0. The value represents the contribution ratio of L1 in the penalty. A value of 1.0 indicates L1 (LASSO) only, a value of 0 indicates L2 (Ridge) only, and a value between is a combination of L1 and L2. The value is a float value between 0 and 1.

Default: 0.15

**IterNumNoChange**

Specifies the number of iterations (minibatches) with no improvement in loss, including the tolerance to stop training. A value of 0 indicates no early stopping and the algorithm continues until MaxIterNum iterations are reached. The value is a positive integer.

Default: 50

**Tolerance**

Specifies the stopping criteria in terms of loss function improvement. Applicable when IterNumNoChange is greater than 0. The value is a positive integer.

Default: 0.001

**Intercept**

Specifies whether intercept is to be estimated based on whether data is already centered.

Default: True

**ClassWeights**

Specifies weights associated with classes. Only applicable for Classification. The format is 0:weight, 1:weight. For example, 0:1.0,1:0.5 gives twice the weight to each observation in class 0 compared to class 1. If the weight of a class is omitted, it is assumed to be 1.0.

Default: 0:1.0, 1:1.0

**LearningRate**

Specifies the learning rate algorithm. Learning rates are:

- Constant
- InvTime
- Optimal
- Adaptive

Default: InvTime for Regression, Optimal for Classification

**InitialEta**

Specifies the initial value of eta for learning rate. For LearningRate set to constant, this value is the learning rate for all iterations.

The learning rate controls how much the SVM algorithm adjusts the weights of the model during training. If the learning rate is too low, the model may converge slowly. If the learning rate is too high, the model may fail to converge at all.

The choice of learning rate can have a significant impact on the accuracy and speed of the SVM algorithm, so it is an important parameter to consider when training a model.

Default: 0.05

#### **DecayRate**

Specifies the decay rate for learning rate. Only applicable for learning rates invtime and adaptive.

Default: 0.25

#### **DecaySteps**

Specifies the number of iterations without decay for the adaptive learning rate. The learning rate changes by decay rate after this many iterations.

Default: 5

#### **Momentum**

Specifies the value to use for momentum learning rate optimizer. A larger value indicates higher momentum contribution. A value of 0 means momentum optimizer is disabled. A value between 0.6-0.95 is recommended. The value is a positive float between 0 and 1.

Default: 0

#### **Nesterov**

Specifies whether to apply the Nesterov optimization to Momentum optimizer or not. Only applicable when Momentum is greater than 0.

Default: False

#### **LocalSGDIterations**

Specifies the number of local iterations to be used for Local SGD algorithm. A value of 0 means Local SGD is disabled. A value higher than 0 enables Local SGD and multiple, equal to the value supplied by the user. With Local SGD algorithm, the recommended values for arguments are:

- LocalSGDIterations: 10
- MaxIterNum: 100
- BatchSize: 50
- IterNumNoChange: 5

The value is a positive integer.

Default: 0

## TD\_SVM Input

### InputTable Schema

Column Name	Data Type	Description
input_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	Column used to train the SVM model.
response_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	Column containing the response value for an observation.

## TD\_SVM Output

TD\_SVM produces two outputs:

Model (Primary output): Contains the trained model along with model statics. The model stores the following model statistics:

- Loss Function
- MSE
- Number of Observations
- AIC
- BIC
- Number of Iterations
- Regularization
- Alpha (L1/L2/Elasticnet)
- Learning Rate (Initial)
- Learning Rate (Final)
- Momentum
- Nesterov
- LocalSGD Iterations

[Optional] MetaInformationTable (Secondary Output): Contains training progress information for each epoch.

### Model Output Table Schema

Column	Data Type	Description
attribute	SMALLINT	Contains a numeric index of predictor and model metrics. Intercept is specified using index 0 and rest of the predictors have positive values. Model metrics have negative indices.
predictor	VARCHAR	Contains the name of the predictor or model metric.

Column	Data Type	Description
estimate	FLOAT	Contains the predictor weights and numeric-based metric values.
value	VARCHAR	Contains the string-based metric values for example, HINGE for LossFunction, L2 for Regularization, and so on.

### MetaInformationTable Output Table Schema [Optional]

Column	Data Type	Description
iteration	INTEGER	Contains the iteration number (epoch number).
num_rows	BIGINT	Contains the total number of rows processed so far.
eta	FLOAT	Contains the learning rate for the iteration.
loss	FLOAT	Contains loss in this iteration.
best_loss	FLOAT	Contains best loss till this iteration.

## Examples: How to Use TD\_SVM

### Example: Cal Housing Data Set

Starting Data (cal\_housing\_ex\_raw)

#### Note:

Only part of the dataset is shown in this example.

id	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
14870	1.858	23	3.901	1.077	1025	2.47	32.64	-117.11	0.675
6044	2.114	27	3.855	1.072	1024	4.633	34.05	-117.74	1.109
3593	6.654	32	6.331	0.995	1285	3.104	34.24	-118.48	2.676
9454	1.228	25	5.504	1.154	991	2.629	39.77	-123.23	0.603
18760	3.282	16	5.998	1.076	1414	3.081	40.6	-122.25	1.283
11670	4.5	28	5.102	1.044	2112	2.63	33.84	-118.01	2.021
17768	2.756	29	4.53	1.04	3572	4.603	37.35	-121.85	1.601
244	2.391	44	4.866	1.164	2269	3.72	37.78	-122.22	1.117
5328	2.768	23	3.039	1.064	2031	1.637	34.04	-118.45	2.775
14365	2.164	43	4.533	0.995	392	1.867	32.72	-117.23	2.442
2313	2.486	15	5.468	1.045	649	2.449	36.94	-119.7	0.863
12342	2.588	28	6.268	1.372	3470	2.59	33.84	-116.53	1.59
6558	6.827	36	7.021	1.036	1897	2.71	34.2	-118.11	3.594

<b>id</b>	<b>MedInc</b>	<b>HouseAge</b>	<b>AveRooms</b>	<b>AveBedrms</b>	<b>Population</b>	<b>AveOccup</b>	<b>Latitude</b>	<b>Longitude</b>	<b>MedHouseVal</b>
17157	9.78	20	6.678	0.918	324	2.219	37.43	-122.21	5
18099	5.753	27	6.437	1.027	1259	2.868	37.32	-122.01	4.314

## Standardize Data Using TD\_ScaleFit with Housing Data

```
SELECT * FROM TD_ScaleFit(
    ON cal_housing_ex_raw AS InputTable
    OUT VOLATILE TABLE OutputTable(scaleFitOut_cal_ex)
    USING
    TargetColumns('medinc', 'houseage', 'averooms', 'avebedrms', 'population',
    'aveoccup', 'latitude',
    'longitude')
    ScaleMethod('STD')
) AS dt2;
```

## Output from TD\_ScaleFit

<b>TD_STATYPE</b>	<b>MedInc</b>	<b>HouseAge</b>	<b>AveRooms</b>	<b>AveBedrms</b>	<b>Population</b>	<b>AveOccup</b>	<b>Latitude</b>	<b>Longitude</b>
min	1.228	15	3.039	0.918	324	1.637	32.64	-123.23
max	9.78	44	7.021	1.372	3572	4.633	40.6	-116.53
sum	57.698	416	79.531	16.08	23714	43.208	536.76	-1795.16
count	15	15	15	15	15	15	15	15
null	0	0	0	0	0	0	0	0
variance	5.755	71.352	1.377	0.01	974223. 638	0.746	6.407	5.488
std	2.318	8.161	1.134	0.099	953.559	0.834	2.445	2.263
ustd	2.318	8.161	1.134	0.099	953.559	0.834	2.445	2.263
multiplier	1	1	1	1	1	1	1	1
intercept	0	0	0	0	0	0	0	0
location	3.847	27.733	5.302	1.072	1580.933	2.881	35.784	-119.677
scale	2.318	8.161	1.134	0.099	953.559	0.834	2.445	2.263
globalscale_false	null	null	null	null	null	null	null	null
ScaleMethodNumberMapping: [0:mean,1:sum,2:ustd,3:std,4: range,5:midrange,6:maxabs, 7:rescale]	3	3	3	3	3	3	3	3
missvalue_KEEP	null	null	null	null	null	null	null	null

## Standardize Data Using TD\_ScaleTransform with Housing Data

```
CREATE MULTISET TABLE cal_housing_ex_scaled AS (
  SELECT * from TD_ScaleTransform(
    ON cal_housing_ex_raw AS InputTable
    ON scaleFitOut_cal_ex AS FitTable DIMENSION
    USING
      Accumulate('id', 'MedHouseVal')
  ) AS dt1
) with data;
```

## Output Using TD\_ScaleTransform

<b>id</b>	<b>MedHouseVal</b>	<b>MedInc</b>	<b>HouseAge</b>	<b>AveRooms</b>	<b>AveBedrms</b>	<b>Population</b>	<b>AveOccup</b>	<b>Latitude</b>	<b>Longitude</b>
14870	0.675	-0.858	-0.58	-1.236	0.051	-0.583	-0.492	-1.286	1.134
6044	1.109	-0.747	-0.09	-1.276	0.004	-.584	2.101	-0.709	0.856
3593	2.676	1.211	0.523	0.908	-0.777	-.31	0.268	-0.631	0.529
9454	0.603	-1.13	-0.335	0.178	0.827	-.619	-0.302	1.63	-1.57
18760	1.283	-0.006	-1.438	0.614	0.043	-0.175	0.24	1.969	-1.137
11670	2.021	0.282	0.033	-0.176	-0.287	0.557	-0.3	-0.795	0.737
17768	1.601	-0.47	0.155	-0.681	-.324	2.088	2.065	0.64	-0.96
244	1.117	-0.628	1.993	-0.385	0.929	0.722	1.006	0.816	-1.123
5328	2.775	-0.465	-0.58	-1.996	-0.077	0.472	-1.491	-0.713	0.542
14365	2.442	-0.726	1.871	-0.678	-0.0776	-1.247	-1.215	-1.253	1.081
2313	0.863	-0.0587	-1.56	0.146	-0.27	-0.977	-0.517	0.473	-.01
12342	1.59	-0.543	0.033	0.852	3.035	1.981	-0.349	-0.795	1.391
6558	3.594	1.286	1.013	1.517	-0.367	0.331	-0.204	-0.648	0.692
17157	5	2.56	-0.948	1.214	-1.558	-1.318	-0.793	0.673	1.119
18099	4.314	0.822	-0.09	1.001	-0.452	-0.338	-0.015	0.628	-1.044

## Example: Using TD\_SVM SQL Using Regression Model

```
CREATE VOLATILE TABLE svm_model_cal_housing AS (
  SELECT * FROM TD_SVM (
    ON cal_housing_ex_scaled
    USING
      InputColumns('medinc', 'houseage', 'averooms', 'avebedrms', 'population',
      'aveoccup', 'latitude', 'longitude')
      ResponseColumn('MedHouseVal')
      ModelType('Regression')
      BatchSize(10)
)
```

```

MaxIterNum(300)
) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;

```

## Output from TD\_SVM

attribute	predictor	estimate	value
-17	OneClass SVM	NaN	FALSE
-16	Kernel	NaN	LINEAR
-15	Intercept Scaling	1.000000	None
-14	Epsilon	0.100000	None
-13	LocalSGD Iterations	0.000000	None
-12	Nesterov	NaN	FALSE
-11	Momentum	0.000000	None
-10	Learning Rate (Final)	0.204246	None
-9	Learning Rate (Initial)	0.050000	None
-8	Number of Iterations	227.000000	CONVERGED
-7	Alpha	0.150000	Elasticnet
-6	Regularization	0.020000	ENABLED
-5	BIC	5.572674	None
-4	AIC	-0.799777	None
-3	Number of Observations	15.000000	None
-2	MSE	0.285556	None
-1	Loss Function	NaN	EPSILON_INSENSITIVE
0	(Intercept)	2.191718	None
1	MedInc	1.226729	None
2	HouseAge	.0232213	None
3	AveRooms	-.332659	None
4	AveBedrms	0.000000	None
5	Population	0.113633	None
6	AveOccup	-0.286434	None
7	Latitude	-0.261509	None

attribute	predictor	estimate	value
8	Longitude	-0.198519	None

### Example: TD\_SVM Classification iris Data Set

id	sepal_length	sepal_width	petal_length	petal_width	variety
61	5.0	2.0	3.5	1.0	Versicolor
51	7.0	3.2	4.7	1.4	Versicolor
50	5.1	3.4	1.5	0.2	Setosa
59	6.6	2.9	4.6	1.3	Versicolor
38	4.9	3.6	1.4	0.1	Setosa
...	...	...	...	...	...
90	5.5	2.5	4.0	1.3	Versicolor
67	5.6	3.0	4.5	1.5	Versicolor
44	5.0	3.5	1.6	0.6	Setosa
42	4.5	2.3	1.3	0.3	Setosa
82	5.5	2.4	3.7	1.0	Versicolor

### TD\_SVM Call Using Classification Model

```
CREATE VOLATILE TABLE svm_model_iris_data AS (
SELECT * FROM TD_SVM (
  ON iris_data
  USING
    InputColumns('[1:4]')
    ResponseColumn('label')
    ModelType('Classification')
    BatchSize(10)
    MaxIterNum(300)
) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;
```

### Output from TD\_SVM

attribute	predictor	estimate	value
-17	OneClass SVM	NaN	FALSE

attribute	predictor	estimate	value
-16	Kernel	NaN	LINEAR
-15	Intercept Scaling	1.000000	None
-13	LocalSGD Iterations	0.000000	None
-12	Nesterov	NaN	FALSE
-11	Momentum	0.000000	None
-10	Learning Rate (Final)	0.677479	None
-9	Learning Rate (Initial)	0.050000	None
-8	Number of Iterations	56.000000	CONVERGED
-7	Alpha	0.150000	Elasticnet
-6	Regularization	0.020000	ENABLED
-5	BIC	20.794415	None
-4	AIC	10.000000	None
-3	Number of Observations	64.000000	None
-2	Loglik	-0.000000	None
-1	Loss Function	NaN	HINGE
0	(Intercept)	0.43063	None
1	sepal_length	0.163885	None
2	sepal_width	0.907101	None
3	petal_length	-1.411058	None
4	petal_width	-0.527589	None

## TD\_VectorDistance

TD\_VectorDistance computes similarity or dissimilarity between two vectors in multi-dimensional space. The distance between vectors is usually calculated using a distance metric, such as Euclidean, Manhattan, or Cosine. It takes a table of target vectors, and a table of reference vectors and returns a table that contains the distance between target-reference pairs. The function computes the distance between the target pair and the reference pair from the same table if you provide only one table as the input.

**Note:**

The algorithm used in this function is of the order of  $N^2$  (where N is the number of rows). The query runs significantly longer as the number of rows increases in either the target table or the reference table. Because the reference table is a DIMENSION input, it is copied to the spool for each AMP before running the query. The user spool limits the size and scalability of the input.

Euclidean distance is the common distance metric used in machine learning algorithms, and it measures the straight-line distance between two points in a multidimensional space. It is calculated by taking the square root of the sum of the squared differences between each corresponding dimension of the two vectors.

Manhattan distance measures the distance between two points by summing the absolute differences between their corresponding dimensions. It is useful when the dimensions have different scales or units.

Cosine similarity measures the angle between two vectors and is commonly used in natural language processing and information retrieval. It measures the cosine of the angle between two vectors, and its value ranges from -1 to 1, with 1 indicating that the vectors are identical and -1 indicating that they are completely dissimilar.

## Function Information

- [TD\\_VectorDistance Syntax](#)
- [Required Syntax Elements for TD\\_VectorDistance](#)
- [Optional Syntax Elements for TD\\_VectorDistance](#)
- [TD\\_VectorDistance Input](#)
- [TD\\_VectorDistance Output](#)
- [Example: TD\\_VectorDistance Using Euclidean, Manhattan, and Cosine](#)

## TD\_VectorDistance Syntax

```
TD_VectorDistance (
  ON { table | view | (query) } AS TargetTable PARTITION BY ANY
  [ ON { table | view | (query) } AS ReferenceTable DIMENSION ]
  USING
    TargetIDColumn ('target_id_column')
    TargetFeatureColumns ({ 'target_feature_column' | target_feature_Column_range }[,...])
    [ RefIDColumn ('ref_id_column') ]
    [ RefFeatureColumns ({ 'ref_feature_column' | ref_feature_Column_range }[,...]) ]
    [ DistanceMeasure ({ 'Cosine' | 'Euclidean' | 'Manhattan'}[,...]) ]
    [ TopK(integer_value) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_VectorDistance

**ON clause**

Accepts the TargetTable clause.

**TargetIDColumn**

Specifies the target table column name that contains identifiers of the target table vectors.

**TargetFeatureColumns**

Specifies the target table column names that contain features of the target table vectors.

**Note:**

You can specify up to 2018 feature columns.

The order of the features columns provided in TargetFeatureColumns argument must be the same as the order as those provided in RefFeatureColumns.

Both TargetFeatureColumns and RefFeatureColumns data types must be the same. Otherwise, an error is reported.

## Optional Syntax Elements for TD\_VectorDistance

**ON clause**

Accepts the ReferenceTable clause.

**RefIDColumn**

Reference table column name that contains identifiers of the reference table vectors.

**RefFeatureColumns**

List of reference table column names that contain features of the reference table vectors.

**Note:**

You can specify up to 2018 feature columns.

**DistanceMeasure**

Specify the distance type to compute between the target and the reference vector:

- Euclidean: Euclidean distance between the target vector and the reference vector.
- Manhattan: Manhattan distance between the target vector and the reference vector.
- Cosine: Cosine distance between the target vector and the reference vector.

**TopK**

Maximum number of closest reference vectors to include in the output table for each target vector. The value k is an integer between 1 and 1024.

Default value: 10

**Important:**

The function returns  $N^2$  output if you use the TopK value as -1 because the function includes all reference vectors in the output table.

**TD\_VectorDistance Input****Target Table Schema**

Column	Data Type	Description
target_id_column	BYTEINT, SMALLINT, BIGINT, INTEGER	Target table column name that contains target table vector identifiers.
target_feature_column	BYTEINT, SMALLINT, BIGINT, INTEGER, DECIMAL, NUMBER, FLOAT, REAL, DOUBLE PRECISION	Target table column names that contain features of the target table vectors.

**Reference Table Schema**

Column	Data Type	Description
ref_id_column	BYTEINT, SMALLINT, BIGINT, INTEGER	The reference table column name that contains identifiers of the reference table vectors.
ref_feature_column	BYTEINT, SMALLINT, BIGINT, INTEGER, DECIMAL,	The reference table column names that contain features of the reference table vectors.

Column	Data Type	Description
	NUMBER, FLOAT, REAL, DOUBLE PRECISION	

## TD\_VectorDistance Output

The function produces a table with the distances between the target and reference vectors.

Column	Data Type	Description
Target_ID	BIGINT	Specified target table column name.
Reference_ID	BIGINT	Specified reference vector.
DistanceType	VARCHAR	Distance measure type. Possible values are { cosine, euclidean, manhattan}.
Distance	DOUBLE PRECISION	The distance between the target and the reference vectors.

## TD\_VectorDistance Usage Notes

- This function requires the UTF8 character set.
- This function does not support the following:
  - Pass-through characters
  - KanjiSJIS data type
  - Graphic data types
- The order of the feature columns in the TargetFeatureColumns and RefFeatureColumns arguments must be the same.
- The function will skip the feature value if the value is: NULL, NAN, or INF.
- The target table can have the following partitioning:
  - No partition
  - PARTITION BY ANY ORDER BY
  - PARTITION BY ANY
- The TargetFeatureColumns and RefFeatureColumns must have the same data type.

## Example: TD\_VectorDistance Using Euclidean, Manhattan, and Cosine

### Target Table

Userid	CallDuration	DataCounter	SMS
1	0.0000333	0.2	0.1
2	0.5	0.4	0.4
3	1	0.8	0.9
4	0.01	0.4	0.2

### Reference Table

Userid	CallDuration	DataCounter	SMS
5	0.93	0.4	0.7
6	0.83	0.3	0.6
7	0.73	0.5	0.7

### TD\_VectorDistance SQL Call

```

SELECT target_id, reference_id, distancetype, cast(distance as decimal(36,8))
as distance FROM TD_VECTORDISTANCE (
ON target_mobile_data_dense AS TargetTable
ON ref_mobile_data_dense AS ReferenceTable DIMENSION
USING
TargetIDColumn('userid')
TargetFeatureColumns('CallDuration', 'DataCounter', 'SMS')
RefIDColumn('userid')
RefFeatureColumns('CallDuration', 'DataCounter', 'SMS')
DistanceMeasure('euclidean', 'cosine', 'manhattan')
topk(2)
) AS dt order by 3,1,2,4;

```

### TD\_VectorDistance Output

Target_ID	Reference_ID	DistanceType	Distance
1	5	cosine	0.45486518

1	7	cosine	0.32604815
2	5	cosine	0.02608923
2	7	cosine	0.00797609
3	5	cosine	0.02415054
3	7	cosine	0.00337338
4	5	cosine	0.43822243
4	7	cosine	0.31184844
1	6	euclidean	0.97408661
1	7	euclidean	0.99138861
2	6	euclidean	0.39862262
2	7	euclidean	0.39102429
3	5	euclidean	0.45265881
3	7	euclidean	0.45044423
4	6	euclidean	0.91782351
4	7	euclidean	0.88226980
1	6	manhattan	1.42996670
1	7	manhattan	1.62996670
2	6	manhattan	0.63000000
2	7	manhattan	0.63000000
3	5	manhattan	0.67000000
3	7	manhattan	0.77000000
4	6	manhattan	1.32000000
4	7	manhattan	1.32000000

## TD\_XGBoost

TD\_XGBoost function, also known as *eXtreme Gradient Boosting*, is an implementation of the gradient boosted decision tree designed for speed and performance. It has recently been dominating applied machine learning.

In gradient boosting, each iteration fits a model to the residuals (errors) of the previous iteration to correct the errors made by existing models. The predicted residual is multiplied by this learning rate and then added to the previous prediction. Models are added sequentially until no further improvements can be made. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

Gradient boosting involves three elements:

- A loss function to be optimized.
- A weak learner to make predictions.
- An additive model to add weak learners to minimize the loss function.

The loss function used depends on the type of problem being solved. For example, regression may use a squared error and binary classification may use binomial. A benefit of the gradient boosting is that a new boosting algorithm does not have to be derived for each loss function. Instead, it provides a generic enough framework that any differentiable loss function can be used. TD\_XGBoost function supports

both regression and classification predictive modeling problems. The model that it creates is used in the [TD\\_XGBoostPredict](#) function for making predictions.

- Regression: The prediction is based on continuous values. XGBoost regression calculates the difference between the current prediction and the known correct target value. This difference is called residual. After that, XGBoost regression trains a weak model that maps features to that residual. This residual predicted by a weak model is added to the existing model input, and thus this process nudges the model towards the correct target. Repeating this step improves the overall model prediction.
- Classification: Similar to regression, XGBoost uses regression trees for classification. In this case, the residual is computed by converting odds (a ratio between the number of events and non-events) to probability and the probability is expressed through log odds to obtain residuals. For example, if your data contains three spam emails and two non-spam emails, the odds are 3:2, that is, 1.5 in decimal notation.

TD\_XGBoost function supports the following features.

- Regression
- Multiple class and binary classification

## Usage

- When a dataset is small, best practice is to distribute the data to one AMP. To do this, create an identifier column as a primary index, and use the same value for each row.

This can also be achieved using DataRedistributionColumn and MinRowsPerAmp.

- For Classification (softmax), a maximum of 500 classes are supported.
- For Classification, while using a SELECT statement for the function input, the SELECT statement must have a deterministic output. Otherwise, the function may not run successfully or return the correct output. For example, the function must not have an ON clause such as "SELECT top 500 \* from table\_t".
- The processing time is controlled by (proportional to):
  - The number of boosted trees (controlled by NumBoostedTrees, TreeSize, and CoverageFactor).
  - The number of iterations (sub-trees) in each boosted tree (controlled by IterNum).
  - The complexity of an iteration (controlled by MaxDepth, MinNodeSize, ColumnSampling, MinImpurity).

A careful choice of these parameters can be used to control the processing time. For example, changing CoverageFactor from 1.0 to 2.0 doubles the number of boosted trees, which as a result, doubles the execution time roughly.

- Use DataRedistributionColumn, in combination with MinRowsPerAmp, to distribute data to any number of AMPs using a specified column. This can be helpful under the following scenarios:
  - The dataset is small and needs to be distributed to fewer AMPs.
  - The dataset is imbalanced and needs to be distributed based on a column uncorrelated with class labels so that all classes are present on each AMP for modeling.
  - Data distribution needs to be changed to eliminate skewness and distribute data evenly among the AMPs.

- TD\_XGBoost input table supports up to 2047 columns when DataRedistributionColumn is used.

Teradata recommends redistributing the input table rows to one or fewer AMPs in the following cases for improved training results:

- Classification with imbalanced datasets: When dealing with classification tasks and encountering an imbalanced dataset, such as having 1% rows for minority class 1 and 99% rows labeled as majority class 0, it is advisable to reduce the number of AMPs that hold the input table rows. This redistribution increases the likelihood of each AMP containing minority class data rows, allowing for better training of sub-tree models on the minority class.
- Large cluster with smaller datasets: In situations where the cluster consists of a large number of AMPs (for example, 200 AMPs) and the dataset size (total number of input table rows) is relatively small (for example, 1000 or fewer rows), redistributing the data rows to fewer AMPs can improve the quality of the model. This redistribution to fewer AMPs enables better training of sub-tree models because each model trains on a more substantial training sample.

## Function Information

- [TD\\_XGBoost Syntax](#)
- [Required Syntax Elements for TD\\_XGBoost](#)
- [Optional Syntax Elements for TD\\_XGBoost](#)
- [TD\\_XGBoost Input](#)
- [TD\\_XGBoost Output](#)
- [Examples: How to Use TD\\_XGBoost](#)

## TD\_XGBoost Syntax

```
TD_XGBoost (
    ON { table | view | (query) } AS INPUTTABLE PARTITION BY ANY
    [ OUT [ PERMANENT | VOLATILE ] TABLE MetaInformationTable(output_table_name) ]
    USING
        InputColumns ({'input_column' | 'input_column_range'}[, ...])
        ResponseColumn('response_column')
        [ ModelType ('classification' | 'regression') ]
        [ MaxDepth (maxdepth) ]
        [ MinNodeSize (minnodesize) ]
        [ NumParallelTrees (numparalleltrees) ]
        [ RegularizationLambda (lambda) ]
        [ LearningRate (learningrate) ]
        [ ColumnSampling (sample_fraction) ]
        [ CoverageFactor (coveragefactor) ]
        [ NumBoostRounds (numboostrounds) ]
        [ Seed (seed) ]
        [ BaseScore (basescore) ]
```

```
[ MinImpurity (minimpurity) ]
[ TreeSize (treesize) ]
[ DataRedistributionColumn (data_redistribution_column) ]
[ MinRowsPerAmp (min_rows_per_amp) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_XGBoost

### ON clause

Specifies intput table used to train XGBoost model.

The InputTable in TD\_XGBoost query can have no partition at all, or have PARTITION BY ANY clause.

### InputColumns

Specifies the input table columns name that need to be used for training the model (predictors, features, or independent variables).

**Note:**

Input column names with double quotation marks are not allowed for this function.

### ResponseColumn

Specifies the column name that contains the class label for classification or target value (dependent variable) for regression.

## Optional Syntax Elements for TD\_XGBoost

### OUT clause

Specifies the name of the output table that records training accuracy over iterations.

**ModelType**

Specifies whether the analysis is a regression (continuous response variable) or a multiple-class classification (predicting result from the number of classes). Only Regression and Classification are accepted values.

Default: Regression.

**MaxDepth**

Specifies a decision tree stopping criterion. If the tree reaches a depth past this value, the algorithm could stops looking for splits. Decision trees can grow to  $(2(\max\_depth+1)-1)$  nodes. This stopping criterion has the greatest effect on the performance of the function. The maximum value is 2147483647.

Default: 5

**MinNodeSize**

Specifies a decision tree stopping criterion; the minimum size of any node within each decision tree.

Default: 1

**NumParallelTrees**

Specifies the parallel boosted trees number. The num\_trees is an INTEGER value in the range [1, 10000]. Each boosted tree operates on a sample of data that fits in an AMP memory. By default, NumBoostedTrees is chosen equal to the number of AMPS with data.

If NumBoostedTrees is greater than the number of AMPS with data, each boosting operates on a sample of the input data, and the function estimates sample size (number of rows) using this formula:  $\text{sample\_size} = \text{total\_number\_of\_input\_rows} / \text{number\_of\_trees}$

The sample\_size must fit in an AMP memory. It always uses the sample size (or tree size) that fits in an AMP memory to build tree models and ignores those rows cannot fit in memory.

A higher NumBoostedTrees value may improve function run time but may decrease prediction accuracy.

**Note:**

You can still use the previous argument name NumBoostedTrees.

Default: -1

**RegularizationLambda**

Specifies the L2 regularization that the loss function uses while boosting trees. The lambda is a DOUBLE PRECISION value in the range [0, 100000]. The higher the lambda, the stronger the regularization effect. The value 0 specifies no regularization.

Default: 1

### **LearningRate**

Specifies the learning rate (weight) of a learned tree in each boosting step. After each boosting step, the algorithm multiplies the learner by shrinkage to make the boosting process more conservative. The shrinkage is a DOUBLE PRECISION value in the range (0, 1]. The value 1 specifies no shrinkage.

---

#### **Note:**

You can still use the previous argument name ShrinkageFactor.

---

Default: 0.5

### **ColumnSampling**

Specifies the features fraction to sample during boosting. The sample\_fractions is a DOUBLE PRECISION value in the range (0, 1].

Default: 1.0

### **CoverageFactor**

Specifies the coverage level for the dataset while boosting trees (in percentage, for example, 1.25 = 125% coverage). You can only use CoverageFactor if you do not supply NumBoostedTrees. When NumBoostedTrees is specified, coverage depends on the value of NumBoostedTrees. If NumBoostedTrees is not specified, NumBoostedTrees is chosen to achieve this level of coverage.

Default: 1.0

### **NumBoostRounds**

Specifies the iterations (rounds) number to boost the weak classifiers. The iterations must be an INTEGER in the range [1, 100000].

---

#### **Note:**

You can still use the previous argument name IterNum.

---

Default: 10

### **Seed**

Specifies an integer value to use in determining the random seed for column sampling.

Default: 1

**BaseScore**

Specifies the initial prediction value for all data points. Typically that value would be set to the mean of the observed value in the training set. This information is shown in the meta row in the model table. For classification, basescore value must be in the range (0, 1) and the default value is 0.5. The regression case accepts any double values in the range [-1e50, 1e50] and the default value is 0.

Default: 0

**MinImpurity**

Specifies the minimum impurity at which the tree stops splitting further down. For regression, a criteria of squared error is used whereas for classification, gini impurity is used.

Default: 0.0

**TreeSize**

Specifies the rows number that each tree uses as its input data set. The function builds a tree using either the number of rows on an AMP, the number of rows that fit into the AMP memory (whichever is less), or the number of rows given by the TreeSize argument. By default, this value is computed as the minimum of the number of rows on an AMP, and the number of rows that fit into the AMP memory.

**Note:**

By using argument TreeSize and reduce the value used in argument NumParallelTrees, most of the exceptions caused by out-of-memory (OOM) can be solved. For example, this is a typical exception caused by OOM.

Default: -1

**DataRedistributionColumn**

Specifies the name of the column used to redistribute the data. The maximum value is 128.

- If the number of unique values in this column is less than the result of "total number of input rows / MinRowsPerAmp argument", then the rows in the input table will be distributed to the AMPS equivalent to the number of unique values in this column.
- If the number of unique values in this column is greater than the result of "total number of input rows / MinRowsPerAmp", then the rows in the input table will be distributed to the AMPS equivalent to the "total number of input rows / MinRowsPerAmp".

**MinRowsPerAmp**

Specifies the minimum number of rows (input table rows) an AMP should have when data needs to be redistributed using DataRedistributionColumn.

Default: 100

## TD\_XGBoost Input

### InputTable Schema

Column Name	Data Type	Description
input_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	Columns that function uses to train a XGBoost model.
response_column (regression)	FLOAT, DECIMAL, NUMBER, INTEGER, BIGINT, SMALLINT, BYTEINT	Column that contains the response value for an observation.
response_column (classification)	INTEGER, BIGINT, SMALLINT, BYTEINT	Column that contains the response value for an observation.
DataRedistributionColumn	INTEGER, BIGINT, SMALLINT, BYTEINT	Column that contains the unique values used to distribute the data in the input table among the available AMPs.

## TD\_XGBoost Output

### Output Table Schema

Column	Data Type	Description
task_index	SMALLINT	Identifier of AMP that produced a boosting tree.
tree_num	SMALLINT	Identifier of boosted tree. Number of unique tree_id values depends on NumBoostedTrees syntax element value and number of AMPs.
iter	SMALLINT	Iteration (boosting round) number.
class_num	SMALLINT	Index of class column to predict. It only appears in classification. For LossFunction ('softmax'), the default: Number of unique class_num values is number of class labels in data set. For K class labels: class_num values are the integers in range [0, K-1]. For LossFunction('binomial'): There is only one class_num value.
tree_order	SMALLINT	Identifier of a complete JSON order of the regression_tree/ classification_tree column.
regression_tree / classification_tree	VARCHAR	JSON representation of decision tree. For JSON types that can appear in the representation, see the following table.  <b>Note:</b> The maximum length is 32000.

**Out Table [Records Training Accuracy over Iterations] Schema**

Column Name	Data Type	Description
task_index	SMALLINT	Identifier of AMP that produced a boosting tree.
iter	SMALLINT	Iteration (boosting round) number.
tree_num	SMALLINT	Identifier of boosted tree. Number of unique tree_id values depends on NumBoostedTrees syntax element value and number of AMPs.
iter	SMALLINT	Iteration (boosting round) number.
mse (regression)/accuracy (classification)	Double	Regression MSE represents the difference between the original and predicted values extracted by averaged the absolute difference over the data set. Classification accuracy is a metric that summarizes the performance of a classification model as the number of correct predictions divided by the total number of predictions.
average residuals (regression)/deviance (classification)	Double	Regression average residuals is the average difference between the total observed values of the dependent variables (y) and the predicted values y hat. Classification deviance measures the difference in "fit" of a candidate model and that of the saturated model.

**JSON Types in JSON Representation of Boosting Tree**

JSON Type	Description
id_	Node identifier.
sum_	Appears only for regression trees. Sum of values of response variable at node identified by id.
sumSq_	Appears only for regression trees. Sum of squared values of response variable at node identified by id.
responseCounts_	Appears only for classification trees. The number of observations in each class at a node, identified by id.
size_	The total number of observations at a node, identified by id.
maxDepth_	Maximum possible depth of the tree, starting from node identified by id. For root node, the value is max_depth; for leaf nodes, 0; for other nodes, maximum possible depth of the tree, starting from that node.
split_	Start of JSON item describing a split at node identified by id.
splitValue_	The attribute value used for splitting a tree node.
score_	Gini score of the node identified by id.
attr_	Attribute (predictor) on which algorithm split at node identified by id.
type_	Type of tree and split. Possible values:

JSON Type	Description
	<ul style="list-style-type: none"> <li>REGRESSION_NUMERIC_SPLIT</li> </ul>
leftNodeSize_	The number of observations assigned to the left node in the split.
rightNodeSize_	The number of observations assigned to the right node in the split.
leftChild_	Start of JSON item describing left child of a node, identified by id.
rightChild_	Start of JSON item describing right child of a node, identified by id.
nodeType_	Type of a node identified by id. Possible values: <ul style="list-style-type: none"> <li>REGRESSION_NODE</li> <li>REGRESSION_LEAF</li> </ul>
prediction_	The value of region prediction of a leaf node.

## Examples: How to Use TD\_XGBoost

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [TD\\_XGBoost for Classification](#)
- [TD\\_XGBoost for Regression](#)
- [TD\\_XGBoost for JSON Tree Model String](#)
- [TD\\_XGBoost using TD\\_DataPartitioner](#)

## TD\_XGBoost for Classification

This section shows the input table, SQL query, and output tables of an example using TD\_XGBoost for classification.

### InputTable

The input is a diabetes dataset sample, with 3 feature columns and a target (response) column 'outcome'. It is a binary classification problem with two classes 0 and 1.

ID	outcome	col_1	col_2	col_3
1	1	23	8.14	0.84054
1	1	20	3.97	0.66351
1	1	0	4.05	0.07022
1	1	15	10.59	0.21719

ID	outcome	col_1	col_2	col_3
1	1	0	6.91	0.18836
1	2	20	3.97	0.66351
1	2	20	3.97	0.5405
1	2	16	1.95	2.77974
1	2	0	4.49	0.05735

## SQL Call

```
SELECT * FROM TD_XGBoost (
ON diabetes_sample PARTITION BY ANY
OUT TABLE MetaInformationTable(xgb_out)
USING
ResponseColumn('response')
InputColumns('[2:4]')
MaxDepth(3)
MinNodeSize(1)
NumParallelTrees(2)
ModelType('CLASSIFICATION')
Seed(1)
RegularizationLambda(1)
LearningRate(0.5)
NumBoostRounds(2)
MinImpurity(0)
ColumnSampling(1.0)
) AS dt;
```

## Output

```
task_index tree_num iter class_num tree_order tree
----- -----
0          1      1    0        0
{"id_":1,"sum_":-2.000000,"sumSq_":1.000000,"size_":4,"maxDepth_":0,"value_":-0.500000,"nodeType_":"REGRESSION_LEAF","prediction_-":-0.500000}
0          1      2    0        0
{"id_":1,"sum_":-1.510163,"sumSq_":0.570148,"size_":4,"maxDepth_":0,"value_":-0.377541,"nodeType_":"REGRESSION_LEAF","prediction_-":-0.389214}
0          2      1    0        0
{"id_":1,"sum_":1.000000,"sumSq_":1.000000,"size_":4,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
```

```
{"splitValue_":8.000000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.750000,"scoreImprove_":0.750000,"leftNodeSize_":1,"rightNodeSize_":1,"maxDepth_":0,"value_":-0.500000,"nodeType_":"REGRESSION_LEAF","prediction_":-0.200000}, "rightNodeSize_":1,"maxDepth_":0,"value_":0.500000,"nodeType_":"REGRESSION_LEAF","prediction_":0.428571}}
0          2          2          0          0
{"id_":1,"sum_":0.733237,"sumSq_":0.669463,"size_":4,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":8.000000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.535054,"scoreImprove_":0.535054,"leftNodeSize_":1,"rightNodeSize_":1,"maxDepth_":0,"value_":-0.450166,"nodeType_":"REGRESSION_LEAF","prediction_":-0.180425}, "rightNodeSize_":1,"maxDepth_":0,"value_":0.394468,"nodeType_":"REGRESSION_LEAF","prediction_":0.344696}}
0          -1          -1          -1          -1
{"lossType":"LOGISTIC","numBoostedTrees":2,"iterNum":2,"avgResponses":0.000000,"classMapping": {"1":0,"2":1}}
```

## Out MetaInformation Table

task_index	tree_num	iter	accuracy	deviance
0	1	1	1	0.378
0	1	2	1	0.291
0	2	1	1	0.408
0	2	2	1	0.338

## TD\_XGBoost for Regression

This section shows the input table, SQL query, and output tables of an example using TD\_XGBoost for regression.

### InputTable

The input is a Boston housing dataset sample, with 3 feature columns and a target (response) column 'medv'.

ID	medv	col_1	col_2	col_3
1	13.9	23	8.14	0.84054
1	36	20	3.97	0.66351

ID	medv	col_1	col_2	col_3
1	23.2	0	4.05	0.07022
1	22.4	15	10.59	0.21719
1	20	0	6.91	0.18836
1	36	20	3.97	0.66351
1	43.5	20	3.97	0.5405
1	11.8	16	1.95	2.77974
1	26.6	0	4.49	0.05735

## SQL Call

```
SELECT * FROM TD_XGBoost (
ON housing_sample partition by ANY
OUT TABLE MetaInformationTable(xgb_out)
USING
ResponseColumn('medv')
InputColumns('[2:4]')
MaxDepth(3)
MinNodeSize(1)
NumParallelTrees(1)
ModelType('REGRESSION')
Seed(1)
RegularizationLambda(1000)
LearningRate(0.8)
NumBoostRounds(3)
ColumnSampling(1.0)
) as dt;
```

## Output

```
task_index tree_num iter tree_order regression_tree
----- -----
0          1      1    0
{"id_":1,"sum_":233.400000,"sumSq_":6964.260000,"size_":9,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":4.010000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":710.645000,"scoreImprove_":710.645000,"leftNodeSize_":3,"rightNodeSize_":6},
"leftChi
{"id_":2,"sum_":115.500000,"sumSq_":4484.250000,"size_":3,"maxDepth_":2,"nodeTyp
```

```

e_": "REGRESSION_NODE", "split_":
{"splitValue_": 0.602005, "attr_": "col_3", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 37.500000, "scoreImprove_": 37.500000, "leftNodeSize_": 1, "rightNodeSize_": 2}, "leftChild_"
{"id_": 4, "sum_": 43.500000, "sumSq_": 1892.250000, "size_": 1, "maxDepth_": 0, "value_": 43.500000, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.034765}, "rightChild_"
 {"id_": 5, "sum_": 72.000000, "sumSq_": 2592.000000, "size_": 2, "maxDepth_": 0, "value_": 36.000000, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.057485}}, "rightChild_"
 {"id_": 3, "sum_": 117.900000, "sumSq_": 2480.010000, "size_": 6, "maxDepth_": 2, "nodeType_": "REGRESSION_NODE", "split_":
 {"splitValue_": 15.500000, "attr_": "col_1", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 138.720000, "scoreImprove_": 138.720000, "leftNodeSize_": 4, "rightNodeSize_": 2}, "leftCh
 {"id_": 6, "sum_": 92.200000, "sumSq_": 2147.560000, "size_": 4, "maxDepth_": 1, "nodeType_": "REGRESSION_NODE", "split_":
 {"splitValue_": 0.063785, "attr_": "col_3", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 16.803333, "scoreImprove_": 16.803333, "leftNodeSize_": 1, "rightNodeSize_": 3}, "leftChild_"
 {"id_": 12, "sum_": 26.600000, "sumSq_": 707.560000, "size_": 1, "maxDepth_": 0, "value_": 26.600000, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.021259}, "rightChild_"
 {"id_": 13, "sum_": 65.600000, "sumSq_": 1440.000000, "size_": 3, "maxDepth_": 0, "value_": 21.866667, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.052323}}, "rightChild_"
 {"id_": 7, "sum_": 25.700000, "sumSq_": 332.450000, "size_": 2, "maxDepth_": 1, "nodeType_": "REGRESSION_NODE", "split_":
 {"splitValue_": 19.500000, "attr_": "col_1", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 2.205000, "scoreImprove_": 2.205000, "leftNodeSize_": 1, "rightNodeSize_": 1}, "leftChild_"
 {"id_": 14, "sum_": 11.800000, "sumSq_": 139.240000, "size_": 1, "maxDepth_": 0, "value_": 11.800000, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.009431}, "rightChild_"
 {"id_": 15, "sum_": 13.900000, "sumSq_": 193.210000, "size_": 1, "maxDepth_": 0, "value_": 13.900000, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.011109}}}}
0      1      2      0
 {"id_": 1, "sum_": 233.051497, "sumSq_": 6944.447140, "size_": 9, "maxDepth_": 3, "nodeType_": "REGRESSION_NODE", "split_":
 {"splitValue_": 4.010000, "attr_": "col_2", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 709.380059, "scoreImprove_": 709.380059, "leftNodeSize_": 3, "rightNodeSize_": 6}, "leftChi
 {"id_": 2, "sum_": 115.350265, "sumSq_": 4472.955398, "size_": 3, "maxDepth_": 2, "nodeType_": "REGRESSION_NODE", "split_":
 {"splitValue_": 0.602005, "attr_": "col_3", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 37.727542, "scoreImprove_": 37.727542, "leftNodeSize_": 1, "rightNodeSize_": 2}, "leftChild_"
 {"id_": 4, "sum_": 43.465235, "sumSq_": 1889.226633, "size_": 1, "maxDepth_": 0, "value_": 43.465235, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.034737}, "rightChild_":

```

```

{"id_":5,"sum_":71.885030,"sumSq_":2583.728765,"size_":2,"maxDepth_":0,"value_":35.942515,"nodeType_":"REGRESSION_LEAF","prediction_":0.057393}],"rightChild_":{"id_":3,"sum_":117.701233,"sumSq_":2471.491742,"size_":6,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":15.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":137.788955,"scoreImprove_":137.788955,"leftNodeSize_":4,"rightNodeSize_":2},"leftChild_":{"id_":6,"sum_":92.021772,"sumSq_":2139.572918,"size_":4,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":0.063785,"attr_":"col_3","type_":"REGRESSION_NUMERIC_SPLIT","score_":17.024614,"scoreImprove_":17.024614,"leftNodeSize_":1,"rightNodeSize_":3},"leftChild_":{"id_":12,"sum_":26.578741,"sumSq_":706.429487,"size_":1,"maxDepth_":0,"value_":26.578741,"nodeType_":"REGRESSION_LEAF","prediction_":0.021242}],"rightChild_":{"id_":13,"sum_":65.443031,"sumSq_":1433.143431,"size_":3,"maxDepth_":0,"value_":21.814344,"nodeType_":"REGRESSION_LEAF","prediction_":0.052198}),"rightChild_":{"id_":7,"sum_":25.679461,"sumSq_":331.918824,"size_":2,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":19.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":2.201477,"scoreImprove_":2.201477,"leftNodeSize_":1,"rightNodeSize_":1}),"leftChild_":{"id_":14,"sum_":11.790569,"sumSq_":139.017527,"size_":1,"maxDepth_":0,"value_":11.790569,"nodeType_":"REGRESSION_LEAF","prediction_":0.009423}],"rightChild_":{"id_":15,"sum_":13.888891,"sumSq_":192.901296,"size_":1,"maxDepth_":0,"value_":13.888891,"nodeType_":"REGRESSION_LEAF","prediction_":0.011100}}}}}
0           1           3           0
{"id_":1,"sum_":232.703615,"sumSq_":6924.700934,"size_":9,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":4.010000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":708.116416,"scoreImprove_":708.116416,"leftNodeSize_":3,"rightNodeSize_":6}),"leftChild_":{"id_":2,"sum_":115.200741,"sumSq_":4461.692021,"size_":3,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":0.602005,"attr_":"col_3","type_":"REGRESSION_NUMERIC_SPLIT","score_":37.955128,"scoreImprove_":37.955128,"leftNodeSize_":1,"rightNodeSize_":2}),"leftChild_":{"id_":4,"sum_":43.430497,"sumSq_":1886.208097,"size_":1,"maxDepth_":0,"value_":43.430497,"nodeType_":"REGRESSION_LEAF","prediction_":0.034710}],"rightChild_":{"id_":5,"sum_":71.770243,"sumSq_":2575.483924,"size_":2,"maxDepth_":0,"value_":35.885122,"nodeType_":"REGRESSION_LEAF","prediction_":0.057302}),"rightChild_":{"id_":3,"sum_":117.502874,"sumSq_":2463.008913,"size_":6,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":15.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":136.863475,"scoreImprove_":136.863475,"leftNodeSize_":4,"rightNodeSize_":2}}

```

```

,"leftCh
{"id_":6,"sum_":91.843937,"sumSq_":2131.620417,"size_":4,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":0.063785,"attr_":"col_3","type_":"REGRESSION_NUMERIC_SPLIT","score":17.246563,"scoreImprove_":17.246563,"leftNodeSize_":1,"rightNodeSize_":3}, "leftChild_
{"id_":12,"sum_":26.557500,"sumSq_":705.300780,"size_":1,"maxDepth_":0,"value_":26.557500,"nodeType_":"REGRESSION_LEAF","prediction_":0.021225}, "rightChild_":
 {"id_":13,"sum_":65.286437,"sumSq_":1426.319637,"size_":3,"maxDepth_":0,"value_":21.762146,"nodeType_":"REGRESSION_LEAF","prediction_":0.052073}}, "rightChild_":
 {"id_":7,"sum_":25.658937,"sumSq_":331.388496,"size_":2,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":19.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score":2.197959,"scoreImprove_":2.197959,"leftNodeSize_":1,"rightNodeSize_":1}, "leftChild_"
 {"id_":14,"sum_":11.781146,"sumSq_":138.795410,"size_":1,"maxDepth_":0,"value_":11.781146,"nodeType_":"REGRESSION_LEAF","prediction_":0.009416}, "rightChild_":
 {"id_":15,"sum_":13.877791,"sumSq_":192.593086,"size_":1,"maxDepth_":0,"value_":13.877791,"nodeType_":"REGRESSION_LEAF","prediction_":0.011091}}}}
0      -1      -1      -1
{"lossType":"MSE","numBoostedTrees":1,"iterNum":3,"avgResponses":0.000000}

```

## Out MetaInformation Table

task_index	tree_num	iter	mse	average	residuals
0	1	1	233.0514974108345	771.6052377797125	
0	1	2	232.7036151951801	769.4112148342409	
0	1	3	232.35635211224303	767.2245697367563	

## TD\_XGBoost for JSON Tree Model String

This section shows the input table, SQL query, and output tables of an example using TD\_XGBoost for a JSON tree model string in multiple rows.

### InputTable

The input is a housing\_full dataset sample, with 13 feature columns and a target (response) column 'medv'.

---

#### Note:

The following table shows only a part of the dataset.

---

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	Istat	medv	partition_id
0.04379	80.0	3.37	0.0	0.398	5.787	31.1	6.6115	4.0	337.0	16.1	396.9	10.24	19.4	5
0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.09	1.0	296.0	15.3	396.9	4.98	24.0	31
0.05789	12.5	6.07	0.0	0.409	5.878	21.4	6.498	4.0	345.0	18.9	396.21	8.1	22.0	5
0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.9	9.14	21.6	31
0.13554	12.5	6.07	0.0	0.409	5.594	36.8	6.498	4.0	345.0	18.9	396.9	13.09	17.4	5
0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7	31
0.12816	12.5	6.07	0.0	0.409	5.885	33.0	6.498	4.0	345.0	18.9	396.9	8.79	20.9	5
0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4	31
0.08826	0.0	10.81	0.0	0.413	6.417	6.6	5.2873	4.0	305.0	19.2	383.73	6.72	24.2	5
0.02985	0.0	2.18	0.0	0.458	6.43	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7	31

## SQL Call

```
SELECT * FROM TD_XGBoost (
ON housing_full PARTITION BY ANY
OUT TABLE MetaInformationTable(xgb_out)
USING
    ResponseColumn('medv')
    InputColumns('[0:3]')
    MaxDepth(10)
    MinNodeSize(1)
    NumParallelTrees(2)
    ModelType('REGRESSION')
    Seed(1)
    RegularizationLambda(1000)
    LearningRate(0.8)
    NumBoostRounds(1)
    ColumnSampling(1.0)
) AS dt;
```

## Output

```
task_index tree_num iter tree_order regression_tree
----- -----
0         -1        -1     -1
{"lossType":"MSE","numBoostedTrees":2,"iterNum":1,"avgResponses":0.000000}
0           1         1     0
{"id_":1,"sum_":3560.900000,"sumSq_":96768.030000,"size_":143,"maxDepth_":10,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":3.985000,"attr_":"indus","type_":"REGRESSION_NUMERIC_SPLIT","score_":3039.486683,"scoreImprove_":3039.486683,"leftNodeSize_":41,"rightNodeSize_":102},
{"id_":2,"sum_":1319.100000,"sumSq_":45601.330000,"size_":41,"maxDepth_":9,"node
```

```

Type_:"REGRESSION_NODE","split_":
{"splitValue_":0.500000,"attr_":"chas","type_":"REGRESSION_NUMERIC_SPLIT","score_":526.668180,"scoreImprove_":526.668180,"leftNodeSize_":39,"rightNodeSize_":2},
"leftCh
 {"id_":4,"sum_":1223.100000,"sumSq_":40985.330000,"size_":39,"maxDepth_":8,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.067920,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":628.809286,"scoreImprove_":628.809286,"leftNodeSize_":22,"rightNodeSize_":17},
 "leftCh
 {"id_":8,"sum_":612.300000,"sumSq_":17947.470000,"size_":22,"maxDepth_":7,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.035810,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":293.582941,"scoreImprove_":293.582941,"leftNodeSize_":9,"rightNodeSize_":13},
 "leftChi
 {"id_":16,"sum_":290.000000,"sumSq_":9699.460000,"size_":9,"maxDepth_":6,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":3.150000,"attr_":"indus","type_":"REGRESSION_NUMERIC_SPLIT","score_":195.360556,"scoreImprove_":195.360556,"leftNodeSize_":8,"rightNodeSize_":1},
 "leftChi
 {"id_":32,"sum_":244.600000,"sumSq_":7638.300000,"size_":8,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.980000,"attr_":"indus","type_":"REGRESSION_NUMERIC_SPLIT","score_":39.446429,"scoreImprove_":39.446429,"leftNodeSize_":1,"rightNodeSize_":7},
 "leftChild_
 {"id_":64,"sum_":24.700000,"sumSq_":610.090000,"size_":1,"maxDepth_":0,"value_":24.700000,"nodeType_":"REGRESSION_LEAF","prediction_":0.019740}, "rightChild_":
 {"id_":65,"sum_":219.900000,"sumSq_":7028.210000,"size_":7,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.012035,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":52.116571,"scoreImprove_":52.116571,"leftNodeSize_":2,"rightNodeSize_":5},
 "leftChild_
 {"id_":130,"sum_":54.200000,"sumSq_":1520.840000,"size_":2,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.010010,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":52.020000,"scoreImprove_":52.020000,"leftNodeSize_":1,"rightNodeSize_":1},
 "leftChild_
 {"id_":260,"sum_":32.200000,"sumSq_":1036.840000,"size_":1,"maxDepth_":0,"value_":32.200000,"nodeType_":"REGRESSION_LEAF","prediction_":0.025734}, "rightChild_":
 {"id_":261,"sum_":22.000000,"sumSq_":484.000000,"size_":1,"maxDepth_":0,"value_":22.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.017582}}, "rightChild_":
 {"id_":131,"sum_":165.700000,"sumSq_":5507.370000,"size_":5,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.013715,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":6.384500,"scoreImprove_":6.384500,"leftNodeSize_":1,"rightNodeSize_":4},
 "left
 
```

```

Child_":
{"id_":262,"sum_":35.400000,"sumSq_":1253.160000,"size_":1,"maxDepth_":0,"value_":35.400000,"nodeType_":"REGRESSION_LEAF","prediction_":0.028292}, "rightChild_":
 {"id_":263,"sum_":130.300000,"sumSq_":4254.210000,"size_":4,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.030610,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":7.207500,"scoreImprove_":7.207500,"leftNodeSize_":3,"rightNodeSize_":1}, "leftChild_":
 {"id_":526,"sum_":95.400000,"sumSq_":3036.200000,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":46.250000,"attr_":"zn","type_":"REGRESSION_NUMERIC_SPLIT","score_":2.160000,"scoreImprove_":2.160000,"leftNodeSize_":1,"rightNodeSize_":2}, "leftChild_":
 {"id_":1052,"sum_":33.000000,"sumSq_":1089.000000,"size_":1,"maxDepth_":0,"value_":33.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.026374}, "rightChild_":
 {"id_":1053,"sum_":62.400000,"sumSq_":1947.200000,"size_":2,"maxDepth_":0,"value_":31.200000,"nodeType_":"REGRESSION_LEAF","prediction_":0.049820}}, "rightChild_":
 {"id_":527,"sum_":34.900000,"sumSq_":1218.010000,"size_":1,"maxDepth_":0,"value_":34.900000,"nodeType_":"REGRESSION_LEAF","prediction_":0.027892}}}}}, "rightChild_":
 {"id_":33,"sum_":45.400000,"sumSq_":2061.160000,"size_":1,"maxDepth_":0,"value_":45.400000,"nodeType_":"REGRESSION_LEAF","prediction_":0.036284}, "rightChild_":
 {"id_":17,"sum_":322.300000,"sumSq_":8248.010000,"size_":13,"maxDepth_":6,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":2.210000,"attr_":"indus","type_":"REGRESSION_NUMERIC_SPLIT","score_":127.956958,"scoreImprove_":127.956958,"leftNodeSize_":2,"rightNodeSize_":11}, "leftChild_":
 {"id_":34,"sum_":64.300000,"sumSq_":2098.450000,"size_":2,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.052235,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":31.205000,"scoreImprove_":31.205000,"leftNodeSize_":1,"rightNodeSize_":1}, "leftChild_":
 {"id_":68,"sum_":28.200000,"sumSq_":795.240000,"size_":1,"maxDepth_":0,"value_":28.200000,"nodeType_":"REGRESSION_LEAF","prediction_":0.022537}, "rightChild_":
 {"id_":69,"sum_":36.100000,"sumSq_":1303.210000,"size_":1,"maxDepth_":0,"value_":36.100000,"nodeType_":"REGRESSION_LEAF","prediction_":0.028851}}, "rightChild_":
 {"id_":35,"sum_":258.000000,"sumSq_":6149.560000,"size_":11,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.060630,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":16.300606,"scoreImprove_":16.300606,"leftNodeSize_":8,"rightNodeSize_":3}, "leftChild_":
 {"id_":70,"sum_":193.600000,"sumSq_":4760.060000,"size_":8,"maxDepth_":4,"nodeTy

```

```

pe_":"REGRESSION_NODE","split_":
{"splitValue_":0.049930,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":40.368000,"scoreImprove_":40.368000,"leftNodeSize_":5,"rightNodeSize_":3},"leftChild_"
{"id_":140,"sum_":112.300000,"sumSq_":2538.410000,"size_":5,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":2.805000,"attr_":"indus","type_":"REGRESSION_NUMERIC_SPLIT","score_":6.844500,"scoreImprove_":6.844500,"leftNodeSize_":1,"rightNodeSize_":4},"leftChild_":
{"id_":280,"sum_":24.800000,"sumSq_":615.040000,"size_":1,"maxDepth_":0,"value_":24.800000,"nodeType_":"REGRESSION_LEAF","prediction_":0.019820},"rightChild_":
 {"id_":281,"sum_":87.500000,"sumSq_":1923.370000,"size_":4,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.037580,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":3.520833,"scoreImprove_":3.520833,"leftNodeSize_":1,"rightNodeSize_":3},"leftChild_":
 {"id_":562,"sum_":23.500000,"sumSq_":552.250000,"size_":1,"maxDepth_":0,"value_":23.500000,"nodeType_":"REGRESSION_LEAF","prediction_":0.018781},"rightChild_":
 {"id_":563,"sum_":64.000000,"sumSq_":1371.120000,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":40.000000,"attr_":"zn","type_":"REGRESSION_NUMERIC_SPLIT","score_":5.606667,"scoreImprove_":5.606667,"leftNodeSize_":2,"rightNodeSize_":1},"leftChild_":
 {"id_":1126,"sum_":44.600000,"sumSq_":994.760000,"size_":2,"maxDepth_":0,"value_":22.300000,"nodeType_":"REGRESSION_LEAF","prediction_":0.035609},"rightChild_":
 {"id_":1127,"sum_":19.400000,"sumSq_":376.360000,"size_":1,"maxDepth_":0,"value_":19.400000,"nodeType_":"REGRESSION_LEAF","prediction_":0.015504}}}},"rightChild_":
 {"id_":141,"sum_":81.300000,"sumSq_":2221.650000,"size_":3,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.056105,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":18.375000,"scoreImprove_":18.375000,"leftNodeSize_":2,"rightNodeSize_":1},"leftChild_"
 {"id_":282,"sum_":57.700000,"sumSq_":1664.690000,"size_":2,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.054315,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.045000,"scoreImprove_":0.045000,"leftNodeSize_":1,"rightNodeSize_":1},"leftChild_":
 {"id_":564,"sum_":28.700000,"sumSq_":823.690000,"size_":1,"maxDepth_":0,"value_":28.700000,"nodeType_":"REGRESSION_LEAF","prediction_":0.022937},"rightChild_":
 {"id_":565,"sum_":29.000000,"sumSq_":841.000000,"size_":1,"maxDepth_":0,"value_":29.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.023177}}}, "rightChild_":
 {"id_":283,"sum_":23.600000,"sumSq_":556.960000,"size_":1,"maxDepth_":0,"value_":23.600000,"nodeType_":"REGRESSION_LEAF","prediction_":0.018861}}}},"rightChild_"

```

```
:
{"id_":71,"sum_":64.400000,"sumSq_":1389.500000,"size_":3,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":0.066705,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":1.926667,"scoreImprove_":1.926667,"leftNodeSize_":2,"rightNodeSize_":1},"leftChild_":
{"id_":142,"sum_":41.800000,"sumSq_":878.740000,"size_":2,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":0.065415,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":5.120000,"scoreImprove_":5.120000,"leftNodeSize_":1,"rightNodeSize_":1},"leftChild_":
{"id_":284,"sum_":22.500000,"sumSq_":506.250000,"size_":1,"maxDepth_":0,"value_":22.500000,"nodeType_":"REGRESSION_LEAF","prediction_":0.017982},"rightChild_":
 {"id_":285,"sum_":19.300000,"sumSq_":372.490000,"size_":1,"maxDepth_":0,"value_":19.300000,"nodeType_":"REGRESSION_LEAF","prediction_":0.015425}},"rightChild_":
 {"id_":143,"sum_":22.600000,"sumSq_":510.760000,"size_":1,"maxDepth_":0,"value_":22.600000,"nodeType_":"REGRESSION_LEAF","prediction_":0.018062}}}},"rightChild_":
 {"id_":9,"sum_":610.800000,"sumSq_":23037.860000,"size_":17,"maxDepth_":7,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.634095,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":185.457025,"scoreImprove_":185.457025,"leftNodeSize_":13,"rightNodeSize_":4}, "leftChi
 {"id_":18,"sum_":490.900000,"sumSq_":19355.010000,"size_":13,"maxDepth_":6,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.564205,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":354.025315,"scoreImprove_":354.025315,"leftNodeSize_":11,"rightNodeSize_":2}, "leftChi
 {"id_":36,"sum_":390.900000,"sumSq_":14355.010000,"size_":11,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.372250,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":85.523669,"scoreImprove_":85.523669,"leftNodeSize_":7,"rightNodeSize_":4}, "leftChild_
 {"id_":72,"sum_":234.000000,"sumSq_":8130.460000,"size_":7,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.098455,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":59.674286,"scoreImprove_":59.674286,"leftNodeSize_":3,"rightNodeSize_":4}, "leftChild_
 {"id_":144,"sum_":110.400000,"sumSq_":4136.240000,"size_":3,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.078450,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":73.500000,"scoreImprove_":73.500000,"leftNodeSize_":2,"rightNodeSize_":1}, "leftChild_
 {"id_":288,"sum_":66.600000,"sumSq_":2217.800000,"size_":2,"maxDepth_":2,"nodeTy
```

```
pe_":"REGRESSION_NODE","split_":
{"splitValue_":2.535000,"attr_":"indus","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.020000,"scoreImprove_":0.020000,"leftNodeSize_":1,"rightNodeSize_":1},"leftChild_":
{"id_":576,"sum_":33.400000,"sumSq_":1115.560000,"size_":1,"maxDepth_":0,"value_":33.400000,"nodeType_":"REGRESSION_LEAF","prediction_":0.026693}, "rightChild_":
{"id_":577,"sum_":33.200000,"sumSq_":1102.240000,"size_":1,"maxDepth_":0,"value_":33.200000,"nodeType_":"REGRESSION_LEAF","prediction_":0.026533}}, "rightChild_":
{"id_":289,"sum_":43.800000,"sumSq_":1918.440000,"size_":1,"maxDepth_":0,"value_":43.800000,"nodeType_":"REGRESSION_LEAF","prediction_":0.035005}}, "rightChild_":
{"id_":145,"sum_":123.600000,"sumSq_":3994.220000,"size_":4,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":0.117935,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":120.333333,"scoreImprove_":120.333333,"leftNodeSize_":1,"rightNodeSize_":3}, "leftChild":
 {"id_":290,"sum_":21.400000,"sumSq_":457.960000,"size_":1,"maxDepth_":0,"value_":21.400000,"nodeType_":"REGRESSION_LEAF","prediction_":0.017103}, "rightChild_":
 {"id_":291,"sum_":102.200000,"sumSq_":3536.260000,"size_":3,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.121435,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":32.201667,"scoreImprove_":32.201667,"leftNodeSize_":1,"rightNodeSize_":2}, "leftChild_":
 {"id_":582,"sum_":38.700000,"sumSq_":1497.690000,"size_":1,"maxDepth_":0,"value_":38.700000,"nodeType_":"REGRESSION_LEAF","prediction_":0.030929}, "rightChild_":
 {"id_":583,"sum_":63.500000,"sumSq_":2038.570000,"size_":2,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.166210,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":22.445000,"scoreImprove_":22.445000,"leftNodeSize_":1,"rightNodeSize_":1}, "leftChild_":
 {"id_":1166,"sum_":28.400000,"sumSq_":806.560000,"size_":1,"maxDepth_":0,"value_":28.400000,"nodeType_":"REGRESSION_LEAF","prediction_":0.022697}, "rightChild_":
 {"id_":1167,"sum_":35.100000,"sumSq_":1232.010000,"size_":1,"maxDepth_":0,"value_":35.100000,"nodeType_":"REGRESSION_LEAF","prediction_":0.028052}}}}, "rightChild_":
 {"id_":73,"sum_":156.900000,"sumSq_":6224.550000,"size_":4,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.537115,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":20.020833,"scoreImprove_":20.020833,"leftNodeSize_":1,"rightNodeSize_":3}, "leftChild_":
 {"id_":146,"sum_":43.100000,"sumSq_":1857.610000,"size_":1,"maxDepth_":0,"value_":43.100000,"nodeType_":"REGRESSION_LEAF","prediction_":0.034446}, "rightChild_":
 {"id_":147,"sum_":113.800000,"sumSq_":4366.940000,"size_":3,"maxDepth_":3,"nodeType_":
```

```

type_:"REGRESSION_NODE","split_":
{"splitValue_":0.540305,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":25.626667,"scoreImprove_":25.626667,"leftNodeSize_":1,"rightNodeSize_":2},"leftChild_"
{"id_":294,"sum_":33.800000,"sumSq_":1142.440000,"size_":1,"maxDepth_":0,"value_":33.800000,"nodeType_":"REGRESSION_LEAF","prediction_":0.027013},"rightChild_":
{"id_":295,"sum_":80.000000,"sumSq_":3224.500000,"size_":2,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":0.545285,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":24.500000,"scoreImprove_":24.500000,"leftNodeSize_":1,"rightNodeSize_":1},"leftChild_"
 {"id_":590,"sum_":43.500000,"sumSq_":1892.250000,"size_":1,"maxDepth_":0,"value_":43.500000,"nodeType_":"REGRESSION_LEAF","prediction_":0.034765},"rightChild_":
 {"id_":591,"sum_":36.500000,"sumSq_":1332.250000,"size_":1,"maxDepth_":0,"value_":36.500000,"nodeType_":"REGRESSION_LEAF","prediction_":0.029171}}}}, "rightChild_":
 {"id_":37,"sum_":100.000000,"sumSq_":5000.000000,"size_":2,"maxDepth_":0,"value_":50.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.079840}),"rightChild_":
 {"id_":19,"sum_":119.900000,"sumSq_":3682.850000,"size_":4,"maxDepth_":6,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.712565,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":37.822500,"scoreImprove_":37.822500,"leftNodeSize_":2,"rightNodeSize_":2},"leftChild_"
 {"id_":38,"sum_":66.100000,"sumSq_":2202.010000,"size_":2,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":0.660080,"attr_":"crim","type_":"REGRESSION_NUMERIC_SPLIT","score_":17.405000,"scoreImprove_":17.405000,"leftNodeSize_":1,"rightNodeSize_":1}),"leftChild_"
 {"id_":76,"sum_":30.100000,"sumSq_":906.010000,"size_":1,"maxDepth_":0,"value_":30.100000,"nodeType_":"REGRESSION_LEAF","prediction_":0.024056}),"rightChild_":
 }}}

```

## TD\_XGBoost using TD\_DataPartitioner

### Input Table

The input is a sample of the iris dataset, with 4 feature columns and a target (response) column 'species'.

<b>id</b>	<b>sepal_length</b>	<b>sepal_width</b>	<b>petal_length</b>	<b>petal_width</b>	<b>species</b>	<b>redist</b>
76	6.6	3.0	4.4	1.4	2	1
101	6.3	3.3	6.0	2.5	3	3

<b>id</b>	<b>sepal_length</b>	<b>sepal_width</b>	<b>petal_length</b>	<b>petal_width</b>	<b>species</b>	<b>redist</b>
57	6.3	3.3	4.7	1.6	2	1
122	5.6	2.8	4.9	2.0	3	2
38	4.9	3.6	1.4	0.1	1	1
99	5.1	2.5	3.0	1.1	2	2
61	5.0	2.0	3.5	1.0	2	1
78	6.7	3.0	5.0	1.7	2	3
93	5.8	2.6	4.0	1.2	2	3
17	5.4	3.9	1.3	0.4	1	2

## SQL Function Call

```
SELECT * FROM TD_XGBoost (
ON iris_train_redist AS inputtable
USING
INPUTCOLUMNS ('[1:4]')
RESPONSECOLUMN ('species')
DataRedistributionColumn ('redist')
MODELTYPE ('Classification')
MAXDEPTH (1 )
NUMBOOSTEDTREES (32 )
ITERNUM (2 )
SEED (1 )
MinRowsPerAmp(10)
isDebug('true')
) as dt
```

## Output

```
task_index tree_num iter class_num tree_order classification_tree
----- -----
0          1      1    0        0
{"id_":1,"sum_":-0.500000,"sumSq_":0.750000,"size_":3,"maxDepth_":1,"nodeType_":
"REGRESSION_NODE","split_":
{"splitValue_":5.850000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT",
"score_":0.666667,"scoreImprove_":0.666667,"leftNodeSize_":1,"rightNodeSize_":
2}, "leftChild_":
{"id_":2,"sum_":0.500000,"sumSq_":0.250000,"size_":1,"maxDepth_":0,"value_":0.50}
```

```

0000,"nodeType_":"REGRESSION_LEAF","prediction_":0.100000},"rightChild_":
{"id_":3,"sum_-1.000000,"sumSq_":0.500000,"size_":2,"maxDepth_":0,"value_-0.
500000,"nodeType_":"REGRESSION_LEAF","prediction_-0.166667}}
6      1      1      0      0
 {"id_":1,"sum_-0.333333,"sumSq_":0.777778,"size_":4,"maxDepth_":1,"nodeType_":
"REGRESSION_NODE","split_":
 {"splitValue_":5.250000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT
","score_":0.750000,"scoreImprove_":0.750000,"leftNodeSize_":1,"rightNodeSize_":
3),"leftChild_":
 {"id_":2,"sum_":0.666667,"sumSq_":0.444444,"size_":1,"maxDepth_":0,"value_":0.66
6667,"nodeType_":"REGRESSION_LEAF","prediction_":0.181818),"rightChild_":
 {"id_":3,"sum_-1.000000,"sumSq_":0.333333,"size_":3,"maxDepth_":0,"value_-0.
333333,"nodeType_":"REGRESSION_LEAF","prediction_-0.200000}}
7      1      1      0      0
 {"id_":1,"sum_-1.500000,"sumSq_":0.750000,"size_":3,"maxDepth_":0,"value_-0.
500000,"nodeType_":"REGRESSION_LEAF","prediction_-0.214286}
0      1      1      1      0
 {"id_":1,"sum_":0.500000,"sumSq_":0.750000,"size_":3,"maxDepth_":1,"nodeType_":
"REGRESSION_NODE","split_":
 {"splitValue_":5.850000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT
","score_":0.666667,"scoreImprove_":0.666667,"leftNodeSize_":1,"rightNodeSize_":
2),"leftChild_":
 {"id_":2,"sum_-0.500000,"sumSq_":0.250000,"size_":1,"maxDepth_":0,"value_-0.
500000,"nodeType_":"REGRESSION_LEAF","prediction_-0.100000),"rightChild_":
 {"id_":3,"sum_":1.000000,"sumSq_":0.500000,"size_":2,"maxDepth_":0,"value_":0.50
0000,"nodeType_":"REGRESSION_LEAF","prediction_":0.166667}}
6      1      1      1      0
 {"id_":1,"sum_-0.333333,"sumSq_":0.777778,"size_":4,"maxDepth_":1,"nodeType_":
"REGRESSION_NODE","split_":
 {"splitValue_":6.200000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT
","score_":0.250000,"scoreImprove_":0.250000,"leftNodeSize_":2,"rightNodeSize_":
2),"leftChild_":
 {"id_":2,"sum_":0.333333,"sumSq_":0.555556,"size_":2,"maxDepth_":0,"value_":0.16
6667,"nodeType_":"REGRESSION_LEAF","prediction_":0.076923),"rightChild_":
 {"id_":3,"sum_-0.666667,"sumSq_":0.222222,"size_":2,"maxDepth_":0,"value_-0.
333333,"nodeType_":"REGRESSION_LEAF","prediction_-0.153846}}
7      1      1      1      0
 {"id_":1,"sum_":0.500000,"sumSq_":0.750000,"size_":3,"maxDepth_":1,"nodeType_":
"REGRESSION_NODE","split_":
 {"splitValue_":5.300000,"attr_":"petal_length","type_":"REGRESSION_NUMERIC_SPLIT
","score_":0.666667,"scoreImprove_":0.666667,"leftNodeSize_":2,"rightNodeSize_":
1),"leftChild_":
 {"id_":2,"sum_":1.000000,"sumSq_":0.500000,"size_":2,"maxDepth_":0,"value_":0.50
0000,"nodeType_":"REGRESSION_LEAF","prediction_":0.166667),"rightChild_":

```

```

{"id_":3,"sum_":-0.500000,"sumSq_":0.250000,"size_":1,"maxDepth_":0,"value_":-0.500000,"nodeType_":"REGRESSION_LEAF","prediction_":-0.100000}
0           1       2   0       0
{"id_":1,"sum_":-0.384694,"sumSq_":0.551145,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":5.850000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.501815,"scoreImprove_":0.501815,"leftNodeSize_":1,"rightNodeSize_":2},"leftChild_":
{"id_":2,"sum_":0.450166,"sumSq_":0.202649,"size_":1,"maxDepth_":0,"value_":0.450166,"nodeType_":"REGRESSION_LEAF","prediction_":0.090212},"rightChild_":
{"id_":3,"sum_":-0.834860,"sumSq_":0.348495,"size_":2,"maxDepth_":0,"value_":-0.417430,"nodeType_":"REGRESSION_LEAF","prediction_":-0.140420}}
6           1       1   2       0
{"id_":1,"sum_":0.666667,"sumSq_":1.111111,"size_":4,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
{"splitValue_":6.200000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":1.000000,"scoreImprove_":1.000000,"leftNodeSize_":2,"rightNodeSize_":2},"leftChild_":
 {"id_":2,"sum_":-0.666667,"sumSq_":0.222222,"size_":2,"maxDepth_":0,"value_":-0.333333,"nodeType_":"REGRESSION_LEAF","prediction_":-0.153846},"rightChild_":
 {"id_":3,"sum_":1.333333,"sumSq_":0.888889,"size_":2,"maxDepth_":0,"value_":0.666667,"nodeType_":"REGRESSION_LEAF","prediction_":0.307692}}
7           1       2   0       0
{"id_":1,"sum_":-1.283254,"sumSq_":0.551779,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":5.300000,"attr_":"petal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.002866,"scoreImprove_":0.002866,"leftNodeSize_":2,"rightNodeSize_":1},"leftChild_":
 {"id_":2,"sum_":-0.811794,"sumSq_":0.329505,"size_":2,"maxDepth_":0,"value_":-0.405897,"nodeType_":"REGRESSION_LEAF","prediction_":-0.136916},"rightChild_":
 {"id_":3,"sum_":-0.471460,"sumSq_":0.222274,"size_":1,"maxDepth_":0,"value_":-0.471460,"nodeType_":"REGRESSION_LEAF","prediction_":-0.094353}}
0           1       2   1       0
{"id_":1,"sum_":0.384694,"sumSq_":0.551145,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":
 {"splitValue_":5.850000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.501815,"scoreImprove_":0.501815,"leftNodeSize_":1,"rightNodeSize_":2},"leftChild_":
 {"id_":2,"sum_":-0.450166,"sumSq_":0.202649,"size_":1,"maxDepth_":0,"value_":-0.450166,"nodeType_":"REGRESSION_LEAF","prediction_":-0.090212},"rightChild_":
 {"id_":3,"sum_":0.834860,"sumSq_":0.348495,"size_":2,"maxDepth_":0,"value_":0.417430,"nodeType_":"REGRESSION_LEAF","prediction_":0.140420}}

```

# Model Scoring Functions

- [TD\\_DecisionForestPredict](#)
- [TD\\_GLMPredict](#)
- [TD\\_KMeansPredict](#)
- [TD\\_NaiveBayesPredict](#)
- [TD\\_OneClassSVMPredict](#)
- [TD\\_SVMPredict](#)
- [TD\\_XGBoostPredict](#)

## TD\_DecisionForestPredict

TD\_DecisionForestPredict function uses the model output by TD\_DecisionForest function to analyze the input data and make predictions. This function outputs the probability that each observation is in the predicted class.

Processing times are controlled by the number of trees in the model. When the number of trees is more than what can fit in memory, then the trees are cached in a local spool space.

See [TD\\_DecisionForest](#).

### Function Information

- [TD\\_DecisionForestPredict Syntax](#)
- [Required Syntax Elements for TD\\_DecisionForestPredict](#)
- [Optional Syntax Elements for TD\\_DecisionForestPredict](#)
- [TD\\_DecisionForestPredict Input](#)
- [TD\\_DecisionForestPredict Output](#)
- [TD\\_DecisionForestPredict Usage Notes](#)
- [Example: How to Use TD\\_DecisionForestPredict](#)

## TD\_DecisionForestPredict Syntax

```
TD_DecisionForestPredict (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    ON { table | view | (query) } AS ModelTable DIMENSION
    USING
    IDColumn ('id_column')
    [ Detailed ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
    [ OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
    [ Responses ('response'[,...]) ]
```

```
[ Accumulate ({'accumulate_column'|'accumulate_column_range'}[,...]) ]
```

```
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_DecisionForestPredict

**ON clause**

Specifies the table name, view name or query as an InputTable and ModelTable.

**IDColumn**

Column with a unique identifier for each test point in the test set.

## Optional Syntax Elements for TD\_DecisionForestPredict

**Detailed**

Indicator to output detailed information about the decision tree and the specific tree information, including the task index and the tree index for each tree.

Default: false.

**OutputProb**

Indicator to output the probability for each response. If you omit Responses, the function outputs only the probability of the predicted class. Options are true and false. Must be true when using Responses.

Default: false.

**Note:**

The OutputProb argument only works with a classification model.

**Responses**

Classes to output the probabilities. Default behavior is to output only the probability of the predicted class.

**Note:**

The Responses argument only works with a classification model.

**Accumulate**

Names of the input columns to copy to the output table.

## TD\_DecisionForestPredict Input

TD\_DecisionForestPredict uses the following input tables.

Table	Description
InputTable	Contains test data, for which to predict outcomes. The input table can have no partition or PARTITION BY ANY clause.
ModelTable	Has the same schema as the output table of TD_DecisionForest function. Model table must be a DIMENSION table, and must be from TD_DecisionForest function.

### InputTable Schema

Column	Data Type	Description
ID_Column	Varies	Unique test point identifier. Cannot be NULL.
target_columns	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	Predictor variable. Column appears once for each specified target_column. Cannot be NULL.
accumulate_columns	Varies	Column to copy to output table. Column appears once for each specified accumulate_column.

### ModelTable Schema

#### For Classification

Name	Data Type	Description
task_index	SMALLINT	Identifier of AMP that produced decision tree.
tree_num	Integer	Decision tree identifier.
tree_order	Integer	Sequence of substring of tree.
classification_tree	VARCHAR(16000)	JSON representation of decision tree. For JSON types that can appear in the representation.

**For Regression:**

Name	Data Type	Description
task_index	SMALLINT	Identifier of AMP that produced decision tree.
tree_num	Integer	Decision tree identifier.
tree_order	Integer	Sequence of substring of tree.
regression_tree	VARCHAR(16000)	JSON representation of decision tree. For JSON types that can appear in the representation.

**JSON Types in JSON Representation of Decision Tree**

JSON Type	Description
id_	Node identifier.
sum_	Sum of values of response variable at node identified by id. Only appears for regression trees.
sumSq_	Sum of squared values of response variable at node identified by id. Only appears for regression trees.
responseCounts_	Number of observations in each class at node identified by id. Only appears for regression trees.
size_	Total number of observations at node identified by id.
maxDepth_	Maximum possible depth of tree, starting from node identified by id. For root node, the value is max_depth. For leaf nodes, the value is 0. For other nodes, maximum possible depth of tree, starting from that node.
split_	Start of JSON item describing a split at node identified by id.
score_	Gini score of node identified by id.
attr_	Attribute (predictor) that the algorithm split at node identified by id.
type_	Type of tree and split. Options are: <ul style="list-style-type: none"><li>• CLASSIFICATION_NUMERIC_SPLIT</li><li>• REGRESSION_NUMERIC_SPLIT</li></ul>
leftNodeSize_	Number of observations assigned to left node of split.
rightNodeSize_	Number of observations assigned to right node of split.
leftChild_	Start of JSON item describing left child of node identified by id.
rightChild_	Start of JSON item describing right child of node identified by id.
nodeType_	Type of node identified by id. Options are: <ul style="list-style-type: none"><li>• CLASSIFICATION_NODE</li><li>• CLASSIFICATION_LEAF</li><li>• REGRESSION_NODE</li></ul>

JSON Type	Description
	<ul style="list-style-type: none"> <li>REGRESSION_LEAF</li> </ul>

## TD\_DecisionForestPredict Output

### Output Table Schema

The table has a set of predictions for each test point.

Name	Data Type	Description
id_column	Same as input table	Column copied from input table. It is the unique row identifier.
prediction	<ul style="list-style-type: none"> <li>Integer for classification model</li> <li>FLOAT for regression model</li> </ul>	Predicted test point value or predicted class, determined by model.
confidence_lower	FLOAT	[Appears with OutputProb is false.] Lower bound of confidence interval. For classification trees, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.
confidence_upper	FLOAT	[Appears with OutputProb is false.] Upper bound of confidence interval. For classification trees, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.
prob	FLOAT	[Column appears only when OutputProb is true, and no Responses syntax element.] Probability that observation belongs to class prediction.
prob_response	FLOAT	[Column appears only when OutputProb is true and the Responses syntax element.] Probability that observation belongs to category response. Appears once for each specified response.
tree_num	VARCHAR(30)	[This column appears only when Detailed is true.] One of the following: <ul style="list-style-type: none"> <li>Concatenation of task_index and tree_num from the model table showing which tree created the prediction.</li> <li>Final to show the overall prediction.</li> </ul>
accumulate_column	Same as input table	Column copied from input table.

## TD\_DecisionForestPredict Usage Notes

Decision forests or random forests, are a type of machine learning algorithm used for both classification and regression tasks. They are composed of multiple decision trees, each trained on a random subset of the training data and a random subset of the available variables. During training, each decision tree in the forest learns to predict the target variable based on the input variables.

To predict unseen examples, the input data is fed to each tree in the forest, and the predictions from each tree are combined to produce a final output. This aggregation of predictions helps to reduce overfitting and improve the accuracy of the overall model.

### Example: How to Predict Using Decision Forests?

There is a dataset containing information about houses, including the size, number of bedrooms, location, and price. Build a model that can predict the price of a house given its variables. To do this, use a decision forest algorithm:

1. Split the dataset into a training set and a test set. Use the training set to train the decision forest model and the test set to evaluate its performance.
2. Build a decision forest model by training multiple decision trees on different subsets of the training data. Each decision tree is trained to predict the price of a house based on a random subset of variables.
3. During testing, feed the variables of a new house to each decision tree in the forest. Each tree produces a prediction of the house price, take the average of all the predictions to get the final predicted price.

For example, there is a new house with the following variables:

- size=2000 sq. ft.
- bedrooms=3
- location>New York

Feed this data to each tree in the decision forest and each tree produces a prediction of the house price based on a random subset of variables.

Here's what the predictions might look like for the first five trees in the forest:

- Tree 1: \$500,000
- Tree 2: \$450,000
- Tree 3: \$480,000
- Tree 4: \$510,000
- Tree 5: \$490,000

Take the average of all the predictions, to get the final prediction:

Final prediction:  $(\$500,000 + \$450,000 + \$480,000 + \$510,000 + \$490,000) / 5 = \$486,000$

Final predicted price for the house is \$486,000.

To illustrate using a different (classification) example, suppose you have the following decision forest with four trees:

Each of these trees is different from the other and can produce different predictions based on the input data.

Example, using the following data:

x0	x1	x2	x3	x4
2.7	6.7	4.2	5.3	4.8

The following are the predictions for each of the trees:

What will the decision forest predict? In the case above, since there are a greater number of 1s, the decision forest predicts the target variable for the row of data to be 1. In the case that there are an equal number of 1s and 0s, the decision forest will either predict the outcome variable randomly (0 or 1) or you can increase or decrease the number of trees to be an odd number so that the decision forest has a definite prediction.

## Example: How to Use TD\_DcisionForestPredict

### Example: TD\_DcisionForestPredict Used to Classify Home Styles

TD\_DcisionForestPredict InputTable

SN	Price	LotSize	Bedrooms	Bathrooms	Stories	Driveways	Recroom	FullBase	GasHW	A
1	42000	4960	2	1	1	1	0	0	0	0
2	130000	6000	4	1	2	1	0	1	0	0
3	60000	2953	3	1	2	1	0	1	0	1
4	43000	3750	3	1	2	1	0	0	0	0
5	86900	4300	6	2	2	1	0	0	0	0
6	141000	8100	4	1	2	1	1	1	0	1

ModelTable

task_index	tree_num	tree_order	classification_tree
0	0	0	{"id":1,"size":10,"maxDepth":5,"responseCounts":{"2":7,"3":3}, "nodeType":"CLASSIFICATION_NODE","split":{"splitValue":93721.000000,"attr":"price","type":"CLASSIFICATION_NUMERIC_SPLIT", "score":0.420000,"scoreImprove":0.420000,"leftNodeSize":7, "rightNodeSize":3},"leftChild":{"id":2,"size":7,"maxDepth":4,"label":2,"nodeType":"CLASSIFICATION_LEAF"}, "rightChild":{"id":3,"size":3,"maxDepth":4,"label":3,"nodeType":"CLASSIFICATION_LEAF"}}
1	0	0	{"id":1,"size":13,"maxDepth":5,"responseCounts":{"1":4,"2":8,"3":1}, "nodeType":"CLASSIFICATION_NODE","split":{"splitValue":49750.000000,"attr":"price","type":"CLASSIFICATION_NUMERIC_SPLIT", "score":0.520710,"scoreImprove":0.383958,"leftNodeSize":4, "rightNodeSize":9}, "leftChild":{"id":2,"size":4,"maxDepth":4,"label":1,"nodeType":"CLASSIFICATION_LEAF"}, "rightChild":{"id":3,"size":9,"maxDepth":4,"label":2,"nodeType":"CLASSIFICATION_LEAF"}}

task_index	tree_num	tree_order	classification_tree
			3,"size_":9,"maxDepth_":4,"responseCounts_":{"2":8,"3":1},"nodeType_":"CLASSIFICATION_NODE","split_":{"splitValue_":82750.000000,"attr_":"price","type_":"CLASSIFICATION_NUMERIC_SPLIT","score_":0.197531,"scoreImprove_":0.136752,"leftNodeSize_":8,"rightNodeSize_":1),"leftChild_":{"id_":6,"size_":8,"maxDepth_":3,"label_":2,"nodeType_":"CLASSIFICATION_LEAF"},"rightChild_":{"id_":7,"size_":1,"maxDepth_":3,"label_":3,"nodeType_":"CLASSIFICATION_LEAF"}}}
2	0	0	{"id_":1,"size_":6,"maxDepth_":5,"responseCounts_":{"1":1,"2":5}, "nodeType_":"CLASSIFICATION_NODE","split_":{"splitValue_":63000.000000,"attr_":"price","type_":"CLASSIFICATION_NUMERIC_SPLIT", "score_":0.277778,"scoreImprove_":0.277778,"leftNodeSize_":1, "rightNodeSize_":5),"leftChild_":{"id_":2,"size_":1,"maxDepth_":4,"label_":1,"nodeType_":"CLASSIFICATION_LEAF"},"rightChild_":{"id_":3,"size_":5,"maxDepth_":4,"label_":2,"nodeType_":"CLASSIFICATION_LEAF"}}}
3	0	0	{"id_":1,"size_":15,"maxDepth_":5,"label_":2,"nodeType_":"CLASSIFICATION_LEAF"}
4	0	0	{"id_":1,"size_":13,"maxDepth_":5,"responseCounts_":{"1":8,"2":2,"3":3}, "nodeType_":"CLASSIFICATION_NODE","split_":{"splitValue_":55000.000000,"attr_":"price","type_":"CLASSIFICATION_NUMERIC_SPLIT", "score_":0.544379,"scoreImprove_":0.359763,"leftNodeSize_":8,"rightNodeSize_":5),"leftChild_":{"id_":2,"size_":8,"maxDepth_":4, "label_":1,"nodeType_":"CLASSIFICATION_LEAF"},"rightChild_":{"id_":3,"size_":5,"maxDepth_":4,"responseCounts_":{"2":2,"3":3}, "nodeType_":"CLASSIFICATION_NODE","split_":{"splitValue_":105500.000000, "attr_":"price","type_":"CLASSIFICATION_NUMERIC_SPLIT", "score_":0.480000,"scoreImprove_":0.184615,"leftNodeSize_":2,"rightNodeSize_":3}, "leftChild_":{"id_":6,"size_":2,"maxDepth_":3,"label_":2,"nodeType_":"CLASSIFICATION_LEAF"}, "rightChild_":{"id_":7,"size_":3,"maxDepth_":3,"label_":3,"nodeType_":"CLASSIFICATION_LEAF"}}}
5	0	0	{"id_":1,"size_":15,"maxDepth_":5,"responseCounts_":{"1":11,"2":3, "3":1}, "nodeType_":"CLASSIFICATION_NODE","split_":{"splitValue_":56500.000000,"attr_":"price","type_":"CLASSIFICATION_NUMERIC_SPLIT", "score_":0.417778,"scoreImprove_":0.317778,"leftNodeSize_":11,"rightNodeSize_":4),"leftChild_":{"id_":2,"size_":11,"maxDepth_":4, "label_":1,"nodeType_":"CLASSIFICATION_LEAF"}, "rightChild_":{"id_":3,"size_":4,"maxDepth_":4,"responseCounts_":{"2":3,"3":1}, "nodeType_":"CLASSIFICATION_NODE","split_":{"splitValue_":102000.000000, "attr_":"price","type_":"CLASSIFICATION_NUMERIC_SPLIT", "score_":0.375000,"scoreImprove_":0.100000,"leftNodeSize_":3,"rightNodeSize_":1}, "leftChild_":{"id_":6,"size_":3,"maxDepth_":3,"label_":2,"nodeType_":"CLASSIFICATION_LEAF"}, "rightChild_":{"id_":7,"size_":1,"maxDepth_":3,"label_":3,"nodeType_":"CLASSIFICATION_LEAF"}}}
6	0	0	{"id_":1,"size_":4,"maxDepth_":5,"responseCounts_":{"1":1,"2":3}, "nodeType_":"CLASSIFICATION_NODE","split_":{"splitValue_":68000.000000,"attr_":"price","type_":"CLASSIFICATION_NUMERIC_SPLIT", "score_":0.375000,"scoreImprove_":0.375000,"leftNodeSize_":1, "rightNodeSize_":3),"leftChild_":{"id_":2,"size_":1,"maxDepth_":4,"label_":1,"nodeType_":"CLASSIFICATION_LEAF"}, "rightChild_":{"id_":3,"size_":3,"maxDepth_":4,"label_":2,"nodeType_":"CLASSIFICATION_LEAF"}}

## TD\_DecisionForestPredict Call

```
SELECT * FROM TD_DecisionForestPredict (
ON input_table AS InputTable PARTITION BY ANY
ON model_table AS ModelTable DIMENSION
USING
  IdColumn ('sn')
  Accumulate('homestyle')
  Detailed('false')
) AS dt;
```

## TD\_DecisionForestPredict Output

sn	Prediction	Confidence_Lower	Confidence_Higher	HomeStyle
1	1	0.7142857142	0.7142857142	1
2	3	0.5714285714	0.5714285714	3
3	2	0.7142857142	0.7142857142	2
4	1	0.7142857142	0.7142857142	1
5	2	0.8571428571	0.8571428571	2
6	3	0.5714285714	0.5714285714	3

## TD\_GLMPredict

TD\_GLMPredict function predicts target values (regression) and class labels (classification) for test data using a GLM model of the [TD\\_GLM](#) function.

Before using the features in the function, you must standardize the input features using [TD\\_ScaleFit](#) and [TD\\_ScaleTransform](#) functions.

TD\_GLMPredict only accepts numeric features. You must convert the categorical features to numeric values before prediction using:

- [TD\\_OneHotEncodingFit/TD\\_OneHotEncodingTransform](#)
- [TD\\_OrdinalEncodingFit/TD\\_OrdinalEncodingTransform](#)
- [TD\\_TargetEncodingFit/TD\\_TargetEncodingTransform](#)

You can use [TD\\_RegressionEvaluator](#), [TD\\_ClassificationEvaluator](#), or [TD\\_ROC](#) function as a post-processing step for evaluating prediction results.

**Note:**

Any observation with missing value (null) in an input column is ignored, and appears in the output with an error code. You can use an imputation function, such as [TD\\_SimpleImputeFit](#) and [TD\\_SimpleImputeTransform](#) to do imputation or fill in the missing values.

GLM prediction involves using a fitted model to make predictions about the response variable for new observations. Given the predictor variables for a new observation, the linear predictor is calculated using the coefficients estimated from the model. This linear predictor is then transformed using the link function to obtain an estimate of the mean of the response variable for the new observation.

The probability distribution assumed by GLM is then used to obtain a prediction interval for the response variable, which describes the range of values that the response variable is likely to fall within with a specified level of confidence. The width of the prediction interval depends on the variability of the response distribution and the uncertainty in the estimated model coefficients.

`TD_GLMPredict` can be used for a variety of tasks, such as predicting the probability of an event (binary classification), predicting the count of events (Poisson regression), or predicting a continuous outcome variable (linear regression).

The accuracy of GLM predictions are evaluated using measures such as the mean squared error or the proportion of correctly classified cases. It is important to assess the predictive performance of a GLM using a separate test dataset, as overfitting can occur when using the same data for both model fitting and prediction.

## Function Information

- [TD\\_GLMPredict Syntax](#)
- [Required Syntax Elements for TD\\_GLMPredict](#)
- [Optional Syntax Elements for TD\\_GLMPredict](#)
- [TD\\_GLMPredict Input](#)
- [TD\\_GLMPredict Output](#)
- [Examples: How to Use TD\\_GLMPredict](#)

## TD\_GLMPredict Syntax

**Important:**

In Analytics Database Release 17.20.03.08 and later, the Model alias is changed to ModelTable in the ON clause. If you are running releases 17.20.03.01 through 17.20.03.07, use Model.

### TD\_GLMPredict Syntax Using Partition by Any

```
TD_GLMPredict (
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY ]
    ON { table | view | (query) } AS ModelTable DIMENSION
```

```

USING
IDColumn ('id_column')
[Accumulate({'accumulate_column'|'accumulate_column_range'}[,...])]
[OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no','n','0'})]
[Responses ('response' [,...])]
[ Family ('Gaussian' | 'Binomial') ]
)

```

## TD\_GLMPredict Syntax Using Partition by Key

```

TD_GLMPredict (
  ON { table | view | (query) } AS InputTable PARTITION BY partition_by_column
  ON { table | view | (query) } AS ModelTable PARTITION BY partition_by_column
  USING
  IDColumn ('id_column')
  [PartitionColumn ('partition_column')]
  [Accumulate({'accumulate_column'|'accumulate_column_range'}[,...])]
  [OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no','n','0'})]
  [Responses ('response' [,...])]
  [ Family ('Gaussian' | 'Binomial') ]
)

```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_GLMPredict

### ON clause

Accepts the PARTITION BY ANY and DIMENSION clauses.

### IDColumn

Specify the column name that uniquely identifies an observation in test table.

## Optional Syntax Elements for TD\_GLMPredict

### Accumulate

Specify the input table column names to copy to the output table.

### OutputProb

Specify whether the function returns the probability for each response. Only applicable if family of probability distribution is BINOMIAL. The default value is false.

### Responses

Specify the class labels if the function returns probabilities for each response. Only applicable if the OutputProb element is True. A class label has the value 0 or 1. If not specified, the function returns the probability of the predicted response.

### Family

Specify the distribution exponential family that was used with TD\_GLM to train the model. Options are Gaussian and Binomial.

Default value is Gaussian.

## TD\_GLMPredict Input

### Input Table Schema

Column Name	Data Type	Description
id_column	Any	Column that uniquely identifies an observation in test table.
input_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	Column used as predictors (features) for model training.
partition_by_column	Any	Unique identifier of model for testing points in InputTable. They must be matched with partition_by_column in the model.
accumulate_column	Any	Input table columns to copy to the output table.

### Model Schema

See [TD\\_GLM](#) for a model example.

Column Name	Data Type	Description
partition_by_column	Any	Unique identifier of model.
attribute	SMALLINT	Numeric index that represents predictor and model metrics wherein model metrics have negative values and predictors take positive values. Intercept is specified using index 0.
predictor	VARCHAR	Predictor or model metric name. The maximum length is 32000 characters.
estimate	FLOAT	Predictor weights and numeric values of metrics.
value	VARCHAR	Value of metric string. The maximum length is 30 characters.

## TD\_GLMPredict Output

### Output Table Schema

Column Name	Data Type	Description
id_column	Any	Specified column name that uniquely identifies an observation in test table.
partition_by_column	CHARACTER, VARCHAR, INTEGER, BIGINT, SMALLINT, BYTEINT	[Partition by key only] Unique identifier of model in InputTable.
prediction	FLOAT	Predicted value of the test observation.
prob	FLOAT	Probability that the observation belongs to the predicted class. Only appears if the OutputProb element is set to True and the Responses element is not specified.
prob_0	FLOAT	Probability that the observation belongs to class 0. Only appears if the Responses element is specified.
prob_1	FLOAT	The probability that the observation belongs to class 1. Only appears if the Responses element is specified.
accumulate_column	Any	Specified column names in the Accumulate element copied to the output table.

## Examples: How to Use TD\_GLMPredict

### Example: TD\_GLMPredict Using Credit Data

This example takes credit data and uses TD\_GLM function to get a model. You can view the input and output in the [TD\\_GLM](#) example.

## TD\_GLMPredict Call for Credit Data

```
CREATE VOLATILE TABLE vt_glm_predict_credit_ex AS (
    SELECT * from TD_GLMPredict (
        ON credit_ex_merged AS INPUTTABLE
        ON td_glm_output_credit_ex AS Model DIMENSION
        USING
        IDColumn ('ID')
        Accumulate('Outcome')
    ) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;
```

## TD\_GLMPredict Output for Credit Data

ID	Prediction	Outcome
61	1	1
297	0	0
631	0	0
122	1	1
...	...	...

## Example: TD\_GLMPredict Using Housing Data

This example takes raw housing data, and does the following:

1. Uses [TD\\_ScaleFit](#) to standardize the data.
2. Uses [TD\\_ScaleTransform](#) to transform the data.
3. Uses [TD\\_GLM](#) to get a model.
4. Uses TD\_GLMPredict to predict target values.

You can view the input and output of steps 1 through 3 in the [TD\\_GLM](#) example.

## TD\_GLMPredict Call for Housing Data

```
CREATE VOLATILE TABLE vt_predict_cal_ex AS (
    SELECT * from TD_GLMPredict (
        ON cal_housing_ex_scaled AS INPUTTABLE
        ON td_glm_cal_ex AS Model DIMENSION
        USING
        IDColumn ('ID')
        Accumulate('MedHouseVal')
```

```

) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;

```

## **TD\_GLMPredict Output for Housing Data**

ID	Prediction	MedHouseVal
2833	1.5762	0.6
5328	2.29801	2.775
5300	1.82705	3.5
12433	0.863867	0.664
...	...	...

## **TD\_KMeansPredict**

TD\_KMeansPredict uses the k-means algorithm to predict the target class of unseen or new data.

The k-means algorithm calculates the distance between data points to determine the number of groups in a dataset based on the data point clusters.

TD\_KMeansPredict function follows this process:

1. Select the number of clusters (K).
2. Initialize the centroids.
3. Assign training data points to clusters based on their proximity to the different clusters
4. Recalculate centroids.
5. Repeat the process until the cluster assignments do not change a lot.
6. Use k-means algorithm to predict the cluster assignments of unseen/test data.

### **Function Information**

- [TD\\_KMeansPredict Syntax](#)
- [Required Syntax Elements for TD\\_KMeansPredict](#)
- [Optional Syntax Elements for TD\\_KMeansPredict](#)
- [TD\\_KMeansPredict Input](#)
- [TD\\_KMeansPredict Output](#)
- [Example: How to Use the k-means Algorithm](#)
- [Examples: How to Use TD\\_KMeansPredict](#)

## TD\_KMeansPredict Syntax

```
TD_KMeansPredict (
  ON { table | view | (query) } AS InputTable
  ON { table | view | (query) } AS ModelTable DIMENSION
  USING
  [ OutputDistance({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
  [ Accumulate({'accumulate_column' | 'accumulate_column_range'}[, ...]) ]
)
```

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_KMeansPredict

The following are the hyperparameters that you tune when using the TD\_KMeansPredict function:

### ON clause

Accepts the InputTable and ModelTable clauses.

- The input table can have no partition at all, or have the following combinations of PARTITION BY ANY/ORDER BY clauses:
  - PARTITION BY ANY ORDER BY
  - PARTITION BY ANY
- ModelTable must be a DIMENSION table with an optional ORDER BY clause. If you specify PARTITION BY column for ModelTable instead of DIMENSION, an error is reported

## Optional Syntax Elements for TD\_KMeansPredict

The following are the hyperparameters that you tune when using the TD\_KMeansPredict function:

### OutputDistance

Indicator to return the distance between each data point and the nearest cluster.

Default: False.

**Accumulate**

Input table column names to include in the output table.

**TD\_KMeansPredict Input****Input Table Schema**

Column	Data Type	Description
IdColumn	Any	InputTable column name that is the unique identifier for each input table row.
TargetColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL/NUMERIC, FLOAT, REAL, DOUBLE PRECISION	InputTable column names used for clustering.

**ModelTable Schema**

Same as OUT TABLE schema of ModelTable generated from [TD\\_KMeans](#).

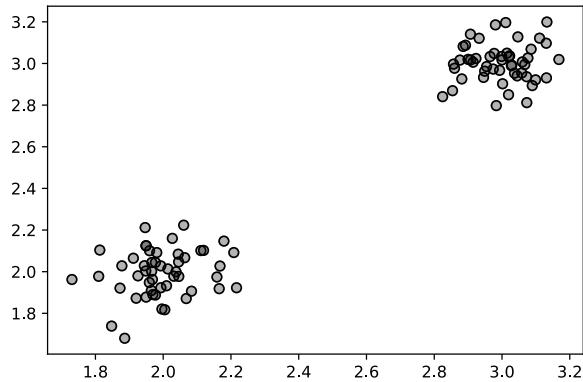
```
[ OUT [ PERMANENT | VOLATILE ] TABLE ModelTable(model_output_table_name) ]
```

**TD\_KMeansPredict Output****Output Table Schema**

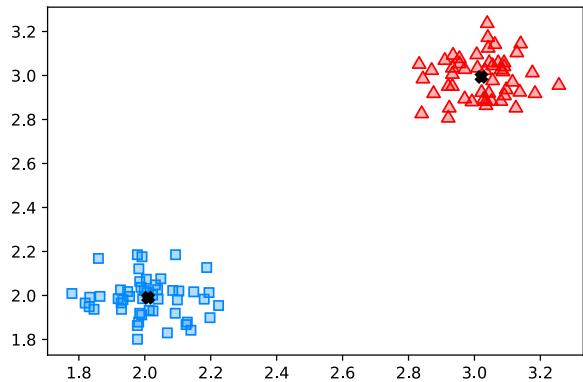
Column	Data Type	Description
Id_Column	ANY	Unique identifier of input rows from the input table.
TD_CLUSTERID_KMEANS	BIGINT	Unique identifier of the cluster.
TD_DISTANCE_KMEANS	REAL	Distance between a data point and the center of the assigned cluster.  <b>Note:</b> The query shows the column when you set the OutputDistance element to true.
Accumulate_Columns	ANY	Input table column names from the output table.

**Example: How to Use the k-means Algorithm**

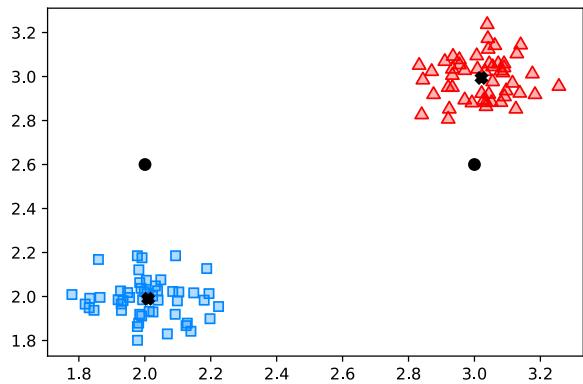
In the following example, you have a set of unlabeled or unclustered points.



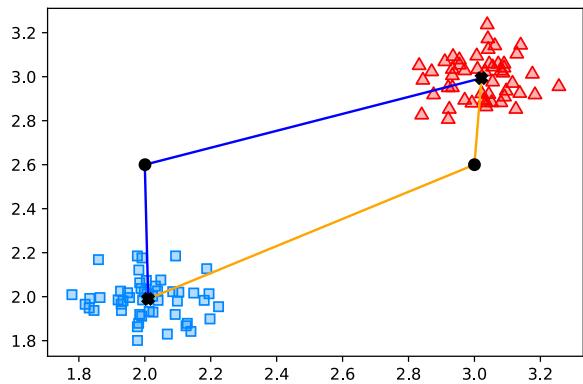
The k-means algorithm creates clusters. The points are shown as squares and triangles with the cluster centers shown as crosses:



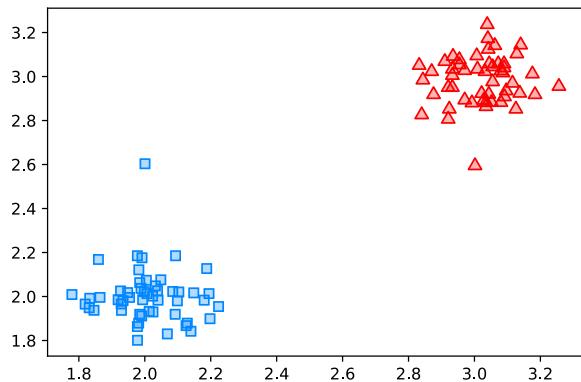
Then, new unlabeled data points are added to the set of points. The following image shows the new data points in circle:



To predict the label of the two new points, the k-means algorithm calculates the distances of each point from each cluster center or centroid. The k-means algorithm assigns the new point to the cluster whose centroid is closest to the new point.



In the previous image, the unknown point on the left is closer to the square cluster and the other is closer to the triangle cluster. The k-means algorithm assigns the new points to their closest cluster respectively for the calculation. The following image shows these assignments by transforming the points with their relevant figure:



## Examples: How to Use TD\_KMeansPredict

The TD\_KMeansPredict function uses the cluster centroids in the TD\_KMeans function output to assign the input data points to the cluster centroids.

### Input Table

This example uses the following input table:

id	C1	C2
1	1	1
2	2	2
3	8	8
4	9	9

### KMeans\_Model (generated using TD\_KMeans)

You can view the TD\_KMeans call provisioned with initial centroids table.

```
SELECT * FROM TD_KMeans (
    ON kmeans_input_table AS InputTable
    ON kmeans_initial_centroids_table AS InitialCentroidsTable DIMENSION
    USING
        IdColumn('id')
        TargetColumns('c1','c2')
        StopThreshold(0.0395)
        MaxIterNum(3)
) AS dt;
```

Result:

td_clusterid_kmeans	C1	C2	td_size_kmeans	td_withinss_kmeans	id	td_modelinfo_kmeans
0	1.5	1.5	2	1	NULL	NULL
1	8.5	8.5	2	1	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	Converged : True
NULL	NULL	NULL	NULL	NULL	NULL	Number of Iterations : 2
NULL	NULL	NULL	NULL	NULL	NULL	Number of Clusters : 2
NULL	NULL	NULL	NULL	NULL	NULL	Total_WithinSS 2.00000000000000E+00
NULL	NULL	NULL	NULL	NULL	NULL	Between_SS : 9.80000000000000E+01
NULL	NULL	NULL	NULL	NULL	NULL	Method For InitialCentroids :
Externally supplied InitialCentroidsTable						

## TD\_KMeansPredict Call

```
SELECT * FROM TD_KMeansPredict (
ON kmeans_input_table AS InputTable
ON kmeans_model AS ModelTable DIMENSION
USING
OutputDistance('true')
Accumulate('c1','c2')
)AS dt order by 1,2,3;
```

## TD\_KMeansPredict Output

id	td_clusterid_kmeans	td_distance_kmeans	C1	C2
--	--	--	--	--
1	0	0.707	1	1
2	0	0.707	2	2
3	1	0.707	8	8
4	1	0.707	9	9

If you set the value of OutputDistance to 'false' and rerun the query, the output shows these columns:

id	td_clusterid_kmeans	C1	C2
--	--	--	--
1	0	1	1
2	0	2	2
3	1	8	8
4	1	9	9

## TD\_NaiveBayesPredict

TD\_NaiveBayesPredict function uses the model generated by the TD\_NaiveBayes function to predict the outcomes for a test set of data.

### Function Information

- [TD\\_NaiveBayesPredict Syntax](#)

- [Required Syntax Elements for TD\\_NaiveBayesPredict](#)
- [Optional Syntax Elements for TD\\_NaiveBayesPredict](#)
- [TD\\_NaiveBayesPredict Input](#)
- [TD\\_NaiveBayesPredict Output](#)
- [Example: How to Use TD\\_NaiveBayesPredict](#)

## TD\_NaiveBayesPredict Syntax

```
TD_NaiveBayesPredict (
  ON { table | view | (query) } AS InputTable
  ON { table | view | (query) } AS ModelTable DIMENSION
  USING
  IDColumn ('id_column')
  [ Responses ('response' [,....]) ]
  [ OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
  [ Accumulate ({'accumulate_column' | 'accumulate_column_range'}[,....]) ]
  { for_dense_input |for_sparse_input }
)
```

### **for\_dense\_input**

```
[ NumericInputs ({ 'numeric_column' |
  'numeric_column_range' }[,....] )
[ CategoricalInputs ({ 'categorical_column' |
  'categorical_column_range' }[,....] ) ]
```

### **for\_sparse\_input**

```
AttributeNameColumn('attribute_name_column')
AttributeValueColumn('attribute_value_column')
```

## Required Syntax Elements for TD\_NaiveBayesPredict

### **ON clause**

Specifies the table name, view name, or query as InputTable and ModelTable.

### **IDColumn**

Specifies the column from the InputTable which contains the identifier of the rows or samples.

## Optional Syntax Elements for TD\_NaiveBayesPredict

### NumericInputs

Specify columns from the InputTable which contains the numeric attributes values.

---

**Note:**

Required only when input is in dense format and if you omit *CategoricalInputs*.

---

### CategoricalInputs

Specify columns from the InputTable which contains the categorical attributes values.

---

**Note:**

Required only when input is in dense format and if you omit *NumericInputs*.

---

### AttributeNameColumn

Specifies the column from the InputTable which contains the attributes.

---

**Note:**

Required only when input is in sparse format.

---

### AttributeValueColumn

Specifies the column from the InputTable which contains the attribute values..

---

**Note:**

Required only when input is in sparse format.

---

### Responses

Specify the responses to output.

### OutputProb

Specify whether to output the calculated probability.

### Accumulate

Specify the columns from the InputTable to copy to the output table.

## TD\_NaiveBayesPredict Input

### InputTable Schema for Dense Input

Column Name	Data Type	Description
id_column	Any	Column containing the identifier of the rows.
numeric_input_column	BYTEINT, SMALLINT, INT, BIGINT, REAL, DECIMAL, NUMBER	Column containing the numeric attribute values.
categorical_input_column	CHAR, VARCHAR	Column containing the categorical attribute values.
accumulate_column	Any	Columns copied from the input table.

### InputTable Schema for Sparse Input

Column Name	Data Type	Description
id_column	Any	Column containing the identifier of the partition or sample.
attribute_name_column	CHAR, VARCHAR	Column containing the attributes names.
attribute_value_column	BYTEINT, SMALLINT, INT, BIGINT, REAL, DECIMAL, NUMBER, CHAR, VARCHAR	Column containing the attributes values.
accumulate_column	Any	Columns copied from the input table.

## TD\_NaiveBayesPredict Output

### OutputTable Schema

Column Name	Data Type	Description
id_column	Any	Column containing the identifier of the rows or partition or sample.
Prediction	VARCHAR	Column containing the prediction value.
Loglik	REAL	Column appears only if responses argument is not present. Column contains the log likelihood of the predicted value.
Prob	REAL	Column appears only if responses argument is not present and Prob argument is present. The Column contains the probability of the predicted value.
Loglik_<response_i>	REAL	Column appears only if responses argument is present. The Column contains the log likelihood of all the response values.

Column Name	Data Type	Description
Prob_<response_i>	REAL	Column appears only if responses argument is present and Prob argument is present. The column contains the probability of all the response values.
accumulate_column	Any	Columns copied from the input table.

## Example: How to Use TD\_NaiveBayesPredict

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

This section shows the input table, SQL query, and output tables of an example using TD\_NaiveBayesPredict.

The following table contains a subset of housing dataset.

### Note:

Only part of the dataset is shown in this example.

```

DROP TABLE housing_test;
CREATE TABLE housing_test (sn integer, price real, lotsize real, bedrooms
integer, bathrms integer, stories integer, driveway varchar(10), recroom
varchar(10), fullbase varchar(10), gashw varchar(10), airco varchar(10), garagepl
integer, prefarea varchar(10), homestyle varchar(10)) PRIMARY INDEX(sn);
insert into housing_test
values(13,27000,1700,3,1,2,'yes','no','no','no','no',0,'no','Classic');
insert into housing_test
values(16,37900,3185,2,1,1,'yes','no','no','no','yes',0,'no','Classic');
insert into housing_test
values(25,42000,4960,2,1,1,'yes','no','no','no','no',0,'no','Classic');
insert into housing_test
values(38,67000,5170,3,1,4,'yes','no','no','no','yes',0,'no','Eclectic');
insert into housing_test
values(53,68000,9166,2,1,1,'yes','no','yes','no','yes',2,'no','Eclectic');
...
SELECT * FROM housing_test ORDER BY 1;

```

### TD\_NaiveBayesPredict SQL Call

```

SELECT * FROM TD_NaiveBayesPredict (
    ON housing_test AS InputTable

```

```

ON housing_train_model AS ModelTable DIMENSION
USING
IDColumn ('sn')
NumericInputs('price','lotsize','bedrooms','bathrms','stories','garagepl')
CategoricalInputs('driveway','recroom','fullbase','gashw','airco','prefarea')
Responses ('Classic', 'Eclectic', 'bungallow')
OutputProb('t')
Accumulate('homestyle')
) AS dt ORDER BY sn;

```

**TD\_NaiveBayesPredict Output Table**

sn	Prediction	Loglik_Classic	Loglik_Eclectic	Loglik_bungalow	Prob_Classic	Prob_Eclectic	Prob_bungalow	homestyle
13	Classic	-25. 2150421	-30. 7597	-44. 04708467	0. 99610712	0. 00389288	. 00000001	Classic
16	Classic	-24. 502771	-29. 6444	-42. 43191857	0. 99418596	0. 00581403	. 00000002	Classic
25	Classic	-22. 1674829	-27. 7935	-41. 38195573	0. 99640986	0. 00359014	. 00000000	Classic
38	Eclectic	-40. 7803961	-28. 2814	-33. 56134748	0. 00000371	0. 99492952	. 00506677	Eclectic
53	Eclectic	-42. 1760628	-28.75	-35. 44498472	0. 00000147	0. 99876298	. 00123555	Eclectic
...	...	...	...	...	...	...	...	...

As the data is in dense format, you need to use TD\_Unpivoting function to change the data to sparse format.

```

DROP TABLE housing_test_sparse;
CREATE MULTISET TABLE housing_test_sparse AS(
SELECT * FROM TD_UNPIVOTING(
ON housing_test AS InputTable
USING
IDColumn('sn')
TargetColumns('price','lotsize','bedrooms','bathrms','stories','garagepl','drive
way','recroom','fullbase','gashw','airco','prefarea')
AttributeName('AttributeName')
ValueColumnName('AttributeValue')
Accumulate('homestyle')
) AS dt) WITH data;

```

## TD\_NaiveBayesPredict SQL Call

```
SELECT * FROM TD_NaiveBayesPredict (
    ON housing_test_sparse AS InputTable
    ON housing_train_model_sparse AS ModelTable DIMENSION
    USING
        IDColumn ('sn')
        AttributeNameColumn('AttributeName')
        AttributeValueColumn('AttributeValue')
        Responses ('Classic', 'Eclectic', 'bungallow')
        OutputProb('t')
        Accumulate('homestyle')
) AS dt ORDER BY sn;
```

TD\_NaiveBayesPredict Output Table

sn	Prediction	Loglik_Classic	Loglik_Eclectic	Loglik_bungalow	Prob_Classic	Prob_Eclectic	Prob_bungalow	homestyle
13	Classic	-25. 2150421	-30. 7597	-44. 04708467	0. 99610712	0. 00389288	. 00000001	Classic
16	Classic	-24. 502771	-29. 6444	-42. 43191857	0. 99418596	0. 00581403	. 00000002	Classic
25	Classic	-22. 1674829	-27. 7935	-41. 38195573	0. 99640986	0. 00359014	. 00000000	Classic
38	Eclectic	-40. 7803961	-28. 2814	-33. 56134748	0. 00000371	0. 99492952	. 00506677	Eclectic
53	Eclectic	-42. 1760628	-28.75	-35. 44498472	0. 00000147	0. 99876298	. 00123555	Eclectic
...	...	...	...	...	...	...	...	...

## TD\_OneClassSVMPredict

TD\_OneClassSVMPredict predicts target class labels (classification) for test data using a one-class SVM model trained by [TD\\_OneClassSVM](#). Output values are 0 and 1. A value of 1 corresponds to a 'normal' observation, and a value of 0 is assigned to 'outlier' observations.

### Assumptions

Similar to TD\_OneClassSVM, input features must be standardized, such as using [TD\\_ScaleFit](#) and [TD\\_ScaleTransform](#), before using in the function. The function takes only numeric features. The categorical features must be converted to numeric values prior to prediction. Rows with missing (null) values are skipped by the function during prediction. For prediction results evaluation, you can use [TD\\_ClassificationEvaluator](#) or [TD\\_ROC](#) as the postprocessing step.

OneClassSVM is a type of support vector machine algorithm used for outlier detection or novelty detection. Unlike traditional SVMs that are used for classification and regression, OneClassSVM is an unsupervised learning algorithm that learns to identify anomalies or outliers in a given dataset.

To make a prediction using OneClassSVM, the algorithm first learns the normal behavior of the dataset during training. Then, during testing, it evaluates new data points to determine if they fit within the normal behavior of the dataset or if they are outliers.

OneClassSVM is commonly used in various fields such as fraud detection, intrusion detection, and fault detection in industrial systems, where detecting anomalies is critical for maintaining the safety and reliability of the systems.

The algorithm works by learning the normal behavior of the dataset during training. It does this by creating a hyperplane that separates the majority of the data points from the rest of the dataset. The hyperplane is created in such a way that it maximizes the margin between the hyperplane and the closest data points. The data points that fall outside of this margin are considered as outliers or anomalies.

During testing, the algorithm predicts whether a new observation belongs to the normal behavior of the dataset or not. To make this prediction, the algorithm evaluates the distance of the new observation to the hyperplane. If the new observation falls within the margin or on the same side as the majority of the training data, it is considered as a normal data point. If the new observation falls outside the margin or on the opposite side of the majority of the training data, it is considered as an outlier.

Here's how it works:

1. Data Preprocessing: The first step is to preprocess the dataset to prepare it for training. This typically involves removing any missing values, scaling the features, and removing any noise or irrelevant information.
2. Training: The next step is to train the OneClassSVM model using the preprocessed data. The model learns to identify the normal behavior of the dataset during this step. It does this by finding a hyperplane that separates the majority of the data points from the rest of the dataset. This hyperplane is created in such a way that it maximizes the margin between the hyperplane and the closest data points.
3. Prediction: Once the model has been trained, it can be used to predict whether new observations belong to the normal behavior of the dataset or not. During prediction, the algorithm evaluates the distance of the new observation to the hyperplane. If the new observation falls within the margin or on the same side as the majority of the training data, it is considered as a normal data point. If the new observation falls outside the margin or on the opposite side of the majority of the training data, it is considered as an outlier.
4. Parameter Tuning: One of the challenges of using OneClassSVM is selecting the right values for the hyperparameters. The hyperparameters control the behavior of the algorithm, such as the width of the margin and the type of kernel used. To select the best hyperparameters, cross-validation can be used to evaluate the performance of the model on a held-out dataset.
5. Model Evaluation: Once the model has been trained and tuned, it is important to evaluate its performance. This typically involves using metrics such as precision, recall, and F1-score to assess how well the model can identify outliers and normal data points.

One of the benefits of OneClassSVM is that it can handle datasets with a high dimensionality. It can also handle datasets with a small number of observations, as long as the observations are representative of

the normal behavior of the dataset. Also, OneClassSVM has a main application in anomaly detection. For example, in fraud detection, OneClassSVM can be used to identify credit card transactions that are unusual or deviate from the normal behavior of the dataset. In intrusion detection, OneClassSVM can be used to identify network traffic that is anomalous or potentially malicious. In industrial systems, OneClassSVM can be used to identify faulty or abnormal behavior in machinery, which can help prevent equipment failure and downtime.

## Function Information

- [TD\\_OneClassSMPredict Syntax](#)
- [Required Syntax Elements for TD\\_OneClassSMPredict](#)
- [Optional Syntax Elements for TD\\_OneClassSMPredict](#)
- [TD\\_OneClassSMPredict Input](#)
- [TD\\_OneClassSMPredict Output](#)
- [Example: How to Use TD\\_OneClassSMPredict](#)

## TD\_OneClassSMPredict Syntax

```
TD_OneClassSMPredict (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    ON { table | view | (query) } AS ModelTable DIMENSION
    USING
        IDColumn ('id_column')
        [Accumulate({'accumulate_column'|'accumulate_column_range'}[,...])]
        [OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no','n','0'})]
        [Responses ('response' [,...])]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_OneClassSMPredict

### ON clause

Accepts the InputTable and ModelTable clauses.

**IDColumn**

Specify the name of the column that uniquely identifies an observation in test table.

## Optional Syntax Elements for TD\_OneClassSVMPredict

**Accumulate**

Specify the names of the input table columns that need to be copied from input test table to output.

**OutputProb**

Specify whether the function outputs the probability for each response. Only applicable if ModelType is CLASSIFICATION. Default value is false.

**Responses**

Specify the class labels to output probabilities. Only applicable if OutputProb is true. A label must be 0 or 1. If not specified, the function outputs the probability of the predicted response.

## TD\_OneClassSVMPredict Input

TD\_OneClassSVMPredict accepts two inputs:

- InputTable containing test input data set.
- ModelTable containing an SVM model trained by TD\_OneClassSVM.

### Input Table Schema

Column Name	Data Type	Description
id_column	ANY	Unique row identifier of input observations.
target_column	INTEGER, BIGINT, SMALLINT, BYTEINT, DOUBLE PRECISION	Column used as predictors (features) for model training.
accumulate_column	ANY	Columns to copy to output table.

### Model Table Schema

Column Name	Data Type	Description
attribute	SMALLINT	Contains a numeric index of predictor and model metrics. Intercept is specified using index 0 and rest of the predictors take positive values. Model metrics take negative indices.
predictor	VARCHAR	Contains the name of the predictor or Model metric.

Column Name	Data Type	Description
estimate	FLOAT	Contains the predictor weights and metric numeric values.
value	VARCHAR	Contains the string values of the metric.

## TD\_OneClassSVMPredict Output

### Model Output Schema

Column	Data Type	Description
prediction	FLOAT	Predicted value of the test observation.
prob	FLOAT	Probability that observation belongs to predicted class. Only appears if OutputProb is true and Responses is not specified.
prob_0	FLOAT	Probability that observation belongs to class 0. Only appears if Responses is specified.
prob_1	FLOAT	Probability that observation belongs to class 1. Only appears if Responses is specified.
accumulate_column	ANY	Column copied from input table.

## Example: How to Use TD\_OneClassSVMPredict

### TD\_OneClassSVMPredict Example: Outlier/Novelty Detection

The input is the diabetes\_train\_scaled set used in the example of [TD\\_OneClassSVM](#).

ID	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
5	1	139	80	23	140	27.1	1.441
1	5	116	74	0	0	25.6	.0201
8	2	197	70	45	45	30.5	0.158
2	2	87	58	16	16	32.7	0.166
4	0	128	68	19	19	30.5	0.121
6	4	130	70	0	0	34.2	0.652
7	10	115	0	0	0	35.3	0.134
3	7	129	68	49	49	38.5	0.439
10	4	110	92	0	0	37.6	0.191
9	0	125	96	0	0	22.5	0.262

**TD\_OneClassSVMPredict Model**

attribute	predictor	estimate	value
-3	Number of Observations	10.00000	None
-12	Nesterov	NaN	FALSE
-16	Kernel	NaN	LINEAR
0	(Intercept)	0.094000	None
5	Insulin	0.988265	None
-5	BIC	20.723266	None
-1	Loss Function	NaN	HINGE
-9	Learning Rate (Initial)	0.010000	None
3	BloodPressure	0.642372	None
-7	Alpha	0.000000	L2
7	DiabetesPedigreeFunction	0.003578	None
-17	OneClass SVM	NaN	TRUE
1	Pregnancies	0.033259	None
-13	LocalSGD Iterations	0.000000	None
-11	Momentum	0.000000	None
-2	Loglik	-0.000000	None
-15	Intercept Scaling	1.000000	None
-8	Number of Iterations	52.000000	CONVERGED
-4	AIC	18.000000	None
6	BMI	0.298855	None
8	Age	0.349694	None
-10	Learning Rate (Final)	0.010000	None
4	SkinThickness	.0144439	None
-6	Regularization	0.100000	ENABLED
2	Glucose	1.212525	None

## TD\_OneClassSVMPredict Call

```
CREATE multiset table prediction AS (
    SELECT * from TD_OneClassSVMPredict(
        ON diabetes_test_scaled AS InputTable PARTITION BY ANY
        ON oneclasssvm_model AS ModelTable DIMENSION
        USING
        IDColumn('id')
    ) AS dt
) WITH DATA;
```

## TD\_OneClassSVMPredict Test Data

ID	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
20	2	122	70	27	0	36.8	0.340
17	1	97	66	15	140	23.2	0.487
12	1	168	88	29	0	35.0	0.905
19	10	101	76	48	180	32.9	0.171
18	9	145	88	34	0	30.3	0.771
13	5	139	80	35	160	31.6	0.361
15	10	119	80	0	0	32.4	0.141
11	6	125	78	31	0	27.6	0.565
14	2	99	52	15	94	24.6	0.637
16	3	103	70	30	48	27.6	0.730

## TD\_OneClassSVMPredict Output

ID	prediction
20	1.0
17	1.0
12	1.0
19	1.0
18	1.0
13	1.0
15	1.0

ID	prediction
11	1.0
14	1.0
16	1.0

Count normal as opposed to outlier.

```
SELECT CASE "prediction"
    WHEN 1 THEN 'normal'
    WHEN 0 THEN 'outlier'
    END SIGN,
    COUNT (*) counter
from prediction
GROUP BY "prediction"
```

SIGN	counter
normal	10

## TD\_SVMPredict

TD\_SVMPredict predicts target values (regression) and class labels (classification) for test data using an SVM model trained by [TD\\_SVM](#).

Similar to TD\_SVM, input features must be standardized, such as using [TD\\_ScaleFit](#) and [TD\\_ScaleTransform](#), before using in the function. The function takes only numeric features. The categorical features must be converted to numeric values prior to prediction. Rows with missing (null) values are skipped by the function during prediction. For prediction results evaluation, you can use [TD\\_RegressionEvaluator](#), [TD\\_ClassificationEvaluator](#), or [TD\\_ROC](#) function as postprocessing step.

### Function Information

- [TD\\_SVMPredict Syntax](#)
- [Required Syntax Elements for TD\\_SVMPredict](#)
- [Optional Syntax Elements for TD\\_SVMPredict](#)
- [TD\\_SVMPredict Input](#)
- [TD\\_SVMPredict Output](#)
- [TD\\_SVMPredict Usage Notes](#)
- [Examples: How to Use TD\\_SVMPredict](#)

## TD\_SVMPredict Syntax

```
TD_SVMPredict (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    ON { table | view | (query) } AS ModelTable DIMENSION
    USING
        IDColumn ('id_column')
    [Accumulate({'accumulate_column'|'accumulate_column_range'}[,...])]
    [OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no','n','0'})]
    [Responses ('response' [,...])]
    [ ModelType ('Classification' | 'Regression') ]
)
```

## Required Syntax Elements for TD\_SVMPredict

### ON clause

Accepts the InputTable and ModelTable clauses.

### IDColumn

Specifies the name of the column that uniquely identifies an observation in the test table.

## Optional Syntax Elements for TD\_SVMPredict

### Accumulate

Specifies the names of the input table columns that need to be copied from input test table to output.

### OutputProb

Specifies whether the function outputs the probability for each response. Only applicable if ModelType is Classification.

Default: false

### Responses

Specifies the class labels to output probabilities. Only applicable if OutputProb is true. A label must be 0 or 1. If not specified, the function outputs the probability of the predicted response.

### ModelType

Specify the model type used by TD\_SVM to train the dataset. Acceptable values are Regression, Classification.

Default: Classification

## TD\_SVMPredict Input

TD\_SVMPredict accepts two inputs:

- InputTable containing test input data set.
- ModelTable containing an SVM model trained by [TD\\_SVM](#).

### TD\_SVMPredict InputTable Schema

Column Name	Data Type	Description
id_column	ANY	Unique row identifier of input observations.
target_column	INTEGER, BIGINT, SMALLINT, BYTEINT, DOUBLE PRECISION	Column used as predictors (features) for model training.
accumulate_column	ANY	Columns to copy to output table.

### TD\_SVMPredict Model Table Schema

Column Name	Data Type	Description
attribute	SMALLINT	Contains a numeric index of predictor and model metrics. Intercept is specified using index 0 and rest of the predictors have positive values. Model metrics take negative indices.
predictor	VARCHAR	Contains the name of the predictor or model metric.
estimate	FLOAT	Columns to copy to output table. Contains the predictor weights and metric numeric values.
value	VARCHAR	Contains the string values of the metric.

## TD\_SVMPredict Output

The output schema is as follows:

Column	Data Type	Description
id_column	Same as input table	Unique row identifier of input observations.
prediction	SMALLINIT, FLOAT	Predicted value of the test observation.
prob	FLOAT	Probability that observation belongs to predicted class. Only appears if OutputProb is true and Responses is not specified.

Column	Data Type	Description
prob_0	FLOAT	Probability that observation belongs to class 0. Only appears if Responses is specified.
prob_1	FLOAT	Probability that observation belongs to class 1. Only appears if Responses is specified.
accumulate_column	ANY	Column copied from input table.

## TD\_SVMPredict Usage Notes

### SVM Optimization

See [TD\\_SVM](#).

### How Support Vector Machine Predicts on the Unseen Data

Support Vector Machine (SVM) uses a learned hyperplane to make predictions on unseen data points. The hyperplane is determined during the training phase by minimizing the error or maximizing the margin, depending on the type of task (classification or regression). The kernel trick is used to transform the input data into a higher-dimensional space if necessary to improve the performance of the SVM algorithm on complex, non-linear datasets.

Classification:

1. Feature Extraction: Extracts the relevant features from the unseen data point.
2. Distance Calculation: Calculates the distance between the new data point and the hyperplane that was learned during the training phase.
3. Decision Rule: The sign of the distance indicates which side of the hyperplane the data point falls on.
  - If the distance is positive, the data point is classified as belonging to the positive class.
  - If the distance is negative, the data point is classified as belonging to the negative class.
4. Confidence Estimation: The magnitude of the distance is also used to estimate the confidence level of the prediction.
  - A larger distance indicates a higher confidence in the prediction.
  - A smaller distance indicates lower confidence.

Regression:

1. Feature Extraction: Extracts the relevant features from the new data point.
2. Hyperplane Calculation: Calculates the predicted output value for the new data point based on the learned hyperplane during the training phase.
3. Confidence Estimation: The distance between the predicted output value and the hyperplane is used to estimate the confidence level of the prediction.
  - A smaller distance indicates a higher confidence in the prediction.
  - A larger distance indicates lower confidence.

TD\_SVMPredict function uses classification and regression tasks.

## Examples: How to Use TD\_SVMPredict

The following data set is used in the examples.

<b>id</b>	<b>MedInc</b>	<b>HouseAge</b>	<b>AveRooms</b>	<b>AveBedrms</b>	<b>Population</b>	<b>AveOccup</b>	<b>Latitude</b>	<b>Longitude</b>
14870	1.858	23	3.901	1.077	1025	2.47	32.64	-117.11
6044	2.114	27	3.855	1.072	1024	4.633	34.05	-117.74
3593	6.654	32	6.331	0.995	1285	3.104	34.24	-118.48
9454	1.228	25	5.504	1.154	991	2.629	39.77	-123.23
18760	3.282	16	5.998	1.076	1414	3.081	40.6	-122.25
11670	4.5	28	5.102	1.044	2112	2.63	33.84	-118.01
17768	2.756	29	4.53	1.04	3572	4.603	37.35	-121.85
244	2.391	44	4.866	1.164	2269	3.72	37.78	-122.22
5328	2.768	23	3.039	1.064	2031	1.637	34.04	-118.45
14365	2.164	43	4.533	0.995	392	1.867	32.72	-117.23
2313	2.486	15	5.468	1.045	649	2.449	36.94	-119.7
12342	2.588	28	6.268	1.372	3470	2.59	33.84	-116.53
6558	6.827	36	7.021	1.036	1897	2.71	34.2	-118.11
17157	9.78	20	6.678	0.918	324	2.219	37.43	-122.21
18099	5.753	27	6.437	1.027	1259	2.868	37.32	-122.01

### Model

<b>attribute</b>	<b>predictor</b>	<b>estimate</b>	<b>value</b>
-17	OneClass SVM	NaN	FALSE
-16	Kernel	NaN	LINEAR
-15	Intercept Scaling	1.000000	None
-14	Epsilon	0.100000	None
-13	LocalSGD Iterations	0.000000	None
-12	Nesterov	NaN	FALSE
-11	Momentum	0.000000	None

attribute	predictor	estimate	value
-10	Learning Rate (Final)	0.204246	None
-9	Learning Rate (Initial)	0.050000	None
-8	Number of Iterations	227.000000	CONVERGED
-7	Alpha	0.150000	Elasticnet
-6	Regularization	0.020000	ENABLED
-5	BIC	5.572674	None
-4	AIC	-0.799777	None
-3	Number of Observations	15.000000	None
-2	MSE	0.285556	None
-1	Loss Function	NaN	EPSILON_INSENSITIVE
0	(Intercept)	2.191718	None
1	MedInc	1.226729	None
2	HouseAge	.0232213	None
3	AveRooms	-.332659	None
4	AveBedrms	0.000000	None
5	Population	0.113633	None
6	AveOccup	-0.286434	None
7	Latitude	-0.261509	None
8	Longitude	-0.198519	None

### Example: TD\_SVMPredict Call Using Regression

```

CREATE VOLATILE TABLE svm_m0del_predict_cal_housing AS (
SELECT * from TD_SVMPredict (
    ON cal_housing_ex_scaled AS INPUTTABLE
    ON svm_model_cal_housing AS ModelTable DIMENSION
    USING
        IDColumn ('id')
        Accumulate('MedHouseVal')
) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;

```

**TD\_SVMPredict Output for Regression**

<b>id</b>	<b>prediction</b>	<b>MedHouseVal</b>
18760	1.268192	1.28300
244	1.815538	1.11700
9454	0.570297	0.60300
14365	2.280287	2.44200
6558	3.628097	3.59400
12342	1.506883	1.59000
5328	2.709805	2.77500
14870	1.601341	0.67500
17768	1.546305	1.60100
18099	2.855638	4.31400
2313	0.976091	0.86300
3593	3.445194	2.67600
6044	1.025779	1.10900
11670	2.814743	2.02100
17157	4.831587	5.00001

**Classification iris Data Set**

<b>id</b>	<b>sepal_length</b>	<b>sepal_width</b>	<b>petal_length</b>	<b>petal_width</b>	<b>variety</b>	<b>label</b>
61	5.0	2.0	3.5	1.0	Versicolor	0
51	7.0	3.2	4.7	1.4	Versicolor	0
50	5.1	3.4	1.5	0.2	Setosa	1
59	6.6	2.9	4.6	1.3	Versicolor	0
38	4.9	3.6	1.4	0.1	Setosa	1
...	...	...	...	...	...	...
90	5.5	2.5	4.0	1.3	Versicolor	0
67	5.6	3.0	4.5	1.5	Versicolor	0
44	5.0	3.5	1.6	0.6	Setosa	1
42	4.5	2.3	1.3	0.3	Setosa	1

<b>id</b>	<b>sepal_length</b>	<b>sepal_width</b>	<b>petal_length</b>	<b>petal_width</b>	<b>variety</b>	<b>label</b>
82	5.5	2.4	3.7	1.0	Versicolor	0

**Model**

<b>attribute</b>	<b>predictor</b>	<b>estimate</b>	<b>value</b>
-17	OneClass SVM	NaN	FALSE
-16	Kernel	NaN	LINEAR
-15	Intercept Scaling	1.000000	None
-13	LocalSGD Iterations	0.000000	None
-12	Nesterov	NaN	FALSE
-11	Momentum	0.000000	None
-10	Learning Rate (Final)	0.677479	None
-9	Learning Rate (Initial)	0.050000	None
-8	Number of Iterations	56.000000	CONVERGED
-7	Alpha	0.150000	Elasticnet
-6	Regularization	0.020000	ENABLED
-5	BIC	20.794415	None
-4	AIC	10.000000	None
-3	Number of Observations	64.000000	None
-2	Loglik	-0.000000	None
-1	Loss Function	NaN	HINGE
0	(Intercept)	0.43063	None
1	sepal_length	0.163885	None
2	sepal_width	0.907101	None
3	petal_length	-1.411058	None
4	petal_width	-0.527589	None

**Example: TD\_SVMPredict Call Using Classification**

```
CREATE VOLATILE TABLE svm_model_predict_iris_data AS (
SELECT * FROM TD_SVMPredict (
ON iris_data AS INPUTTABLE
```

```

ON svm_model_iris_data AS ModelTable DIMENSION
USING
IdColumn('id')
Accumulate('label')
OutputProb('true')
Responses('0','1')
) AS dt
) WITH DATA
ON COMMIT PRESERVE ROWS;

```

### **TD\_SVMPredict Output for Classification**

<b>id</b>	<b>prediction</b>	<b>prob_0</b>	<b>prob_1</b>	<b>label</b>
61	0.0	0.916982	0.083018	0
51	0.0	0.947352	0.052648	0
40	1.0	0.106359	0.893641	1
59	0.0	0.954081	0.045919	0
38	1.0	0.077917	0.922083	1
...	...	...	...	...
90	0.0	0.938794	0.061206	0
67	0.0	0.955700	0.044300	0
44	1.0	0.135795	0.864205	1
42	1.0	0.220665	0.779335	1
82	0.0	0.903738	0.096262	0

### **TD\_XGBoostPredict**

TD\_XGBoostPredict function runs the predictive algorithm based on the model generated by TD\_XGBoost. TD\_XGBoost function, also known as eXtreme Gradient Boosting, performs classification and regression analysis on datasets. See [TD\\_XGBoost](#).

TD\_XGBoostPredict performs prediction for test input data using multiple simple trees in the trained model. The test input data must have the same attributes as used during the training phase, which can be up to 2046. These attributes are used to score based on the trees in the model.

The output contains prediction for each data point in the test data based on regression or classification. The prediction probability is computed based on the majority vote from participating trees. A higher probability implies a more confident prediction by the model. Majority of the trees result in the same prediction.

## Function Information

- [TD\\_XGBoostPredict Syntax](#)
- [Required Syntax Elements for TD\\_XGBoostPredict](#)
- [Optional Syntax Elements for TD\\_XGBoostPredict](#)
- [TD\\_XGBoostPredict Input](#)
- [TD\\_XGBoostPredict Output](#)
- [Examples: How to Use TD\\_XGBoostPredict](#)

## TD\_XGBoostPredict Syntax

### For Regression

```
TD_XGBoostPredict(
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY ]
    ON { table | view | (query) } AS ModelTable DIMENSION
    USING
        IDColumn ({'id_column' | id_column_range })
        [ NumParallelTrees (num_trees) ]
        [ NumBoostRounds (iterations) ]
        [ ModelType ('regression') ]
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
)
```

### For Classification

```
TD_XGBoostPredict(
    ON { table | view | (query) } AS InputTable [ PARTITION BY ANY ]
    ON { table | view | (query) } AS ModelTable DIMENSION
    USING
        IDColumn ({'id_column' | id_column_range })
        [ NumParallelTrees (num_trees) ]
        [ NumBoostRounds (iterations) ]
        [ ModelType ('classification') ]
        [ OutputProb ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
        [ Responses ('response')[,...] )
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_XGBoostPredict

**ON clause**

Specifies the table name, view name or query as an InputTable and ModelTable.

**IDColumn**

Specifies the column that contains a unique identifier for each test point in the test set.

**Note:**

Input column names with double quotation marks are not allowed for this function.

## Optional Syntax Elements for TD\_XGBoostPredict

**NumParallelTrees**

Specifies how many boosted trees to use to make predictions.

A combination of both task\_Index and tree\_num in the model table determines the ampID and number of trees generated by that AMP. As the model table is ordered with these two arguments, the number of boosted trees that are loaded are based on this order.

For example, if there are two AMPS on the system and AMP 1 (task\_index - 0) generates three boosted trees (tree\_num:1,2,3) while amp 2 (task\_index -1) generate two boosted trees (tree\_num: 1,2). Then, NumBoostedTree(4) loads three boosted trees from AMP1 (task\_index - 0) and one boosted tree from AMP2 (task\_index - 1).

As one boosted tree is skipped altogether from loading it in memory and making predictions, this results in a faster elapsed time for queries compared to loading all trees in memory. However, this can also lead to loss in prediction accuracy. In addition, any unique tree is determined by task\_index, tree\_id and iterNum in the model table.

**Note:**

You can still use the previous argument name NumBoostedTrees.

Default: 1000

**NumBoostRounds**

Specifies how many iterations to load for each boosted tree to make predictions.

For example, AMP1 (task\_index:0) generates three boosted trees (tree\_num: 1,2,3) with each tree having four iterations (iter:1,2,3,4). There are 12 trees in total. IterNum(2) only loads two iterations per boosted tree, that is, only six trees are loaded for this example.

As trees are skipped from loading it in memory and making predictions over them, this results in a faster elapsed time for queries compared to loading all trees in memory. However, this can also lead to loss in prediction accuracy.

**Note:**

You can still use the previous argument name IterNum.

Default: 10

**ModelType**

For classification, output the prediction column as integers. These integral values represent different categories, and so are better observed as an integer column. To make the output schema for prediction column as an integer, set ModelType as Classification.

Default: the prediction column is output as Real Valued Column.

**OutputProb**

Specifies whether to output the probability for each response.

**Note:**

- If OutputProb is true and responses are not provided, output the probability of the predicted class.
- The OutputProb argument works only with a classification.

Default: false

**Responses**

Specifies the classes for which to output probabilities.

**Note:**

- If OutputProb is true and responses are not provided, output the probability of the predicted class.
- The Responses argument works only with a classification.

**Accumulate**

Specifies the input columns names to copy to the output table.

**Note:**

- The processing time is controlled by (proportional to):
  - The number of boosted trees used for prediction from the model (controlled by NumParallelTrees).
  - The number of iterations (sub-trees) used for prediction from the model in each boosted tree (controlled by IterNum).

A careful choice of these parameters can be used to control the processing time. When the boosted trees size grows more than what can fit in memory, the trees are cached in a local spool space, which may impact the performance of the function compared to the case when all trees fit in memory.

**TD\_XGBoostPredict Input****InputTable Schema**

Column Name	Data Type	Description
ID_Column	Any	Unique test point identifier. Cannot be NULL.
target_column(s)	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	Column appears once for each specified target_column. Predictor variable. Cannot be NULL.
accumulate_column(s)	Any	Column appears once for each specified accumulate_column. Column to copy to output table.

**Model Table Schema**

Column Name	Data Type	Description
task_index	SMALLINT	Identifier of AMP that produced a boosting tree.
tree_num	SMALLINT	Identifier of boosted tree. Number of unique tree_id values depends on NumBoostedTrees syntax element value and number of AMPS.
Iter	SMALLINT	Iteration (boosting round) number.
class_num	SMALLINT	Index of class column to predict. It only appears in classification. For LossFunction ('softmax'), the default: Number of unique class_num values is number of class labels in data set. For K class

Column Name	Data Type	Description
		labels: class_num values are the integers in range [0, K-1]. For LossFunction('binomial'): There is only one class_num value.
tree_order	SMALLINT	Identifier of a complete JSON order of the regression_tree/classification_tree column.
regression_tree / classification_tree	VARCHAR 32000	JSON representation of decision tree. For JSON types that can appear in the representation, see the following table.

## JSON Types in JSON Representation of Decision Tree

JSON Type	Description
id_	Node identifier.
sum_	Appears only for regression trees. Sum of values of response variable at node identified by id.
sumSq_	Appears only for regression trees. Sum of squared values of response variable at node identified by id.
responseCounts_	Appears only for classification trees. The number of observations in each class at a node, identified by id.
size_	The total number of observations at a node, identified by id.
maxDepth_	Maximum possible depth of the tree, starting from node identified by id. For root node, the value is max_depth; for leaf nodes, 0; for other nodes, maximum possible depth of the tree, starting from that node.
split_	Start of JSON item describing a split at node identified by id.
splitValue_	The attribute value used for splitting a tree node.
score_	Gini score of the node identified by id.
attr_	Attribute (predictor) on which algorithm split at node identified by id.
type_	Type of tree and split. Possible values: • REGRESSION_NUMERIC_SPLIT
leftNodeSize_	The number of observations assigned to the left node in the split.
rightNodeSize_	The number of observations assigned to the right node in the split.
leftChild_	Start of JSON item describing left child of a node, identified by id.
rightChild_	Start of JSON item describing right child of a node, identified by id.
nodeType_	Type of a node identified by id. Possible values: • REGRESSION_NODE • REGRESSION_LEAF

## JSON Types in JSON Representation of Region Prediction

JSON Type	Description
id	Region identifier.
value	The value in the region.

## TD\_XGBoostPredict Output

### Output Table Schema

Column	Data Type	Description
id_column	Same as in input table	Column copied from input table. Unique row identifier.
prediction	Double Precision /Integer	Predicted test point value or predicted class, determined by model. Datatype double precision by default. If ModelType is set to classification, the dataType is set to integer.
confidence_lower	Double Precision	Appears with OutputProb ('false'). Lower bound of confidence interval. For classification trees, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.
confidence_upper	Double Precision	Appears with OutputProb ('false'). Upper bound of confidence interval. For classification trees, confidence_lower and confidence_upper have the same value, which is the probability of the predicted class.
prob	Double Precision	Column appears only with OutputProb ('true') and without the Responses syntax element. Probability that observation belongs to class prediction.
prob_response	Double Precision	Column appears only with OutputProb ('true') and the Responses syntax element. Appears once for each specified response. Probability that observation belongs to category response.
accumulate_column	Same as in input table	Column copied from input table.

## Examples: How to Use TD\_XGBoostPredict

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [XGBoostPredict for Classification](#)
- [XGBoostPredict for Regression](#)

## XGBoostPredict for Classification

This example shows how to use XGBoost for classification: first use TD\_XGBoost to train a classification model, and then use TD\_XGBoostPredict to test the model.

The following sections show the input table, SQL query, and output model table when training a classification model using TD\_XGBoost.

### Input Table: xgboostTrain

	<b>id</b>	<b>rent</b>	<b>lotsize</b>	<b>bedrooms</b>	<b>bathrms</b>	<b>driveway</b>	<b>homestyle</b>
0	1	3200	900	2	1	0	1
1	1	4000	1400	3	2	1	2
2	1	2000	750	1	1	0	1
3	1	10000	3800	4	2	1	3
4	1	7500	2750	3	2	0	2
5	1	15000	2750	5	3	1	3

### SQL Function Call: TD\_XGBoost

```
create table xgboost_classification_model AS(
    SELECT * FROM TD_XGBoost (
        ON xgboostTrain AS inputtable partition by ANY
        USING
        ResponseColumn('homestyle')
        InputColumns('rent','lotsize','bedrooms','bathrms','driveway')
        MaxDepth(3)
        NumBoostedTrees(1)
        ModelType('classification')
        Seed(1)
        IterNum(2)
    ) AS dt
)with data;
```

### Output Model Table: xgboost\_classification\_model

	<b>task_index</b>	<b>tree_num</b>	<b>iter</b>	<b>class_num</b>	<b>tree_order</b>	<b>classification_tree</b>
0	0	1	1	2	0	{"id_":1,"sum_":0.000000,"sumSq_":1.333333,"size_":6,"maxDepth_":3,"nodeType_":"REGRESSION NODE","split_":{"splitValue_":8750.000000,"attr_":}

task_index	tree_num	iter	class_num	tree_order	classification_tree
					{"rent","type_":"REGRESSION_NUMERIC_SPLIT","score_":1.333333,"scoreImprove_":1.333333,"leftNodeSize_":4,"rightNodeSize_":2},"leftChild_":{"id_":2,"sum_-":-1.333333,"sumSq_":0.444444,"size_":4,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5750.000000,"attr_":"rent","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":3,"rightNodeSize_":1},"leftChild_":{"id_":4,"sum_-":-1.000000,"sumSq_":0.333333,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":3600.000000,"attr_":"rent","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":2,"rightNodeSize_":1},"leftChild_":{"id_":8,"sum_-":-0.666667,"sumSq_":0.222222,"size_":2,"maxDepth_":0,"value_-":-0.333333,"nodeType_":"REGRESSION_LEAF","prediction_-":0.000000}, "rightChild_":{"id_":9,"sum_-":-0.333333,"sumSq_":0.111111,"size_":1,"maxDepth_":0,"value_-":-0.333333,"nodeType_":"REGRESSION_LEAF","prediction_-":0.000000}}, "rightChild_":{"id_":5,"sum_-":-0.333333,"sumSq_":0.111111,"size_":1,"maxDepth_":0,"value_-":-0.333333,"nodeType_":"REGRESSION_LEAF","prediction_-":0.000000}}, "rightChild_":{"id_":3,"sum_-":1.333333,"sumSq_":0.888889,"size_":2,"maxDepth_":0,"value_-":0.666667,"nodeType_":"REGRESSION_LEAF","prediction_-":0.000001}}
1	0	1	2	1	0
2	0	1	2	2	0

task_index	tree_num	iter	class_num	tree_order	classification_tree
					<pre> NODE", "split_": {"splitValue_": 8750.000000, "attr_": "rent", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 1.333332, "scoreImprove_": 1.333332, "leftNodeSize_": 4, "rightNodeSize_": 2}, "leftChild_": {"id_": 2, "sum_": -1.333333, "sumSq_": 0.444444, "size_": 4, "maxDepth_": 2, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 3600.000000, "attr_": "rent", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 0.000000, "scoreImprove_": 0.000000, "leftNodeSize_": 2, "rightNodeSize_": 2}, "leftChild_": {"id_": 4, "sum_": -0.666666, "sumSq_": 0.222222, "size_": 2, "maxDepth_": 0, "value_": -0.333333, "nodeType_": "REGRESSION_LEAF", "prediction_": -0.000000}, "rightChild_": {"id_": 5, "sum_": -0.666666, "sumSq_": 0.222222, "size_": 2, "maxDepth_": 0, "value_": -0.333333, "nodeType_": "REGRESSION_LEAF", "prediction_": -0.000000}, "rightChild_": {"id_": 3, "sum_": 1.333333, "sumSq_": 0.888888, "size_": 2, "maxDepth_": 0, "value_": 0.666666, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.000001}} </pre>
3	0	-1	-1	-1	{"lossType": "SOFTMAX", "numBoostedTrees": 1, "iterNum": 2, "classMapping": {"1": 0, "2": 1, "3": 2}}
4	0	1	2	0	<pre> {"id_": 1, "sum_": -0.000000, "sumSq_": 1.333332, "size_": 6, "maxDepth_": 3, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 3600.000000, "attr_": "rent", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 1.333332, "scoreImprove_": 1.333332, "leftNodeSize_": 2, "rightNodeSize_": 4}, "leftChild_": {"id_": 2, "sum_": 1.333333, "sumSq_": 0.888888, "size_": 2, "maxDepth_": 0, "value_": 0.666666, "nodeType_": "REGRESSION_LEAF", "prediction_": 0.000001}, "rightChild_": {"id_": 3, "sum_": -1.333333, "sumSq_": 0.444444, "size_": 4, "maxDepth_": 2, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 8750.000000, "attr_": "rent", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 0.000000, "scoreImprove_": 0.000000, "leftNodeSize_": 2, "rightNodeSize_": 2}, "leftChild_": {"id_": 6, "sum_": -0.666666, "sumSq_": 0.222222, "size_": 2, "maxDepth_": 0, "value_": -0.333333, "nodeType_": "REGRESSION_LEAF", "prediction_": -0.000000}, "rightChild_": {"id_": 7, "sum_": -0.666666, "sumSq_": 0.222222, "size_": 2, "maxDepth_": 0, "value_": -0.333333, "nodeType_": "REGRESSION_LEAF", "prediction_": -0.000000}}} </pre>
5	0	1	1	0	<pre> {"id_": 1, "sum_": 0.000000, "sumSq_": 1.333333, "size_": 6, "maxDepth_": 3, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 3600.000000, "attr_": "rent", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 0.333333, "scoreImprove_": 0.333333, "leftNodeSize_": 2, "rightNodeSize_": 4}, "leftChild_": {"id_": 2, "sum_": -0.666667, "sumSq_": 0.222222, </pre>

task_index	tree_num	iter	class_num	tree_order	classification_tree
					{"size_":2,"maxDepth_":0,"value_-":-0.333333,"nodeType_":"REGRESSION_LEAF","prediction_-":-0.000000}, {"rightChild_":{"id_":3,"sum_":0.666667,"sumSq_":1.111111,"size_":4,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":8750.000000,"attr_":"rent","type_":"REGRESSION_NUMERIC_SPLIT","score_":1.000000,"scoreImprove_":1.000000,"leftNodeSize_":2,"rightNodeSize_":2}, "leftChild_":{"id_":6,"sum_":1.333333,"sumSq_":0.888889,"size_":2,"maxDepth_":0,"value_":0.666667,"nodeType_":"REGRESSION_LEAF","prediction_":0.000001}, "rightChild_":{"id_":7,"sum_-":-0.666667,"sumSq_-":0.222222,"size_-":2,"maxDepth_-":0,"value_-":-0.333333,"nodeType_-":"REGRESSION_LEAF","prediction_-":-0.000000}}}
6	0	1	1	0	{"id_":1,"sum_":0.000000,"sumSq_":1.333333,"size_-":6,"maxDepth_-":3,"nodeType_-":"REGRESSION_NODE","split_":{"splitValue_":3600.000000,"attr_":"rent","type_":"REGRESSION_NUMERIC_SPLIT","score_-":1.333333,"scoreImprove_-":1.333333,"leftNodeSize_-":2,"rightNodeSize_-":4}, "leftChild_":{"id_-":2,"sum_-":1.333333,"sumSq_-":0.888889,"size_-":2,"maxDepth_-":0,"value_-":0.666667,"nodeType_-":"REGRESSION_LEAF","prediction_-":0.000001}, "rightChild_":{"id_-":3,"sum_-":-1.333333,"sumSq_-":0.444444,"size_-":4,"maxDepth_-":2,"nodeType_-":"REGRESSION_NODE","split_":{"splitValue_-":12500.000000,"attr_":"rent","type_-":"REGRESSION_NUMERIC_SPLIT","score_-":0.000000,"scoreImprove_-":0.000000,"leftNodeSize_-":3,"rightNodeSize_-":1}, "leftChild_":{"id_-":6,"sum_-":-1.000000,"sumSq_-":0.333333,"size_-":3,"maxDepth_-":1,"nodeType_-":"REGRESSION_NODE","split_":{"splitValue_-":8750.000000,"attr_":"rent","type_-":"REGRESSION_NUMERIC_SPLIT","score_-":0.000000,"scoreImprove_-":0.000000,"leftNodeSize_-":2,"rightNodeSize_-":1}, "leftChild_":{"id_-":12,"sum_-":-0.666667,"sumSq_-":0.222222,"size_-":2,"maxDepth_-":0,"value_-":-0.333333,"nodeType_-":"REGRESSION_LEAF","prediction_-":-0.000000}, "rightChild_":{"id_-":7,"sum_-":-0.333333,"sumSq_-":0.111111,"size_-":1,"maxDepth_-":0,"value_-":-0.333333,"nodeType_-":"REGRESSION_LEAF","prediction_-":-0.000000}}}}

The following sections show the input table, SQL query and output table when testing the classification model using TD\_XGBoostPredict.

**Input Table: xgboostTest**

	<b>id</b>	<b>rent</b>	<b>lotsize</b>	<b>bedrooms</b>	<b>bathrooms</b>	<b>driveway</b>
0	3	1250	550	1	1	1
1	4	5500	3550	3	1	0
2	1	12500	4000	3	2	1
3	2	18000	5550	8	6	0

**SQL Function Call: TD\_XGBoostPredict Query**

```
SELECT * FROM TD_XGBoostPredict(
    ON test_data AS inputtable partition by ANY
    ON xgboost_classification_model AS modeltable dimension
    USING
        IdColumn('id')
        ModelType('Classification')
        OutputProb('t')
        Responses('1','2','3')
) AS dt;
```

**Output Table**

	<b>id</b>	<b>Prediction</b>	<b>Prob_1</b>	<b>Prob_2</b>	<b>Prob_3</b>
0	3	1	0.33333377777792600	0.3333331111110370	0.3333331111110370
1	4	2	0.3333331111110370	0.33333377777792600	0.3333331111110370
2	1	3	0.3333331111110370	0.3333331111110370	0.33333377777792600
3	2	3	0.3333331111110370	0.3333331111110370	0.33333377777792600

**XGBoostPredict for Regression**

This example shows how to use XGBoost for regression: first use TD\_XGBoost to train a regression model, and then use TD\_XGBoostPredict to test the model.

The following section shows the Output (xgboost\_results) table that was generated as described in [TD\\_XGBoost for Regression](#). It serves as the input regression model table for TD\_XGBoostPredict.

**Input Model Table: xgboost\_res**

This is the same model that was generated using TD\_XGBoost (regression option) on some input data (with the same columns).

task_index	tree_num	iter	tree_order	regression_tree	
0	3	1	2	0	{"id_":1,"sum_":0.000000,"sumSq_":0.000000,"size_":1,"maxDepth_":0,"value_":0.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.000000}
1	0	1	3	0	{"id_":1,"sum_":-0.007523,"sumSq_":391.724642,"size_":5,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.000000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":114.766845,"scoreImprove_":114.766845,"leftNodeSize_":1,"rightNodeSize_":4},"leftChild_":{"id_":2,"sum_":-9.583440,"sumSq_":91.842314,"size_":1,"maxDepth_":0,"value_":-9.583440,"nodeType_":"REGRESSION_LEAF","prediction_":-0.008268}),"rightChild_":{"id_":3,"sum_":9.575916,"sumSq_":299.882329,"size_":4,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":3.500000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":224.467449,"scoreImprove_":224.467449,"leftNodeSize_":1,"rightNodeSize_":3},"leftChild_":{"id_":6,"sum_":15.368978,"sumSq_":236.205471,"size_":1,"maxDepth_":0,"value_":15.368978,"nodeType_":"REGRESSION_LEAF","prediction_":0.015475}),"rightChild_":{"id_":7,"sum_":-5.793061,"sumSq_":63.676858,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":11.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":47.990338,"scoreImprove_":47.990338,"leftNodeSize_":2,"rightNodeSize_":1}),"leftChild_":{"id_":14,"sum_":1.794244,"sumSq_":6.109656,"size_":2,"maxDepth_":0,"value_":0.897122,"nodeType_":"REGRESSION_LEAF","prediction_":0.001436}),"rightChild_":{"id_":15,"sum_":-7.587305,"sumSq_":57.567202,"size_":1,"maxDepth_":0,"value_":-7.587305,"nodeType_":"REGRESSION_LEAF","prediction_":-0.006339}}}}
2	3	1	3	0	{"id_":1,"sum_":0.000000,"sumSq_":0.000000,"size_":1,"maxDepth_":0,"value_":0.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.000000}
3	0	-1	-1	-1	{"lossType":"MSE","numBoostedTrees":4,"iterNum":3,"avgResponses":20.600000}
4	3	1	1	0	{"id_":1,"sum_":0.000000,"sumSq_":0.000000,"size_":1,"maxDepth_":0,"value_":0.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.000000}
5	0	1	2	0	{"id_":1,"sum_":-0.003769,"sumSq_":392.461463,"size_":5,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.000000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":114.983184,"scoreImprove_":114.983184,"leftNodeSize_":1,"rightNodeSize_":4}),"leftChild_":{"id_":2,"sum_":-9.591716,"sumSq_":92.001008,"size_":1,"maxDepth_":0,"value_":-9.591716,"nodeType_":"REGRESSION_LEAF","prediction_":-0.008276}),"rightChild_":{"id_":3,"sum_":9.587947,"sumSq_":300.460456,"size_":4,"maxDepth_":2,"nodeType_":}

task_index	tree_num	iter	tree_order	regression_tree
				<pre>"REGRESSION_NODE","split_":{"splitValue_":3.500000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":224.899861,"scoreImprove_":224.899861,"leftNodeSize_":1,"rightNodeSize_":3},"leftChild_":{"id_":6,"sum_":15.384477,"sumSq_":236.682120,"size_":1,"maxDepth_":0,"value_":15.384477,"nodeType_":"REGRESSION_LEAF","prediction_":0.015499},"rightChild_":{"id_":7,"sum_":-5.796530,"sumSq_":63.778336,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":11.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":48.078415,"scoreImprove_":48.078415,"leftNodeSize_":2,"rightNodeSize_":1},"leftChild_":{"id_":14,"sum_":1.797120,"sumSq_":6.114820,"size_":2,"maxDepth_":0,"value_":0.898560,"nodeType_":"REGRESSION_LEAF","prediction_":0.001438},"rightChild_":{"id_":15,"sum_":-7.593650,"sumSq_":57.663516,"size_":1,"maxDepth_":0,"value_":-7.593650,"nodeType_":"REGRESSION_LEAF","prediction_-0.006344}}}}</pre>
6	1	1	3	<pre>{"id_":1,"sum_":0.000000,"sumSq_":0.000000,"size_":1,"maxDepth_":0,"value_":0.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.000000}</pre>
7	0	1	1	<pre>{"id_":1,"sum_":-0.000000,"sumSq_":393.200000,"size_":5,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.000000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":115.200000,"scoreImprove_":115.200000,"leftNodeSize_":1,"rightNodeSize_":4},"leftChild_":{"id_":2,"sum_-9.600000,"sumSq_":92.160000,"size_":1,"maxDepth_":0,"value_-9.600000,"nodeType_":"REGRESSION_LEAF","prediction_-0.008284},"rightChild_":{"id_":3,"sum_":9.600000,"sumSq_":301.040000,"size_":4,"maxDepth_":2,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":3.500000,"attr_":"col_2","type_":"REGRESSION_NUMERIC_SPLIT","score_":225.333333,"scoreImprove_":225.333333,"leftNodeSize_":1,"rightNodeSize_":3},"leftChild_":{"id_":6,"sum_":15.400000,"sumSq_":237.160000,"size_":1,"maxDepth_":0,"value_":15.400000,"nodeType_":"REGRESSION_LEAF","prediction_":0.015523},"rightChild_":{"id_":7,"sum_":-5.800000,"sumSq_":63.880000,"size_":3,"maxDepth_":1,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":11.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":48.166667,"scoreImprove_":48.166667,"leftNodeSize_":2,"rightNodeSize_":1},"leftChild_":{"id_":14,"sum_":1.800000,"sumSq_":6.120000,"size_":2,"maxDepth_":0,"value_":0.900000,"nodeType_":"REGRESSION_LEAF","prediction_":0.001440},"rightChild_":{"id_":15,"sum_":-7.600000,"sumSq_":57.760000,"size_":1,"maxDepth_":0,"value_":-7.600000,"nodeType_":"REGRESSION_LEAF","prediction_-0.006350}}}}</pre>

task_index	tree_num	iter	tree_order	regression_tree	
8	1	1	2	0	{"id_":1,"sum_":0.000000,"sumSq_":0.000000,"size_":1,"maxDepth_":0,"value_":0.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.000000}
9	1	1	1	0	{"id_":1,"sum_":0.000000,"sumSq_":0.000000,"size_":1,"maxDepth_":0,"value_":0.000000,"nodeType_":"REGRESSION_LEAF","prediction_":0.000000}
10	2	1	1	0	{"id_":1,"sum_":0.000000,"sumSq_":220.500000,"size_":2,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":17.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":220.500000,"scoreImprove_":220.500000,"leftNodeSize_":1,"rightNodeSize_":1},"leftChild_":{"id_":2,"sum_":-10.500000,"sumSq_":110.250000,"size_":1,"maxDepth_":0,"value_":-10.500000,"nodeType_":"REGRESSION_LEAF","prediction_":-0.009223}, "rightChild_":{"id_":3,"sum_":10.500000,"sumSq_":110.250000,"size_":1,"maxDepth_":0,"value_":10.500000,"nodeType_":"REGRESSION_LEAF","prediction_":0.009223}}
11	2	1	3	0	{"id_":1,"sum_":0.000000,"sumSq_":219.726348,"size_":2,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":17.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":219.726348,"scoreImprove_":219.726348,"leftNodeSize_":1,"rightNodeSize_":1}, "leftChild_":{"id_":2,"sum_":-10.481564,"sumSq_":109.863174,"size_":1,"maxDepth_":0,"value_":-10.481564,"nodeType_":"REGRESSION_LEAF","prediction_":-0.009203}, "rightChild_":{"id_":3,"sum_":10.481564,"sumSq_":109.863174,"size_":1,"maxDepth_":0,"value_":10.481564,"nodeType_":"REGRESSION_LEAF","prediction_":0.009203}}
12	2	1	2	0	{"id_":1,"sum_":0.000000,"sumSq_":220.112797,"size_":2,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":17.500000,"attr_":"col_1","type_":"REGRESSION_NUMERIC_SPLIT","score_":220.112797,"scoreImprove_":220.112797,"leftNodeSize_":1,"rightNodeSize_":1}, "leftChild_":{"id_":2,"sum_":-10.490777,"sumSq_":110.056399,"size_":1,"maxDepth_":0,"value_":-10.490777,"nodeType_":"REGRESSION_LEAF","prediction_":-0.009213}, "rightChild_":{"id_":3,"sum_":10.490777,"sumSq_":110.056399,"size_":1,"maxDepth_":0,"value_":10.490777,"nodeType_":"REGRESSION_LEAF","prediction_":0.009213}}

The following sections show the input table, SQL query and output table when testing the regression model using TD\_XGBoostPredict.

#### Input Table: xgboost\_reg\_test

id		medv	col_1	col_2	col_3
0	3	17.9	4	5.45	0.97022

	<b>id</b>	<b>medv</b>	<b>col_1</b>	<b>col_2</b>	<b>col_3</b>
1	1	23.9	18	4.14	0.54054
2	2	14	29	9.97	0.26351

### SQL Function Call: TD\_XGBoostPredict Query

```
SELECT * FROM TD_XGBoostPredict(
    ON xgboost_reg_test AS inputtable partition by ANY
    ON xgboost_res AS modeltable dimension
    USING
        IdColumn('id')
        ModelType('Regression')
) AS dt;
```

### Output Table

	<b>id</b>	<b>Prediction</b>	<b>Confidence_Lower</b>	<b>Confidence_upper</b>
0	3	20.592225	20.58509094229030	20.599359057709700
1	1	20.602868666666700	20.58788390759070	20.61785342574260
2	2	20.602868666666700	20.58788390759070	20.61785342574260

# Model Evaluation Functions

- [TD\\_SHAP](#)
- [TD\\_Silhouette](#)
- [TD\\_ClassificationEvaluator](#)
- [TD\\_RegressionEvaluator](#)
- [TD\\_ROC](#)
- [TD\\_TrainTestSplit](#)

## TD\_SHAP

SHapley Additive exPlanations (SHAP) algorithm is a method to explain individual predictions (feature contributions) for a machine learning model based on the cooperative game theory. SHAP computes the contribution of each feature in a prediction as an average marginal contribution of the feature value across all possible coalitions. TD\_SHAP also computes mean absolute contribution of each feature as global explanation (using OUT clause) which can be used as a measure of feature importance.

TD\_SHAP supports the regression and classification models of the following functions:

- TD\_GLM
- TD\_DecisionForest
- TD\_XGBoost

For Tree-based methods, SHAP computation is quite intensive and therefore should be run on a small subset of prediction dataset. TD\_SHAP needs an input table and a model table to output the feature contribution based on Shapley values.

The following is an example of how to use TD\_SHAP:

1. Prepare training dataset to be used for building a classification or regression model.
2. Generate model using the training dataset. The supported functions are: TD\_GLM, TD\_DecisionForest, and TD\_XGBoost.
3. Prepare test dataset to be used for prediction/evaluating the model for SHAP contributions.
4. Generate SHAP contributions on test dataset using TD\_SHAP function.

## Function Information

- [TD\\_SHAP Syntax](#)
- [Required Syntax Elements for TD\\_SHAP](#)
- [Optional Syntax Elements for TD\\_SHAP](#)
- [TD\\_SHAP Input](#)
- [TD\\_SHAP Output](#)
- [Example: How to Use TD\\_SHAP](#)

## TD\_SHAP Syntax

```

TD_SHAP(
    ON { table | view | (query) } AS InputTable
    ON { table | view | (query) } AS ModelTable DIMENSION
    OUT [PERMANENT | VOLATILE] Table GlobalExplanation (outtable)
USING
    IDColumn ('row_id_column')
    TrainingFunction({'td_glm' | 'td_decisionforest' | 'td_xgboost'})
    InputColumns({<input column> | <input column range>} [,,...])
    [ ModelType({'regression' | 'classification'}) ]
    [ Detailed({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ Accumulate({<column> | <column range>} [,,...]) ]
    [ NumParallelTrees(<NumberofTrees>) ]
    [ NumBoostRounds(<IterationNumber>) ]
)

```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_SHAP

**ON clause**

Specifies the table name, view name, or query as an InputTable and ModelTable.

**IdColumn**

Specifies the column to identify the sample of input table.

**Training Function**

Specifies the training function for the model.

**Note:**

The number of trees and tree size in the model controls the processing time for TD\_DecisionForest and TD\_XGBoost models. When the model size grows more than what can fit in memory, the trees are cached in a local spool space, which may impact the performance of the function compared to the case when all trees fit in memory. Because the SHAP algorithm parses through all branches of a tree and its execution time is heavily impacted by the model size, it is recommended to use it for explainability of a small dataset.

**InputColumns**

Specifies the input columns (features) for which Shapley values must be calculated.

**Note:**

The input column names and case must exactly match with the column names used to train the model.

## Optional Syntax Elements for TD\_SHAP

**ModelType**

Specifies the training model type.

**Note:**

Generate the model table from the TD\_DecisionForest /TD\_GLM/ TD\_XGBoost function.

Default: Regression

**Detailed**

Specifies whether to output detailed shap information for each tree in the model for TD\_DecisionForest and TD\_XGBoost.

**Note:**

TreeNum with value as "FINAL" in the output represents the records having aggregated shap values for corresponding tree.

Default: 'false'

**Accumulate**

Specifies the names of the input columns to copy to the output table.

**NumParallelTrees**

Specify how many boosted trees should be used to make predictions.

Default: 1000

**NumBoostRounds**

Specifies the number of iterations.

The iterations must be an INTEGER in the range [1, 100000].

**Note:**

NumBoostRounds argument is only allowed when the trainingFunction is TD\_XGBoost.

Default: 3

**TD\_SHAP Input**

TD\_SHAP function has two input tables: InputTable and ModelTable.

Table	Description
InputTable	The table that contains sample data for which to predict feature contribution.
ModelTable	Table with same schema as OutputTable of TD_DecisionForest / TD_XGBoost function for trees or with same schema as OutputTable of TD_GLM function for linear regression.

**InputTable Schema**

Column	Data Type	Description
accumulate_columns	Any	Column appears once for each specified accumulate_column. Column to copy to output table.
input_column	INTEGER, BIGINT, SMALLINT, BYTEINT, FLOAT, DECIMAL, NUMBER	Column appears once for each specified input_column. Predictor variable. If the Detailed argument is set to true, two more columns are appended to the output schema. This restricts the number of columns supported by maximum number of input columns (2048) supported by Teradata - 2 (2046).
id	Any	Identifier for each row of input data.

## ModelTable Schema

The model table schema is the same as OutputTable for the following training functions:

- TD\_GLM. See [TD\\_GLM Output](#).
- TD\_DecisionForest. See [TD\\_DecisionForest Output](#).
- TD\_XGBoost. See [TD\\_XGBoost Output](#).

## TD\_SHAP Output

The output table has a set of predictions for each test point.

### OutputTable Schema

Column	Data Type	Description
id	INTEGER	Unique id of each input record.
accumulate_columns	Any	Column copied from InputTable.
input_columns	DOUBLE PRECISION	Calculated shap value determined by model
label	INTEGER	The label specified for the class (for classification models only) corresponding to Responses argument.
tree_num	VARCHAR	Concatenated task Index and tree num from model (for tree-based models only when Detailed('true') is specified).
iter_num	INTEGER	The iter num of a tree (for TD_XGBoost model only when Detailed('true') is specified).

The secondary output table contains mean absolute shapley values for each input feature.

### Secondary OutputTable Schema GlobalExplanations

Column Name	Data Type	Description
input_columns	DOUBLE PRECISION	Calculated global shap value determined by model.
label	INTEGER	The label specified for the class (this column is supported ONLY for TD_DecisionForest and TD_XGBoost classification models).

## Example: How to Use TD\_SHAP

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

TD\_SHAP requires an input table and a model table to output the feature contribution based on calculated Shapley values. The Shapley value is calculated as average marginal contribution of a feature value across all possible coalitions.

- [Regression Model Using TrainingFunction with TD\\_GLM](#)
- [Classification Model Using TrainingFunction with TD\\_XGBoost](#)

## Regression Model Using TrainingFunction with TD\_GLM

**Input Table**

<b>id</b>	<b>x</b>	<b>y</b>	<b>z</b>	<b>w</b>
1	106	108	114	10
2	106	380	390	10
3	106	179	340	10
4	106	153	380	10
5	106	166	243	10
6	294	326	328	20
7	4	299	158	50
8	4	299	237	50
9	4	301	193	30
10	4	301	186	30

**Model Table**

<b>attribute</b>	<b>predictor</b>	<b>estimate</b>	<b>value</b>
-13	LocalSGD Iterations	1.000000000000000E 000	?
-12	Nesterov	?	TRUE
-11	Momentum	9.00E-01	?
-10	Learning Rate (Final)	8.41E-03	?
-9	Learning Rate (Initial)	1.00E-02	?
-8	Number of Iterations	2.000000000000000E 000	CONVERGED
-7	Alpha	1.50E-01	Elasticnet
-6	Regularization	2.00E-02	ENABLED
-5	BIC	1.76975548934661E 002	?

attribute	predictor	estimate	value
-4	AIC	1.76657782767942E 002	?
-3	Number of Observations	8.00000000000000E 000	?
-2	MSE	1.43183524428577E 009	?
-1	Loss Function	?	SQUARED_ERROR
0	(Intercept)	-8.37474392156743E 002	?
1	x	-2.98450611997491E 004	?
2	y	-2.31418858908765E 005	?
3	z	-1.96116814935662E 005	?

### TD\_SHAP SQL Call Using TD\_GLM

```
SELECT * FROM TD_SHAP (
    ON exai AS InputTable
    ON GLMModel_table AS ModelTable DIMENSION
    USING
        IDColumn ('id')
        InputColumns('x','y','z')
        trainingFunction('td_glm')
        modelType('regression')
) AS dt;
```

### TD\_SHAP Output Using TD\_GLM

id	td_x_shap	td_y_shap	td_z_shap
1	-6.56591346394479E 005	3.31391805957352E 007	2.80250928543060E 007
2	-6.56591346394479E 005	-2.98067490274490E 007	-2.61031480679366E 007
3	-6.56591346394479E 005	1.67084416132129E 007	-1.62973073211535E 007
4	-6.56591346394479E 005	2.27253319448407E 007	-2.41419799185800E 007
5	-6.56591346394479E 005	1.97168867790268E 007	2.72602372760569E 006
6	-6.26746285194730E 006	-1.73101306463756E 007	-1.39439055419256E 007
7	2.38760489597993E 006	-1.10618214558390E 007	1.93959529971369E 007
8	2.38760489597993E 006	-1.10618214558390E 007	3.90272461721966E 006
9	2.38760489597993E 006	-1.15246591736565E 007	1.25318644743888E 007
10	2.38760489597993E 006	-1.15246591736565E 007	1.39046821789384E 007

## Classification Model Using TrainingFunction with TD\_XGBoost

**Input Table**

id	sepal_length	sepal_width	petal_length	petal_width	species
1	5.10000000000000E000	3.50000000000000E000	1.40000000000000E000	2.00E-01	1
2	4.90000000000000E000	3.00000000000000E000	1.40000000000000E000	2.00E-01	1
3	4.70000000000000E000	3.20000000000000E000	1.30000000000000E000	2.00E-01	1
4	4.60000000000000E000	3.10000000000000E000	1.50000000000000E000	2.00E-01	1
5	5.00000000000000E000	3.60000000000000E000	1.40000000000000E000	2.00E-01	1
6	5.40000000000000E000	3.90000000000000E000	1.70000000000000E000	4.00E-01	1
7	4.60000000000000E000	3.40000000000000E000	1.40000000000000E000	3.00E-01	1
8	5.00000000000000E000	3.40000000000000E000	1.50000000000000E000	2.00E-01	1
9	4.40000000000000E000	2.90000000000000E000	1.40000000000000E000	2.00E-01	1
10	4.90000000000000E000	3.10000000000000E000	1.50000000000000E000	1.00E-01	1
...	...	...	...	...	...

**Model Table**

task_index	tree_num	iter	class_num	tree
0	-1	-1	-1	{"lossType": "SOFTMAX", "numBoostedTrees": 8, "iterNum": 1, "classMapping": {"1": 0, "2": 1, "3": 2}}
0	1	1	2	{"id": 1, "sum": 510.666667, "sumSq": 16047.111111, "size": 22, "maxDepth": 5, "nodeType": "REGRESSION_NODE", "split": {"splitValue": 5.05, "attr": "petal_length", "type": "REGRESSION_NUMERIC_SPLIT", "score": 4193.454545, "scoreImprove": 4193.454545, "leftNodeSize": 16, "rightNodeSize": 6}, "leftChild": {"id": 2, "sum": 506.666667, "sumSq": 16044.444444, "size": 16, "maxDepth": 4, "nodeType": "REGRESSION_NODE", "split": {"splitValue": 1.75, "attr": "petal_length", "type": "REGRESSION_NUMERIC_SPLIT", "score": 4193.454545, "scoreImprove": 4193.454545, "leftNodeSize": 10, "rightNodeSize": 6}}}

task_index	tree_num	iter	class_num	tree
				<pre>"type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":8,"rightNodeSize_":8,"leftChild_":{"id_":4,"sum_":253.333333,"sumSq_":8022.222222,"size_":8,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":4.950000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":2,"rightNodeSize_":6},"leftChild_":{"id_":8,"sum_":63.333333,"sumSq_":2005.555556,"size_":2,"maxDepth_":0,"value_":3}}</pre>
0	1	1	1	<pre>{"id_":1,"sum_":353911255.666666,"sumSq_":25197504836695924736.000000,"size_":22,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":4.600000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":4279076496322816512.000000,"scoreImprove_":4279076496322816512.000000,"leftNodeSize_":1,"rightNodeSize_":21),"leftChild_":{"id_":2,"sum_":-2004945786.333333,"sumSq_":4019807606135788032.000000,"size_":1,"maxDepth_":0,"value_":-2004945786.333333,"nodeType_":"REGRESSION_LEAF"}, "rightChild_":{"id_":3,"sum_":2358857042.000000,"sumSq_":21177697230560141312.000000,"size_":21,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":1.450000,"attr_":"petal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":5972017308980156416.000000,"scoreImprove_":5972017308980156416.000000,"leftNodeSize_":4,"rightNodeSize_":17),"leftChild_":{"id_":6,"sum_":4846800936.666667,"sumSq_":8497445001240113152.000000,"size_":4,"maxDepth_":0}}</pre>
0	1	1	0	<pre>{"id_":1,"sum_":1.666667,"sumSq_":5.444444,"size_":22,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5.150000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":5.318182,"scoreImprove_":5.318182,"leftNodeSize_":9,"rightNodeSize_":13),"leftChild_":{"id_":2,"sum_":6.000000,"sumSq_":4.000000,"size_":9,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":4.950000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":3,"rightNodeSize_":6}, "leftChild_":{"id_":4,"sum_":2.000000,"sumSq_":1.333333,"size_":3,"maxDepth_":0,"value_":0.666667,"nodeType_":"REGRESSION_LEAF"}, "rightChild_":{"id_":5,"sum_":4.000000,"sumSq_":2.666667,"size_":6,"maxDepth_":0,"value_":0.666667,"nodeType_":"REGRESSION_LEAF"}}, "rightChild_":{"id_":3,"sum_":-4.333333,"sumSq_":1.444444,"size_":13,"maxDepth_":0,"value_":-0.333333,"nodeType_":"REGRESSION_LEAF"}}</pre>
1	1	1	0	<pre>{"id_":1,"sum_":0.333333,"sumSq_":3.888889,"size_":17,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.650000,"attr_":"petal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":3.882353,"scoreImprove_":3.882353,"leftNodeSize_":6,"rightNodeSize_":11),"leftChild_":{"id_":2,"sum_":4.000000,"sumSq_":2.666667,"size_":6,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5.300000,"attr_":"sepal_length","type_":0}}}</pre>

task_index	tree_num	iter	class_num	tree
				<pre>": "REGRESSION_NUMERIC_SPLIT", "score_": 0.000000, "scoreImprove_": 0.000000, "leftNodeSize_": 3, "rightNodeSize_": 3}, "leftChild_": {"id_": 4, "sum_": 2.000000, "sumSq_": 1.333333, "size_": 3, "maxDepth_": 0, "value_": 0.666667, "nodeType_": "REGRESSION_ LEAF"}, "rightChild_": {"id_": 5, "sum_": 2.000000, "sumSq_": 1. 333333, "size_": 3, "maxDepth_": 0, "value_": 0.666667, "nodeType_": "REGRESSION_LEAF"}}, "rightChild_": {"id_": 3, "sum_": -3.666667, "sumSq_": 1.222222, "size_": 11, "maxDepth_": 0, "value_": -0.333333, "nodeType_": "REGRESSION_LEAF"}}</pre>
1	1	1	2	<pre>{"id_": 1, "sum_": 352.333333, "sumSq_": 11033.222222, "size_": 17, "maxDepth_": 5, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 4.900000, "attr_": "petal_length", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 3730.941176, "scoreImprove_": 3730.941176, "leftNodeSize_": 11, "rightNodeSize_": 6}, "leftChild_": {"id_": 2, "sum_": 348.333333, "sumSq_": 11030. 555556, "size_": 11, "maxDepth_": 4, "nodeType_": "REGRESSION_ NODE", "split_": {"splitValue_": 5.700000, "attr_": "sepal_length", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 0.000000, "scoreImprove_": 0.000000, "leftNodeSize_": 8, "rightNodeSize_": 3}, "leftChild_": {"id_": 4, "sum_": 253.333333, "sumSq_": 8022. 222222, "size_": 8, "maxDepth_": 3, "nodeType_": "REGRESSION_ NODE", "split_": {"splitValue_": 5.000000, "attr_": "sepal_length", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 0.000000, "scoreImprove_": 0.000000, "leftNodeSize_": 2, "rightNodeSize_": 6}, "leftChild_": {"id_": 8, "sum_": 63.333333, "sumSq_": 2005.555556, "size_": 2, "maxDepth_": 0, "value_": 3}}</pre>
1	1	1	1	<pre>{"id_": 1, "sum_": 1475714351.333333, "sumSq_": 17853048966241581056.000000, "size_": 17, "maxDepth_": 5, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 5. 450000, "attr_": "sepal_length", "type_": "REGRESSION_NUMERIC_ SPLIT", "score_": 3700050645612942848.000000, "scoreImprove_": 3700050645612942848.000000, "leftNodeSize_": 5, "rightNodeSize_": 12}, "leftChild_": {"id_": 2, "sum_": -3179690523.666666, "sumSq_": 10553647499833659392.000000, "size_": 5, "maxDepth_": 4, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 4. 000000, "attr_": "sepal_width", "type_": "REGRESSION_NUMERIC_ SPLIT", "score_": 6987097604131751936.000000, "scoreImprove_": 6987097604131751936.000000, "leftNodeSize_": 4, "rightNodeSize_": 1}, "leftChild_": {"id_": 4, "sum_": -4908002423.333333, "sumSq_": 7566585477304254464.000000, "size_": 4, "maxDepth_": 3, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 4. 600000, "attr_": "sepal_length", "type_": "REGRESSION_NUMERIC_ SPLIT", "score_": 1527216767450153984.000000, "scoreI</pre>
2	1	1	2	<pre>{"id_": 1, "sum_": 416.333333, "sumSq_": 13039.222222, "size_": 20, "maxDepth_": 5, "nodeType_": "REGRESSION_NODE", "split_": {"splitValue_": 1.650000, "attr_": "petal_width", "type_": "REGRESSION_NUMERIC_SPLIT", "score_": 3480.192857, "scoreImprove_": 3480.192857, "leftNodeSize_": 14, "rightNodeSize_": 6}, "leftChild_": {"id_": 2, "sum_": 412.333333, "sumSq_": 13036. 555556, "size_": 14, "maxDepth_": 4, "nodeType_": "REGRESSION_ NODE", "split_": {"splitValue_": 2.250000, "attr_": "sepal_width",</pre>

task_index	tree_num	iter	class_num	tree
				<pre>"type_":"REGRESSION_NUMERIC_SPLIT","score_":892.357143,"scoreImprove_":892.357143,"leftNodeSize_":1,"rightNodeSize_":13,"leftChild_":{"id_":4,"sum_":0.666667,"sumSq_":0.444444,"size_":1,"maxDepth_":0,"value_":0.666667,"nodeType_":"REGRESSION_LEAF"},"rightChild_":{"id_":5,"sum_":411.666667,"sumSq_":13036.111111,"size_":13,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":3.350000,"attr_":"sepal_width","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000}}</pre>
2	1	1	1	<pre>{"id_":1,"sum_":7545668031.333336,"sumSq_":4066936141337179648.000000,"size_":20,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":3.350000,"attr_":"sepal_width","type_":"REGRESSION_NUMERIC_SPLIT","score_":281557116780666272.000000,"scoreImprove_":281557116780666272.000000,"leftNodeSize_":13,"rightNodeSize_":7),"leftChild_":{"id_":2,"sum_":3772834017.666666,"sumSq_":2033468070668589824.000000,"size_":13,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":1.650000,"attr_":"petal_width","type_":"REGRESSION_NUMERIC_SPLIT","score_":357532846705607936.000000,"scoreImprove_":357532846705607936.000000,"leftNodeSize_":9,"rightNodeSize_":4),"leftChild_":{"id_":4,"sum_":1616928867.000000,"sumSq_":871486316000824064.000000,"size_":9,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":4.950000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":331994786226635776.000000,"scoreImprove_":3}}</pre>
...	...	...	...	...
5	1	1	1	<pre>{"id_":1,"sum_":4311810306.000000,"sumSq_":2323963509335531008.000000,"size_":15,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.350000,"attr_":"petal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":474475881982233472.000000,"scoreImprove_":474475881982233472.000000,"leftNodeSize_":5,"rightNodeSize_":10),"leftChild_":{"id_":2,"sum_":2694881438.333333,"sumSq_":1452477193334706944.000000,"size_":5,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5.350000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":4,"rightNodeSize_":1),"leftChild_":{"id_":4,"sum_":2155905150.666667,"sumSq_":1161981754667765504.000000,"size_":4,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5.150000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":3,"rightNodeSize_":1}},"le</pre>
6	1	1	2	<pre>{"id_":1,"sum_":354.333333,"sumSq_":11034.555556,"size_":20,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5.700000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":4756.950000,"scoreImprove_":4756.950000,"leftNodeSize_":11,"rightNodeSize_":9}}</pre>

task_index	tree_num	iter	class_num	tree
				<pre>":9}","leftChild_":{"id_":2,"sum_":348.333333,"sumSq_":11030.555556,"size_":11,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5.350000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":8,"rightNodeSize_":3}),"leftChild_":{"id_":4,"sum_":253.333333,"sumSq_":8022.222222,"size_":8,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":4.750000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNodeSize_":2,"rightNodeSize_":6}),"leftChild_":{"id_":8,"sum_":63.333333,"sumSq_":2005.555556,"size_":2,"maxDepth_":0,"value_":3}</pre>
6	1	1	1	<pre>{"id_":1,"sum_":-1.666667,"sumSq_":3.888889,"size_":20,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.550000,"attr_":"sepal_width","type_":"REGRESSION_NUMERIC_SPLIT","score_":1.488095,"scoreImprove_":1.488095,"leftNodeSize_":6,"rightNodeSize_":14}),"leftChild_":{"id_":2,"sum_":2.000000,"sumSq_":2.000000,"size_":6,"maxDepth_":0,"value_":0.333333,"nodeType_":"REGRESSION_LEAF"},"rightChild_":{"id_":3,"sum_":-3.666667,"sumSq_":1.888889,"size_":14,"maxDepth_":0,"value_":-0.261905,"nodeType_":"REGRESSION_LEAF"}}</pre>
6	1	1	0	<pre>{"id_":1,"sum_":-0.666667,"sumSq_":4.222222,"size_":20,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.450000,"attr_":"petal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":4.200000,"scoreImprove_":4.200000,"leftNodeSize_":6,"rightNodeSize_":14}),"leftChild_":{"id_":2,"sum_":4.000000,"sumSq_":2.666667,"size_":6,"maxDepth_":0,"value_":0.666667,"nodeType_":"REGRESSION_LEAF"},"rightChild_":{"id_":3,"sum_":-4.666667,"sumSq_":1.555556,"size_":14,"maxDepth_":0,"value_":-0.333333,"nodeType_":"REGRESSION_LEAF"}}</pre>
7	1	1	1	<pre>{"id_":1,"sum_":7545668031.333335,"sumSq_":4066936141337179648.000000,"size_":20,"maxDepth_":5,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":1.750000,"attr_":"petal_width","type_":"REGRESSION_NUMERIC_SPLIT","score_":281557116780666656.000000,"scoreImprove_":281557116780666656.000000,"leftNodeSize_":13,"rightNodeSize_":7}),"leftChild_":{"id_":2,"sum_":3772834017.666665,"sumSq_":2033468070668589824.000000,"size_":13,"maxDepth_":4,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":2.900000,"attr_":"petal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":938523722602220800.000000,"scoreImprove_":938523722602220800.000000,"leftNodeSize_":7,"rightNodeSize_":6}),"leftChild_":{"id_":4,"sum_":3772834013.666666,"sumSq_":2033468070668589824.000000,"size_":7,"maxDepth_":3,"nodeType_":"REGRESSION_NODE","split_":{"splitValue_":5.750000,"attr_":"sepal_length","type_":"REGRESSION_NUMERIC_SPLIT","score_":0.000000,"scoreImprove_":0.000000,"leftNo</pre>

task_index	tree_num	iter	class_num	tree
7	1	1	2	{"id":1,"sum":416.333333,"sumSq":13039.222222,"size":20,"maxDepth":5,"nodeType":"REGRESSION_NODE","split":{"splitValue":1.750000,"attr":"petal_width","type":"REGRESSION_NUMERIC_SPLIT","score":4372.550000,"scoreImprove":4372.550000,"leftNodeSize":13,"rightNodeSize":7},"leftChild":{"id":2,"sum":411.666667,"sumSq":13036.111111,"size":13,"maxDepth":4,"nodeType":"REGRESSION_NODE","split":{"splitValue":3.250000,"attr":"sepal_width","type":"REGRESSION_NUMERIC_SPLIT","score":0.000000,"scoreImprove":0.000000,"leftNodeSize":8,"rightNodeSize":5},"leftChild":{"id":4,"sum":253.333333,"sumSq":8022.222222,"size":8,"maxDepth":3,"nodeType":"REGRESSION_NODE","split":{"splitValue":5.150000,"attr":"sepal_length","type":"REGRESSION_NUMERIC_SPLIT","score":0.000000,"scoreImprove":0.000000,"leftNodeSize":2,"rightNodeSize":6},"leftChild":{"id":8,"sum":63.333333,"sumSq":2005.555556,"size":2,"maxDepth":0,"value":31}}
7	1	1	0	{"id":1,"sum":0.333333,"sumSq":4.555556,"size":20,"maxDepth":5,"nodeType":"REGRESSION_NODE","split":{"splitValue":2.900000,"attr":"petal_length","type":"REGRESSION_NUMERIC_SPLIT","score":4.550000,"scoreImprove":4.550000,"leftNodeSize":7,"rightNodeSize":13},"leftChild":{"id":2,"sum":4.666667,"sumSq":3.111111,"size":7,"maxDepth":4,"nodeType":"REGRESSION_NODE","split":{"splitValue":5.000000,"attr":"sepal_length","type":"REGRESSION_NUMERIC_SPLIT","score":0.000000,"scoreImprove":0.000000,"leftNodeSize":3,"rightNodeSize":4},"leftChild":{"id":4,"sum":2.000000,"sumSq":1.333333,"size":3,"maxDepth":0,"value":0.666667,"nodeType":"REGRESSION_LEAF"}, "rightChild":{"id":5,"sum":2.666667,"sumSq":1.777778,"size":4,"maxDepth":0,"value":0.666667,"nodeType":"REGRESSION_LEAF"}}, "rightChild":{"id":3,"sum":-4.333333,"sumSq":1.444444,"size":13,"maxDepth":0,"value":-0.333333,"nodeType":"REGRESSION_LEAF"}}

## TD\_SHAP SQL Call Using TD\_XGBoost

```

DROP TABLE iris_predict;
CREATE MULTISET TABLE iris_predict AS (
  SELECT * FROM TD_SHAP (
    ON iris_input AS InputTable
    ON iris_model AS ModelTable DIMENSION
    OUT TABLE GlobalExplanation(shap_xgb_class_out)
    USING
      idcolumn('id')
      InputColumns ('[1:4]')
      modeltype('classification')
      trainingFunction('td_xgboost')
)
```

```

NumParallelTrees(5)
NumBoostRounds(1)
detailed('1')
) AS dt
) WITH DATA;

```

**shap\_xgb\_class\_out Output Using TD\_XGBoost**

tree_num	IterNum	Label	TD_sepallength_SHAP	TD_sepalwidth_SHAP	TD_petallength_SHAP	TD_petalwidth_SHAP
01	1	0	1.53E-01	0.000000000000000000	0.000000000000000000	0.000000000000000000
01	1	1	1.50480831299513E008	7.46910482102235E007	5.50716619785201E008	1.01145895530632E008
01	1	2	8.81E-16	1.19E-15	1.24000000000000E001	0.00000000000000E000
11	1	0	1.85E-17	0.00000000000000E000	1.50E-01	0.00000000000000E000
11	1	1	3.84996856566540E008	3.14393034807082E007	3.72723775820163E008	9.48018283159568E007
11	1	2	8.99E-16	2.20E-16	1.40411764705883E001	0.00000000000000E000
21	1	0	0.00000000000000E000	0.00000000000000E000	1.50E-01	0.00000000000000E000
21	1	1	7.05217859489031E007	1.09354036635137E008	6.42199824836310E007	8.64033426786501E007
21	1	2	1.39E-15	2.13014285714286E000	0.00000000000000E000	1.26701428571429E001
31	1	0	0.00000000000000E000	0.00000000000000E000	1.48E-01	0.00000000000000E000
31	1	1	1.92E-17	0.00000000000000E000	1.00E-01	1.07E-01

tree_num	IterNum	Label	TD_sepallength_SHAP	TD_sepawidth_SHAP	TD_petal_length_SHAP	TD_petal_width_SHAP
31	1	2	3.05E-15	0.000000000000000000	0.000000000000000000	1.28362962962963E001
41	1	0	1.38E-01	0.000000000000000000	0.000000000000000000	0.000000000000000000
41	1	1	1.01E-01	0.000000000000000000	1.06E-01	0.000000000000000000
41	1	2	1.85E-15	1.31E-15	1.45814814814815E001	0.000000000000000000
FINAL	1	0	5.42E-02	0.000000000000000000	8.97E-02	0.000000000000000000
FINAL	1	1	9.56349356957458E007	2.18928751669318E007	1.59293785385713E008	3.29549267347127E007
FINAL	1	2	1.12E-15	4.26E-01	7.79954842543079E000	5.05536190476190E000

### TD\_SHAP SQL Call Using TD\_XGBoost

```
drop table iris_predict;
CREATE MULTISET TABLE iris_predict AS (
  SELECT * FROM TD_SHAP (
    ON iris_input AS InputTable
    ON iris_model AS ModelTable DIMENSION
    USING
      idcolumn('id')
      InputColumns ('[1:4]')
      modeltype('classification')
      trainingFunction('td_xgboost')
      NumParallelTrees(5)
      NumBoostRounds(1)
      detailed('0')
  ) AS dt
) WITH DATA;
```

TD\_SHAP Output Table Using TD\_XGBoost

Label	Id	TD_sepel_length_SHAP	TD_sepel_width_SHAP	TD_petal_length_SHAP	TD_petal_width_SHAP
0	1	9.12E-02	0.00000000000000E000	1.31E-01	0.00000000000000E000
0	2	9.12E-02	0.00000000000000E000	1.31E-01	0.00000000000000E000
0	3	9.12E-02	0.00000000000000E000	1.31E-01	0.00000000000000E000
0	4	9.12E-02	0.00000000000000E000	1.31E-01	0.00000000000000E000
0	5	9.12E-02	0.00000000000000E000	1.31E-01	0.00000000000000E000
...	...	...	...	...	...
1	1	-3.07E-02	-4.66E-02	-4.44E-02	1.44E-02
1	2	-6.86E-02	-9.36E-03	-4.43E-02	1.48E-02
1	3	-6.86E-02	-9.36E-03	-4.43E-02	1.48E-02
1	4	-6.86E-02	-9.36E-03	-4.43E-02	1.48E-02
1	5	-3.07E-02	-4.66E-02	-4.44E-02	1.44E-02
...	...	...	...	...	...
2	1	-1.64E-15	3.76E-01	6.29025549613785E000	3.51579365079365E000
2	2	-1.73E-15	3.76E-01	6.29025549613785E000	3.51579365079365E000
2	3	-1.55E-15	3.76E-01	6.29025549613785E000	3.51579365079365E000
2	4	-1.20E-15	3.76E-01	6.29025549613785E000	3.51579365079365E000
2	5	-1.38E-15	3.76E-01	6.29025549613785E000	3.51579365079365E000
...	...	...	...	...	...

## TD\_SHAP SQL Call Using TD\_XGBoost

```
DROP TABLE iris_predict;
CREATE MULTISET TABLE iris_predict AS (
SELECT * FROM TD_SHAP (
    ON iris_input AS InputTable
    ON iris_model AS ModelTable DIMENSION
    OUT TABLE GlobalExplanation(shap_xgb_class_out_2)
    USING
        idcolumn('id')
        InputColumns ('[1:4]')
        modeltype('classification')
        trainingFunction('td_xgboost')
        NumParallelTrees(5)
        NumBoostRounds(1)
        detailed('false')
) AS dt
) WITH DATA;
```

Label	TD_sepal_length_SHAP	TD_sepal_width_SHAP	TD_petal_length_SHAP	TD_petal_width_SHAP
0	5.42E-02	0.000000000000000E 000	8.97E-02	0.000000000000000E 000
2	1.12E-15	4.26E-01	7.79954842543079E 000	5.05536190476190E 000
1	2.67E-02	3.35E-02	6.10E-02	3.24E-02

## TD\_SHAP SQL Call

```
DROP TABLE iris_predict;
CREATE MULTISET TABLE iris_predict AS (
SELECT * FROM TD_SHAP (
    ON iris_input AS InputTable
    ON iris_model AS ModelTable DIMENSION
    USING
        idcolumn('id')
        InputColumns ('[1:4]')
        modeltype('classification')
        trainingFunction('td_xgboost')
        NumParallelTrees(5)
        NumBoostRounds(1)
```

```

detailed('true')
) AS dt
) WITH DATA;

```

**TD\_SHAP Output**

tree_num	IterNum	Label	id	TD_sepallength_SHAP	TD_sepal_width_SHAP	TD_petal_length_SHAP	TD_petal_width_SHAP
01	1	0	25	1. 97E-01	0. 000000000000000E 000	0. 000000000000000E 000	0. 000000000000000E 000
01	1	0	8	1. 97E-01	0. 000000000000000E 000	0. 000000000000000E 000	0. 000000000000000E 000
01	1	0	113	-1. 36E-01	0. 000000000000000E 000	0. 000000000000000E 000	0. 000000000000000E 000
01	1	0	1	1. 97E-01	0. 000000000000000E 000	0. 000000000000000E 000	0. 000000000000000E 000
01	1	0	148	-1. 36E-01	0. 000000000000000E 000	0. 000000000000000E 000	0. 000000000000000E 000
...	...	...	...	...	...	...	...
FINAL	1	2	108	-8. 22E-16	1.55E-01	-1. 23097445038622E 001	-8. 66277777777778E 000
FINAL	1	2	15		3.76E-01	6. 29025549613785E 000	3. 51579365079365E 000
FINAL	1	2	139		1.55E-01	9.03E-02	-8. 66277777777778E 000
FINAL	1	2	78		1.55E-01	-6. 10974450386215E 000	-2. 46277777777778E 000
FINAL	1	2	100	-6. 22E-16	3.76E-01	6. 29025549613785E 000	3. 51579365079365E 000

## TD\_Silhouette

TD\_Silhouette function is a method of interpretation and validation of consistency within clusters of data. The function determines how well the data is clustered among clusters.

The silhouette value determines the similarity of an object to its cluster (cohesion) compared to other clusters (separation). The silhouette plot displays a measure of how close each point in one cluster is to the points in the neighboring clusters and thus provides a way to assess parameters like the optimal number of clusters.

The silhouette scores are as follows:

- 1: Data is appropriately clustered
- -1: Data is not appropriately clustered
- 0: Datum is on the border of two natural clusters

### Note:

The algorithm used in this function is of the order of  $N^2$  (where N is the number of rows). Queries run significantly longer as the number of rows increases in the input table.

The silhouette coefficient can be used to evaluate the performance of clustering algorithms. It is a measure of how well each data point fits its assigned cluster and how different it is from the points in the neighboring clusters.

The silhouette coefficient ranges from -1 to 1, where values closer to 1 indicate that the data point is well-clustered and values closer to -1 indicate that the data point may be assigned to the wrong cluster. A value of 0 indicates that the data point is close to the decision boundary between two clusters.

The silhouette coefficient can be used to compare the performance of different clustering algorithms or to optimize the hyperparameters of a specific algorithm. The algorithm that produces the highest silhouette coefficient is usually considered to be the best choice for clustering the dataset.

Different clustering algorithms may perform differently depending on the characteristics of the dataset, and the silhouette coefficient can help identify which algorithm is best suited for a specific dataset.

For example, density-based clustering algorithms may perform better on datasets with complex shapes and varying densities, while hierarchical clustering algorithms may perform better on datasets with clear hierarchical structures.

The most commonly used clustering algorithms follow:

- K-means: Used to classify n-observations in k clusters. The algorithm starts with randomly chosen centroids and then assigns each point to the nearest centroid. It then recalculates the centroids and repeats the process until the centroids stabilize. K-means is simple, efficient, and widely used for clustering data in different industries, such as finance, biology, social sciences, and image processing.
- Hierarchical Clustering: Used to group similar data points into nested clusters by using either agglomerative or divisive methods. In agglomerative clustering, each point starts in a separate cluster and pairs of clusters are merged based on their similarity. Divisive clustering starts with all the points

in one cluster and recursively splits them into smaller clusters. Hierarchical clustering is often used in biology, text analysis, and social networks.

- DBSCAN: Density-Based Spatial Clustering of Applications with Noise is a clustering algorithm that identifies clusters based on their density. It starts by randomly picking a data point, collects its neighbors in a given radius, and then expands the clusters by including their neighbors until the density reaches a minimum threshold. DBSCAN is used when the clusters are irregularly shaped or of different sizes, and when the noise/outliers need to be detected.
- Mean Shift: This algorithm attempts to find the local maximum of a density function by shifting the centroid of a cluster over a feature space. The algorithm starts by defining a kernel function that weighs the distance of data points from the centroid. The centroid is then shifted to the area with the highest density of data points until convergence. Mean shift is used in computer vision, image processing, and object tracking.
- Spectral Clustering: Spectral clustering is based on graph theory, where the data is viewed as a graph with pairwise similarities as edges between nodes. The algorithm reduces the graph to k clusters by computing the Laplacian matrix of the graph, diagonalizing it, and then clustering the eigenvectors of the matrix. Spectral clustering is useful when the data is non-linearly separable, and when there is a clear graphical representation of the data. It is used in social networks analysis, image segmentation, and recommendation systems.

After clustering, the silhouette coefficient is calculated to assess the overall quality of clustering and to compare the clustering solutions of different algorithms on the same dataset. It is a useful metric to evaluate the quality of clustering solutions. It provides a quantitative measure of how well each data point is assigned to its cluster, based on its distance from other points in the same cluster and from points in neighboring clusters.

The silhouette coefficient is calculated for each data point i as follows:

- Compute the average distance ( $a_i$ ) between data point i and all other points in the same cluster.
- Compute the average distance ( $b_i$ ) between data point i and all points in the nearest neighboring cluster.
- Compute the silhouette coefficient for data point i as  $(b_i - a_i) / \max(a_i, b_i)$ .
- Compute the overall Silhouette coefficient as the average of all Silhouette coefficients for all data points.

In the formula, the denominator represents the maximum of  $a_i$  and  $b_i$ , which ensures that the silhouette coefficient falls within the range of -1 to 1. If  $a_i$  is much smaller than  $b_i$ , the data point i is well-clustered, and the silhouette coefficient approaches 1. If  $a_i$  is much larger than  $b_i$ , the data point i may be assigned to the wrong cluster, and the silhouette coefficient approaches -1. If  $a_i$  and  $b_i$  are similar, the data point i is near the decision boundary between two clusters, and the silhouette coefficient approaches 0.

Suppose you have a dataset with five data points, and you apply K-means clustering with K=2. You compute the Euclidean distance between each pair of data points and obtain the following distance matrix:

	1	2	3	4	5
1	0	2	6	4	3
2	2	0	5	3	2

3	6	5	0	6	7
4	4	3	6	0	5
5	3	2	7	5	0

You assign data points 1, 2, and 3 to cluster A and data points 4 and 5 to cluster B. You then compute the silhouette coefficient for each data point:

- Data point 1:

$$a_1 = (0+2)/2 = 1, \text{ and } b_1 = 3$$

The silhouette coefficient for data point 1 is  $(3-1)/3 = 0.67$ .

- Data point 2:

$$a_2 = (0+2)/2 = 1, \text{ and } b_2 = 3$$

The silhouette coefficient for data point 2 is  $(3-1)/3 = 0.67$ .

- Data point 3:

$$a_3 = (6+5)/2 = 5.5, \text{ and } b_3 = 3$$

The silhouette coefficient for data point 3 is  $(3-5.5)/5.5 = -0.45$ .

- Data point 4:

$$a_4 = (0+3)/2 = 1.5, \text{ and } b_4 = 2$$

The silhouette coefficient for data point 4 is  $(2-1.5)/2 = 0.25$ .

- Data point 5:

$$a_5 = (0+2)/2 = 1, \text{ and } b_5 = 3$$

The silhouette coefficient for data point 5 is  $(3-1)/3 = 0.67$ .

The average silhouette coefficient for the entire dataset is a measure of the overall quality of the clustering algorithm. Higher average silhouette coefficients indicate better clustering, while lower average silhouette coefficients suggest that the clusters may be poorly defined or overlapping.

The silhouette coefficient is a useful measure for evaluating the quality of clustering algorithms. It considers both the intra-cluster and inter-cluster distances to determine how well a data point fits into its assigned cluster, compared to how well it could fit into its nearest neighboring cluster. A high silhouette coefficient value indicates that the clustering algorithm separated the data points into distinct and well-defined clusters, while a low value suggests that the clusters may be overlapping or poorly defined. Do not use the silhouette coefficient in isolation to determine the quality of clustering. Consider other measures, such as domain-specific knowledge and visualization techniques, in evaluating the results of a clustering algorithm.

## Function Information

- [TD\\_Silhouette Syntax](#)
- [Required Syntax Elements for TD\\_Silhouette](#)
- [Optional Syntax Elements for TD\\_Silhouette](#)

- [TD\\_Silhouette Input](#)
- [TD\\_Silhouette Output](#)
- [Example: How to Use TD\\_Silhouette](#)

## TD\_Silhouette Syntax

```
TD_Silhouette (
    ON { table | view | (query) } as InputTable
    USING
        IdColumn('id_column')
        ClusterIdColumn('clusterid_column')
        TargetColumns({ 'target_column' | 'target_column_range' }[, ...])
    [
        {
            [ OutputType({ 'SCORE' | 'CLUSTER_SCORES' }) ]
        }
        |
        {
            OutputType('SAMPLE_SCORES')
            [ Accumulate({ 'accumulate_column' | 'accumulate_column_range' }[, ...]) ]
        }
    ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_Silhouette

### ON clause

Accept the InputTable clause.

### IdColumn

Specify the unique identifier column name.

### ClusterIdColumn

Specify the column name that contains the assigned clusterIds for the input data points.

**TargetColumns**

Specify the features or columns for clustering.

## Optional Syntax Elements for TD\_Silhouette

**OutputType**

Specify the output type or format.

- SCORE: Returns average silhouette score of all input samples.
- SAMPLE\_SCORES: Returns silhouette score for each input sample.
- CLUSTER\_SCORES: Returns average silhouette scores of input samples for each cluster.

Default Value: SCORE

**Accumulate**

[Applicable for 'SAMPLE\_SCORES' output type] Specify the input table columns to copy to the output table.

## TD\_Silhouette Input

### Input Table Schema

Column	Data Type	Description
id_column	ANY	The unique identifier column of the input table.
clusterid_column	BYTEINT, SMALLINT, INTEGER, BIGINT	The column that contains the assigned cluster identifiers for the input data points.
TargetColumns	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL /NUMERIC, FLOAT, REAL, DOUBLE PRECISION	The columns used for clustering.

## TD\_Silhouette Output

### Output Table Schema for SCORE

Column	Data Type	Description
Silhouette_Score	REAL	Silhouette Coefficient (that is, the Mean Silhouette Score)

## Output Table Schema for SAMPLE\_SCORES

Column	Data Type	Description
id_Column	ANY	The unique identifier of input rows copied from the input table.
clusterid_Column	BYTEINT, SMALLINT, INTEGER, BIGINT	The ClusterIds of the input data points copied from the input table.
A_i	REAL	The mean distance of a data point to other data points in the same cluster.
B_i	REAL	The minimum mean dissimilarity to other clusters.
Silhouette_Score	REAL	The silhouette score for the data point.
accumulate_columns	ANY	The specified columns in the Accumulate element are copied from the input table to the output table.

## Output Table Schema for CLUSTER\_SCORES

Column	Data Type	Description
clusterid_column	BYTEINT, SMALLINT, INTEGER, BIGINT	The clusterId of the cluster.
Silhouette_Score	REAL	The mean silhouette score for the clusterId.

## Example: How to Use TD\_Silhouette

### TD\_Silhouette InputTable

```
id clusterid c1 c2
-- -----
1      1  1  1
2      1  2  2
3      2  8  8
4      2  9  9
```

### Example: TD\_Silhouette Call Using SCORE

```
SELECT * FROM TD_Silhouette(
ON input_tbl as InputTable
USING
IdColumn('id')
ClusterIdColumn('clusterid')
```

```
TargetColumns('c1','c2')
OutputType('SCORE')
) as dt;
```

## **TD\_Silhouette Output Using SCORE**

```
silhouette_score
-----
0.856410256
```

## **Example: TD\_Silhouette Call Using SAMPLE\_SCORES**

```
SELECT * FROM TD_Silhouette(
ON input_tbl as InputTable
USING
IdColumn('id')
ClusterIdColumn('clusterid')
TargetColumns('c1','c2')
OutputType('SAMPLE_SCORES')
) as dt;
```

## **TD\_Silhouette Output Using SAMPLE\_SCORES**

id	clusterid	a_i	b_i	silhouette_score
1	1	1.414213562	10.606601718	0.866666667
4	2	1.414213562	10.606601718	0.866666667
3	2	1.414213562	9.192388155	0.846153846
2	1	1.414213562	9.192388155	0.846153846

## **Example: TD\_Silhouette Call Using CLUSTER\_SCORES**

Output when OutputType is set to CLUSTER\_SCORES.

```
SELECT * FROM TD_Silhouette(
ON input_tbl as InputTable
USING
IdColumn('id')
ClusterIdColumn('clusterid')
TargetColumns('c1','c2')
OutputType('CLUSTER_SCORES')
) as dt;
```

## TD\_Silhouette Output Using CLUSTER\_SCORES

```
clusterid silhouette_score
-----
1      0.856410256
2      0.856410256
```

### Example: TD\_Silhouette Using TargetColumns and SCORE

```
SELECT * FROM TD_Silhouette(
ON input_tbl as InputTable
USING
IdColumn('id')
ClusterIdColumn('clusterid')
TargetColumns('[2:3]')
OutputType('SCORE')
) as dt;
```

## TD\_ClassificationEvaluator

TD\_ClassificationEvaluator function computes evaluation metrics to evaluate and compare multiple classification models and summarize how close predictions are to their expected values. It takes the actual and predicted values of the dependent variables to calculate specified metrics. Apart from accuracy, the secondary output table returns micro, macro and weighted averaged metrics of precision, recall and F1 score values.

Classification problems use a confusion matrix to visualize the performance of a classifier. The confusion matrix contains predicted labels represented across the row axis and actual labels represented across the column axis. Each cell in the confusion matrix corresponds to the count of occurrences of labels in the test data.

---

#### Note:

The function works for multiclass scenarios as well. In any case, the primary output table contains classlevel metrics, whereas the secondary output table contains metrics that are applicable across classes.

---

Apart from accuracy, the secondary output table returns micro, macro, and weighted averaged metrics of precision, recall, and F1 score values.

Classification is a type of Machine Learning algorithm where the goal is to predict a categorical variable or class label based on a set of input features. The algorithm learns to classify new observations by training on a labeled dataset, where the class labels are already known. The most common type of classification is logistic regression where the algorithm models the probability of an event taking place by having the log odds for the event to be linear combination of one or more independent variables.

In the classification process, a model is trained on a dataset consisting of input variables and corresponding categorical label. The model tries to estimate the probability of an event occurring based on a given set of input variables. There are different types of classification algorithms, including decision trees, logistic regression, naïve Bayes, and so on.

You can use the following metrics to evaluate the performance of a classification model:

- Accuracy
- Precision
- Recall
- F1 score
- ROC curve

One crucial aspect of classification is selecting the appropriate variables. Too many variables can lead to overfitting, where the model performs well on the training data but poorly on the test data. On the other hand, too few variables can lead to underfitting, where the model fails to capture the underlying patterns in the data.

## Function Information

- [TD\\_ClassificationEvaluator Syntax](#)
- [Required Syntax Elements for TD\\_ClassificationEvaluator](#)
- [Optional Syntax Elements for TD\\_ClassificationEvaluator](#)
- [TD\\_ClassificationEvaluator Input](#)
- [TD\\_ClassificationEvaluator Output](#)
- [TD\\_ClassificationEvaluator Usage Notes](#)
- [Example: How to Use TD\\_ClassificationEvaluator](#)

## TD\_ClassificationEvaluator Syntax

```
TD_ClassificationEvaluator (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE OutputTable (output_table_name) ]
    USING
    ObservationColumn('observation_column')
    PredictionColumn('prediction_column')
    {
        NumLabels (labels_count) | Labels ('label1'[,...])
    }
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_ClassificationEvaluator

### ON clause

The function accepts the InputTable clause.

**Note:**

The function accepts only one ON clause for the InputTable.

### ObservationColumn

Specify the column name that has observed labels.

### PredictionColumn

Specify the column name that has predicted labels.

## Optional Syntax Elements for TD\_ClassificationEvaluator

### OUT clause

Accept the OutputTable clause.

### Labels

Specify the list of predicted labels.

### NumLabels

Specify the total count of labels.

## TD\_ClassificationEvaluator Input

### InputTable Schema

Column	Data Type	Description
ObservationColumn	BYTEINT, SHORTINT, INTEGER,CHAR, VARCHAR	The InputTable column name that has observed labels.

Column	Data Type	Description
PredictionColumn	BYTEINT, SHORTINT, INTEGER, CHAR, VARCHAR	The InputTable column name that has predicted labels.

## TD\_ClassificationEvaluator Output

### Output Table Schema

The Primary Output table is as follows:

Column	Data Type	Description
SeqNum	INTEGER	The sequence number of the row.
Prediction	VARCHAR	The column name that has predicted labels.
Mapping	VARCHAR	The mapping used for the label.
Class_N	BIGINT	The N columns denoting N labels.
Precision	REAL	The positive predictive value. Refers to the fraction of relevant instances among the total retrieved instances.
Recall	REAL	Refers to the fraction of relevant instances retrieved over the total amount of relevant instances.
F1	REAL	F1 score, defined as the harmonic mean of the precision and recall.
Support	BIGINT	The number of times a label displays in the ObservationColumn.

### Output Table Schema

The Secondary Output table is as follows:

Column	Data Type	Description
SeqNum	INTEGER	The sequence number of the row.
Metric	VARCHAR	The metric name.
MetricValue	REAL	The value for the corresponding metric.

## TD\_ClassificationEvaluator Usage Notes

Classification is a type of statistical analysis used to classify or categorize data into different groups based on their characteristics. It involves predicting a categorical response based on one or more independent variables. You can use it in finance, medicine, marketing, and so on. Logistic regression is a type of

classification where the algorithm models the probability of an event taking place by having the log odds for the event to be linear combination of one or more independent variables.

In classification algorithms, the goal is to find a decision boundary that separates the different classes in the data space.

The goal of classification algorithms, such as logistic regression, is to estimate the values of coefficients that maximize the likelihood of the observed data. The likelihood function measures the probability of observing the actual values of the dependent variable, given the predicted probabilities from the model. The maximum likelihood of the coefficients is obtained using numerical optimization techniques such as gradient descent. Gradient descent is an optimization algorithm that iteratively adjusts the values of coefficients to maximize the likelihood.

You can use the following metrics to evaluate the performance of a classification model::

- Accuracy is the fraction of predictions our model got right.
- Precision refers to the fraction of relevant instances among the total retrieved instances.
- Recall refers to the fraction of relevant instances retrieved over the total amount of relevant instances.
- F1 score is defined as the harmonic mean of precision and recall.
- ROC curve plots True positive rate vs False positive rate at different classification thresholds.

## Example: How to Use TD\_ClassificationEvaluator

### **TD\_ClassificationEvaluator Input Table**

<code>id</code>	<code>observed_value</code>	<code>predicted_value</code>
5	setosa	setosa
10	setosa	setosa
15	setosa	setosa
20	setosa	setosa
25	setosa	setosa
30	setosa	setosa
35	setosa	setosa
40	setosa	setosa
45	setosa	setosa
50	setosa	setosa
55	versicolor	versicolor
60	versicolor	versicolor
65	versicolor	versicolor
70	versicolor	versicolor
75	versicolor	versicolor
80	versicolor	versicolor
85	virginica	versicolor
90	versicolor	versicolor
95	versicolor	versicolor

```

100  versicolor      versicolor
105  virginica       virginica
110  virginica       virginica
115  virginica       virginica
120  versicolor      virginica
125  virginica       virginica
130  versicolor      virginica
135  versicolor      virginica
140  virginica       virginica
145  virginica       virginica
150  virginica       virginica

```

### **TD\_ClassificationEvaluator SQL Call**

```

SELECT * FROM TD_ClassificationEvaluator (
    ON iris_pred AS InputTable
    OUT TABLE OutputTable (additional_metrics)
    USING
        ObservationColumn ('observed_value')
        PredictionColumn ('predicted_value')
        Labels ('setosa','versicolor','virginica')
) AS dt ORDER BY SeqNum;

```

### **TD\_ClassificationEvaluator Output Table**

Primary Output Table:

Precision	Prediction	Mapping	CLASS_1	CLASS_2	CLASS_3	Support
		Recall		F1		
0	setosa	CLASS_1	10	0	0	1.00000000000000E
000	1.00000000000000E	000	1.00000000000000E	000	10	
1	versicolor	CLASS_2	0	9	1	
9.00000000000000E-001			7.50000000000000E-001	8.18181818181818E-001	12	
2	virginica	CLASS_3	0	3	7	
7.00000000000000E-001			8.75000000000000E-001	7.77777777777778E-001	8	

### **TD\_ClassificationEvaluator Output Table**

Secondary Output Table:

SeqNum	Metric	MetricValue
1	Accuracy	8.66666666666667E-001
2	Micro-Precision	8.66666666666667E-001
3	Micro-Recall	8.66666666666667E-001
4	Micro-F1	8.66666666666667E-001
5	Macro-Precision	8.66666666666667E-001
6	Macro-Recall	8.75000000000000E-001
7	Macro-F1	8.65319865319865E-001
8	Weighted-Precision	8.80000000000000E-001
9	Weighted-Recall	8.66666666666667E-001
10	Weighted-F1	8.68013468013468E-001

## TD\_RegressionEvaluator

The TD\_RegressionEvaluator function computes metrics to evaluate and compare multiple models and summarizes how close predictions are to their expected values. It takes the actual and predicted values of the dependent variables to calculate specified metrics, you choose which metrics you want to calculate from a list of supported metrics.

Regression is a type of machine learning algorithm that aims to establish a relationship between a dependent variable and one or more independent variables. Use regression to predict the value of a dependent variable based on the values of independent variables. The most common type of regression is linear regression, where the relationship between variables is assumed to be linear.

In the regression process, a model is trained on a dataset consisting of input variables and corresponding output variables. The model tries to find the best fit line or curve that passes through the data points minimizing the difference between the actual and predicted values.

Use metrics to evaluate the performance of a regression model. The most commonly used metrics are:

- Mean squared error (MSE).
- Mean absolute error (MAE).
- R-squared.

These metrics measure the accuracy of the model's predictions by comparing the predicted values with the actual values.

Selecting the appropriate features or independent variables is one crucial aspect of regression. Too many features can lead to overfitting, where the model performs well on the training data but poorly on the test data. Alternatively, too few features can lead to underfitting, where the model fails to capture the underlying patterns in the data.

### Function Information

- [TD\\_RegressionEvaluator Syntax](#)
- [Required Syntax Elements for TD\\_RegressionEvaluator](#)

- [Optional Syntax Elements in TD\\_RegressionEvaluator](#)
- [TD\\_RegressionEvaluator Input](#)
- [TD\\_RegressionEvaluator Output](#)
- [TD\\_RegressionEvaluator Usage Notes](#)
- [Example: How to Use TD\\_RegressionEvaluator](#)

## TD\_RegressionEvaluator Syntax

```
TD_RegressionEvaluator(
  ON { table | view | (query) } AS InputTable
  USING
    ObservationColumn('observation_column')
    PredictionColumn('prediction_column')
    [ Metrics('metric_1',[ 'metric_2',....'metric_n']) ]
    [ NumOfIndependentVariables(value) ]
    [ DegreesOfFreedom(df1,df2) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

---

## Required Syntax Elements for TD\_RegressionEvaluator

### ON clause

Accepts the InputTable clause.

### ObservationColumn

Specifies the column name that has observation values.

### PredictionColumn

Specifies the column name that has prediction values.

## Optional Syntax Elements in TD\_RegRESSIONEvaluator

### Metrics

Specifies the list of evaluation metrics. The function returns the following metrics if the list is not provided:

Metrics	Description
MAE	Mean absolute error (MAE) is the arithmetic average of the absolute errors between observed values and predicted values.
MSE	Mean squared error (MSE) is the average of the squares of the errors between observed values and predicted values.
MSLE	Mean Square Log Error (MSLE) is the relative difference between the log-transformed observed values and predicted values.
MAPE	Mean Absolute Percentage Error (MAPE) is the mean or average of the absolute percentage errors of forecasts.
MPE	Mean percentage error (MPE) is the computed average of percentage errors by which predicted values differ from observed values.
RMSE	Root means squared error (RMSE) is the square root of the average of the squares of the errors between observed values and predicted values.
RMSLE	Root means Square Log Error (RMSLE) is the square root of the relative difference between the log-transformed observed values and predicted values.
R2	R Squared (R2) is the proportion of the variation in the dependent variable that is predictable from the independent variable(s).
AR2	Adjusted R-squared (AR2) is a modified version of R-squared that has been adjusted for the independent variables in the model.
EV	Explained variation (EV) measures the proportion to which a mathematical model accounts for the variation (dispersion) of a given data set.
ME	Maximum-Error (ME) is the worst-case error between observed values and predicted values.
MPD	Mean Poisson Deviance (MPD) is equivalent to Tweedie Deviances when the power parameter value is 1.
MGD	Mean Gamma Deviance (MGD) is equivalent to Tweedie Deviances when the power parameter value is 2.
FSTAT	F-statistics (FSTAT) conducts an F-test.
An F-test is any statistical test in which the test statistic has an F-distribution under the null hypothesis.	
F_score	F_score value from the F-test.

Metrics	Description
F_Criticalue	F critical value from the F-test. (alpha, df1, df2, UPPER_TAILED) , alpha = 95%
P_value	Probability value associated with the F_score value (F_score, df1, df2, UPPER_TAILED)
F_conclusion	F-test result, either 'reject null hypothesis' or 'fail to reject null hypothesis'. If F_score > F_Criticalue, then 'reject null hypothesis' Else 'fail to reject null hypothesis'

**NumOfIndependentVariables**

Specifies the number of independent variables in the model.

**Note:**

Required with Adjusted R Squared (AR2) metric, otherwise ignored.

**DegreesOfFreedom**

Specifies the numerator degrees of freedom (df1) and denominator degrees of freedom (df2).

**Note:**

Required with FSTAT metric, otherwise ignored.

**TD\_RegressionEvaluator Input****Input Table Schema**

Column	Data Type	Description
Metrics	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, FLOAT, REAL, DOUBLE PRECISION.	The observation values.
Prediction_Column	BYTEINT, SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, FLOAT, REAL, DOUBLE PRECISION.	The prediction values.

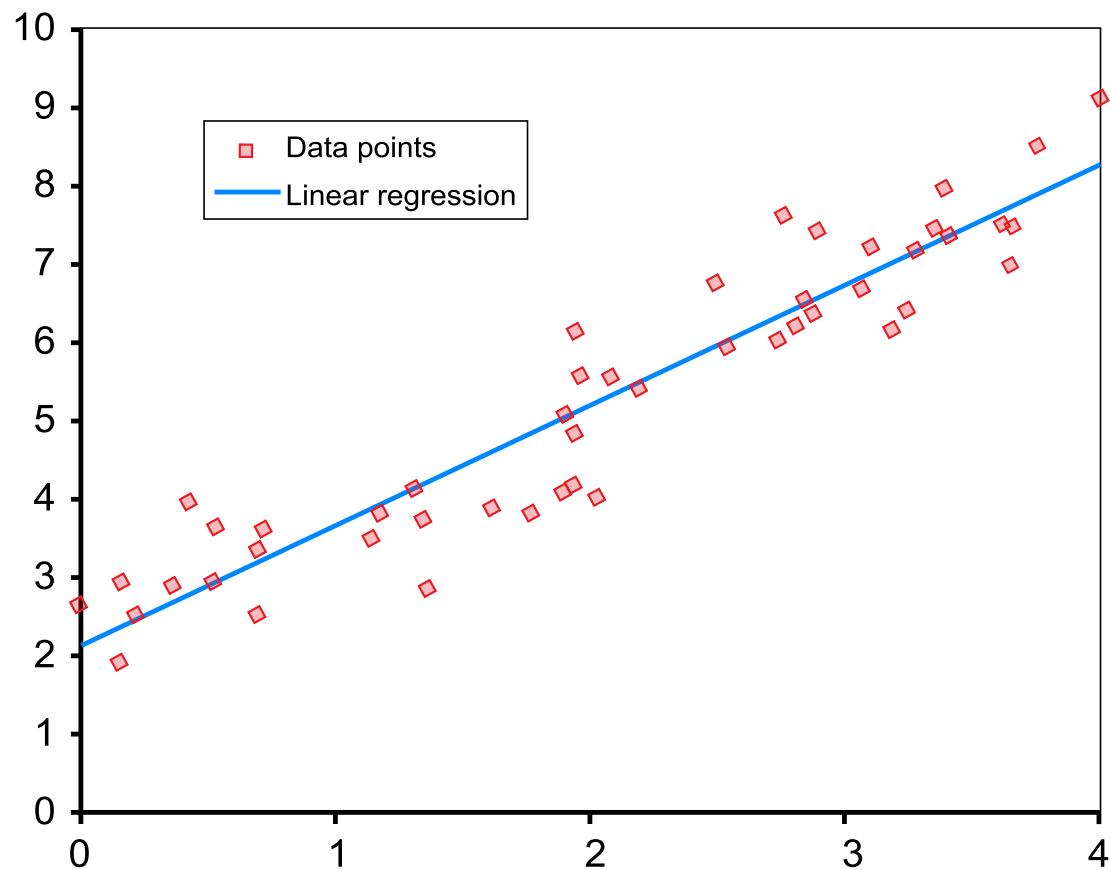
## TD\_RegressionEvaluator Output

### Output Table Schema

Column	Data Type	Description
Metrics <sub>i</sub>	FLOAT	Displays the metrics specified in the Metrics syntax element. For FSTAT, the following columns display: <ul style="list-style-type: none"><li>• F_score</li><li>• F_Criticalvalue</li><li>• P_value</li><li>• F_Conclusion</li></ul>

## TD\_RegressionEvaluator Usage Notes

In linear regression, the relationship between the dependent variable Y and independent variables X is represented by a straight line.



The equation for a simple linear regression model is:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where  $Y$  is the dependent variable,  $X$  is the independent variable,  $\beta_0$  is the intercept,  $\beta_1$  is the slope, and  $\epsilon$  is the error term. The slope represents the change in  $Y$  for a unit change in  $X$ , and the intercept represents the value of  $Y$  when  $X$  is zero.

The goal of linear regression is to estimate the values of  $\beta_0$  and  $\beta_1$  that minimize the sum of squared errors (SSE) between the predicted values and actual values. SSE is calculated as:

$$SSE = \sum (Y - \hat{Y})^2$$

where  $Y$  is the actual value of the dependent variable, and  $\hat{Y}$  is the predicted value.

There are techniques used to estimate the values of  $\beta_0$  and  $\beta_1$ , such as ordinary least squares (OLS) and gradient descent. OLS is a method that finds the values of  $\beta_0$  and  $\beta_1$  that minimize SSE by calculating their partial derivatives with respect to SSE and setting them to zero. Gradient descent is an optimization algorithm that iteratively adjusts the values of  $\beta_0$  and  $\beta_1$  to minimize SSE.

Use these metrics to evaluate the performance of a linear regression model:

Metric	Description
R-squared	Measures the proportion of variation in the dependent variable explained by the independent variables
Mean squared error (MSE)	Measures the average of the squared differences between the predicted values and actual values.
Mean absolute error (MAE)	Measures the average of the absolute differences between the predicted values and actual values.

## Example: How to Use TD\_RegRESSIONEvaluator

### TD\_RegRESSIONEvaluator InputTable

sn	price	prediction
--	--	--
13	27000.000000000	40446.918834842
16	37900.000000000	40510.148673279
25	42000.000000000	43449.453484331
38	67000.000000000	76624.879832478
53	68000.000000000	71463.482418863
104	132000.000000000	116919.270833333
111	43000.000000000	44914.331354282
117	93000.000000000	65017.025152392
132	44500.000000000	40953.263035303
140	43000.000000000	43084.061169765
142	40000.000000000	40842.383578431

157	60000.00000000	63601.429679229
161	63900.00000000	63577.865086289
162	130000.00000000	118893.154761905
176	57500.00000000	65472.830594775
177	70000.00000000	62739.489325450
195	33000.00000000	39967.151210673
198	40500.00000000	44205.358401854
224	78500.00000000	66951.540759118
234	32500.00000000	42075.221979656
237	43000.00000000	42838.368042767
239	26000.00000000	40172.789343484
249	44500.00000000	40931.339168183
251	48500.00000000	43288.879830816
254	60000.00000000	71441.950774676
255	61000.00000000	62427.945104114
260	41000.00000000	45264.892185064
274	64900.00000000	64333.059596141
294	47000.00000000	42006.077797518
301	55000.00000000	59668.624729461
306	64000.00000000	64594.501483399
317	80000.00000000	69883.134938113
329	115442.00000000	116388.318452381
339	141000.00000000	131657.638888889
340	62500.00000000	60979.553793302
353	78500.00000000	69278.445119583
355	86900.00000000	64204.176931452
364	72000.00000000	75421.353748405
367	114000.00000000	126319.444444444
377	140000.00000000	110247.569444444
401	92500.00000000	80670.206257863
403	77500.00000000	80768.002205787
408	87500.00000000	79691.581621186
411	90000.00000000	77262.550218560
440	69000.00000000	75592.404463299
441	51900.00000000	60775.316629141
443	65000.00000000	58709.087267676
459	44555.00000000	38949.583765397
463	49000.00000000	41389.644529286
469	55000.00000000	62925.830957098
472	60500.00000000	59982.047344907
527	105000.00000000	113294.568452381
530	108000.00000000	112760.937500000
540	85000.00000000	77329.868236111

## **Example: Using RMSE, R2, and FSTAT Metrics with TD\_RegressionEvaluator SQL Call**

```
SELECT * FROM TD_RegressionEvaluator(
ON decision_predict_output AS InputTable
USING
ObservationColumn('price')
PredictionColumn('prediction')
Metrics('RMSE','R2','FSTAT')
DegreesOfFreedom(5,48)
NUMOFINDEPENDENTVARIABLES(5)
) AS dt;
```

### **TD\_RegressionEvaluator Output Table**

RMSE	R2	F_SCORE	F_CRITICALVALUE	P_VALUE	F_CONCLUSION
9604.405925830	0.888090785	66.047306996	2.408514119	0.000000000	Reject null hypothesis

## **TD\_ROC**

TD\_ROC (Receiver Operating Characteristic) function accepts a set of prediction-actual pairs for a binary classification model and calculates the following values for a range of discrimination thresholds:

- True-positive rate (TPR)
- False-positive rate (FPR)
- The area under the ROC curve (AUC)
- Gini coefficient

An ROC curve shows how much a model is capable of distinguishing between classes. It is a graph showing the performance of a classification model at various classification thresholds, ranging from 0 to 1.

Each prediction by a classifier is either a:

- True Positive (TP, positive prediction that was actually positive)
- True Negative (TN, negative prediction that was actually negative)
- False Positive (FP, positive prediction that was actually negative)
- False Negative (FN, negative prediction that was actually positive)

The curve plots two parameters:

- TPR – The true positive rate also known as sensitivity, is calculated as  $TP/TP+FN$ . TPR is the probability that an actual positive is predicted as positive by the model.
- FPR – The false positive rate is calculated as  $FP/FP+TN$ . FPR is the probability that an actual negative is predicted as positive by the model.

The ROC curve shows the tradeoff between sensitivity (or TPR) and specificity ( $1 - FPR$ ). Typically, a lower decision threshold identifies more positive cases, because you set a lower bar to classify an observation as positive. However, as you classify more observations as positive due to lenient threshold, you might

mis-classify more negative cases as positive as well. A better classifier makes fewer tradeoffs to catch more of both classes correctly. A ROC plot illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1).

AUC provides an aggregate measure of performance across classification thresholds.

An AUC of 1 indicates a perfect classifier, an AUC of 0 indicates a classifier that always predicts the opposite of the actual class, and an AUC of 0.5 indicates a classifier that performs as good as random guessing.

## Function Information

- [TD\\_ROC Syntax](#)
- [Required Syntax Elements for TD\\_ROC](#)
- [Optional Syntax Elements for TD\\_ROC](#)
- [TD\\_ROC Input](#)
- [TD\\_ROC Output](#)
- [Example: How to Use TD\\_ROC](#)

## TD\_ROC Syntax

```
TD_ROC(
    ON { table | view | (query) } AS InputTable
    [ OUT [VOLATILE| PERMANENT] TABLE OutputTable(output_table_name) ]
    USING
    [ ModelIDColumn ('model_id_column') ]
    ProbabilityColumn ('probability_column')
    ObservationColumn ('observation_column')
    [ PositiveLabel ('positive_class_label') ]
    [ NumThresholds (num_thresholds) ]
    [ AUC ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
    [ Gini ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'}) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
-

## Required Syntax Elements for TD\_ROC

### **ON clause**

Accepts the InputTable clause.

### **ProbabilityColumn**

Specifies the InputTable column name that contains the probability values for predictions.

### **ObservationColumn**

Specifies the InputTable column name that contains the actual classes.

## Optional Syntax Elements for TD\_ROC

### **OUT clause**

Accepts the OutputTable clause.

### **ModelIDColumn**

Specifies the InputTable column name that contains the model or partition identifiers for the ROC curves.

### **PositiveLabel**

Specifies the positive class label.

Default: 1.

### **NumThresholds**

Specifies the thresholds number for this function. The value must be in the range [1, 10000].

Default: 50 (The function uniformly distributes the thresholds between 0 and 1.)

### **AUC**

Specifies whether the function displays the AUC calculated from the ROC values (thresholds, false positive rates, and true positive rates).

### **GINI**

Specifies whether the function displays the Gini coefficient calculated from the ROC values. The Gini coefficient is an inequality measure among the values of a frequency distribution. A Gini coefficient of 0 indicates that all values are the same. The closer the Gini coefficient is to 1, the more unequal are the values in the distribution.

## TD\_ROC Input

### Input Table Schema

Column	Data Type	Description
model_id_column	VARCHAR, CHAR, SMALLINT, BIGINT, INTEGER	The model identifier or partition for ROC curve associated with observation.
probability_column	NUMERIC	The predicted probability that the observation is in positive class.
observation_column	VARCHAR, CHAR, SMALLINT, BIGINT, INTEGER	The actual class of the observation.

## TD\_ROC Output

### Output Table Schema

If the OutputTable is given in OUT clause:

Column	Data Type	Description
Model_id	VARCHAR, CHAR, SMALLINT, BIGINT, INTEGER	The model identifier or partition for ROC curve associated with observation. The column is not displayed if you do not provide the ModelIdColumn syntax element.
Threshold	REAL	The threshold at which function classifies an observation as positive.
TPR	REAL	The TPR for the threshold. Calculated as: The number of observations correctly predicted as positive based on the threshold divided by the number of positive observations.
FPR	REAL	The FPR for threshold. Calculated as: The number of observations incorrectly predicted as positive based on the threshold divided by the number of negative observations.

### Output Table Schema

If the GINI or AUC syntax element is set to 'True':

Column	Data Type	Description
AUC	REAL	The area under the ROC curve for data in the partition. The column is not displayed if the AUC syntax element is false.

Column	Data Type	Description
GINI	REAL	GINI coefficient for ROC curve for data in the partition. The column is not displayed if the GINI syntax element is false.

## Example: How to Use TD\_ROC

### TD\_ROC Input Table

```
model id observation probability
-----
1 2 1      0.500000000
1 1 0      0.150000000
2 1 1      0.250000000
2 2 0      0.550000000
3 2 1      0.250000000
3 1 0      0.450000000
```

### TD\_ROC SQL Call

```
SELECT * from TD_ROC(
  ON roc_input AS InputTable
  OUT PERMANENT TABLE OutputTable(RocTable)
  USING
    ModelIDColumn ('model')
    ProbabilityColumn ('probability')
    ObservationColumn ('observation')
    PositiveLabel ('1')
    NumThresholds (5)
    AUC('true')
    GINI('true')
)As dt;
```

### TD\_ROC Output Table

```
model AUC          GINI
-----
1 1.000000000  1.000000000
3 0.500000000  0.000000000
2 0.000000000 -1.000000000
```

## ROC Output Table

```
SELECT * FROM RocTable;
```

model	threshold_value	tpr	fpr
1	0.500000000	1.000000000	0.000000000
1	1.000000000	0.000000000	0.000000000
1	0.250000000	1.000000000	0.000000000
1	0.750000000	0.000000000	0.000000000
1	0.000000000	1.000000000	1.000000000
2	0.250000000	1.000000000	1.000000000
2	0.000000000	1.000000000	1.000000000
2	1.000000000	0.000000000	0.000000000
2	0.750000000	0.000000000	0.000000000
2	0.500000000	0.000000000	1.000000000
3	0.500000000	0.000000000	0.000000000
3	1.000000000	0.000000000	0.000000000
3	0.750000000	0.000000000	0.000000000
3	0.250000000	1.000000000	1.000000000
3	0.000000000	1.000000000	1.000000000

## TD\_TrainTestSplit

The TD\_TrainTestSplit function simulates how a model would perform on new data. The function randomly divides the data sets into train and test subsets to evaluate machine learning algorithms and validation processes. The first subset is used to train the model. The second subset is used to make predictions and compare the predictions to actual values. The proportion of the data allocated to the training set and the testing set can vary, but a common split is 75% of the data for training and 25% for testing.

The TD\_TrainTestSplit function gives consistent results across multiple runs on same machine. With different machines, it might produce different train and test datasets.

### Function Information

- [TD\\_TrainTestSplit Syntax](#)
- [Required Syntax Elements for TD\\_TrainTestSplit](#)
- [Optional Syntax Elements for TD\\_TrainTestSplit](#)
- [TD\\_TrainTestSplit Input](#)
- [TD\\_TrainTestSplit Output](#)
- [TD\\_TrainTestSplit Examples](#)

## TD\_TrainTestSplit Syntax

```
TD_TrainTestSplit(
  ON { table | view | (query) } AS InputTable [ PARTITION BY ANY | PARTITON BY ANY
  ORDER BY col-list ]
  USING
  [IDColumn('id_column')]
  [TrainSize(float_value)]
  [TestSize(float_value)]
  [Seed(seed_value)]
  [StratifyColumn('stratify_column')]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_TrainTestSplit

**ON clause**

Accepts the InputTable clause.

## Optional Syntax Elements for TD\_TrainTestSplit

**IDColumn**

Column that contains a unique identifier for each row in the input table.

**Note:**

Mandatory when Seed argument is present so that the output of TD\_TrainTestSplit is deterministic across multiple function calls.

**TrainSize**

Size of the train dataset. It accepts float values in the (0, 1) range. Default is 0.75.

**TestSize**

Size of the test dataset. It accepts float values in the (0, 1) range. Default is 0.25.

**Seed**

Seed value that controls the shuffling applied to the data before applying the split. Pass an INT for reproducible output across multiple function calls. When the argument is not specified, different runs of the query generate different outputs. The random seed value must be in the range of 0 to INT\_MAX.

**StratifyColumn**

Column name that contains the labels indicating which data needs to be stratified.

**Note:**

- If both trainSize and testSize arguments are specified, then their sum must be equal to 1.
- TrainSize and the TestSize must be greater than the number of classes when using stratify.
- If the input table does not have an identifier column, then TD\_FillRowID can be used to generate one.

## TD\_TrainTestSplit Input

The TD\_TrainTestSplit function has one input table that contains the test data.

**Input Table Schema**

Column	Data Type	Description
IDColumn	VARCHAR, CHAR, SMALLINT, BIGINT, BYTEINT, INTEGER	Unique row identifier. Cannot be NULL.
StratifyColumn	VARCHAR, CHAR, SMALLINT, BIGINT, BYTEINT, INTEGER	If passed into the function, this column cannot have NULL values.
Columns	Any	Rows copied to the function output as either train row or test row along with IDColumn and stratify column.

## TD\_TrainTestSplit Output

**Output Table Schema**

Column	Data Type	Description
TD_IsTrainRow	BYTEINT	This column has values 0 and 1. The test rows have a value of 0, and the train rows have a value of 1.

Column	Data Type	Description
input_column	Same as input table	The columns copied from the input table.

Create the train table and test table from the output of the TD\_TrainTestSplit function in the following way:

```
CREATE Multiset table TrainTable AS (SELECT * FROM TTS_OUTPUT WHERE
TD_IsTrainRow = 1) WITH data;
```

```
CREATE Multiset table TestTable AS (SELECT * FROM TTS_OUTPUT WHERE
TD_IsTrainRow = 0) WITH data;
```

TrainTable and TestTable have the same columns and data types.

The output rows are ordered on the IDColumn to achieve the deterministic split. If Seed argument is passed, then the rows are ordered based on the random number sequence, and the rows chosen for train and test data sets are deterministic. If Seed argument is not passed, then the rows chosen for train and test data sets are not deterministic across multiple function calls.

If a column is specified for the StratifyColumn argument, then unique values from the specified column are divided in the ratio of TrainSize and TestSize.

**Note:**

When the value specified for TrainSize multiplied by the number of rows is a fraction, then the function rounds the TrainSize value to the nearest lower integer.

The number of rows in TestSize is equal to the total number of input rows minus the TrainSize value.

**Note:**

As the division depends on how the data is distributed, a 1 to 2% variance can be expected.

## TD\_TrainTestSplit Examples

The following input table is used with the examples:

**titanicDataset Input Table**

PassengerId	Survived	Pclass	Name	Age
1	0	3	Mr. Owen Harris	22
2	1	1	Mrs. John Bradley	38
3	1	3	Mrs. Laina	26
4	0	3	Mrs. Jacques Heath	35
5	0	3	Mr. William Henry	35

PassengerId	Survived	Pclass	Name	Age
6	0	3	Mr. James	38

### Example: StratifyColumn and Seed Do Not Exist

When Seed is not specified, the Seed value is randomly generated and the rows chosen for train and test data sets are not deterministic across multiple function calls.

```
SELECT * FROM TD_TrainTestSplit(
ON titanicDataset AS InputTable
USING
IDColumn('PassengerId')
trainSize(0.75)
testSize(0.25)
)AS dt;
```

#### StratifyColumn and Seed Do Not Exist Output

TD_IsTrainRow	PassengerId	Survived	Pclass	Name	Age
0	3	1	3	Miss Liana	26
0	6	0	3	Mr. James	38
1	1	0	3	Mr. Owen Harris	22
1	2	1	1	Mrs. John Bradley	38
1	4	0	3	Mr. Jacques Heath	35
1	5	0	3	Mr. William Henry	35

### Example: StratifyColumn Does Not Exist and Seed Exists

```
SELECT * FROM TD_TrainTestSplit(
ON titanicDataset AS InputTable
USING
IDColumn('PassengerId')
trainSize(0.75)
testSize(0.25)
Seed(42)
)AS dt;
```

**StratifyColumn Does Not Exist and Seed Exists Output**

TD_IsTrainRow	PassengerId	Survived	Pclass	Name	Age
0	3	1	3	Miss Liana	26
0	6	0	3	Mr. James	38
1	1	0	3	Mr. Owen Harris	22
1	2	1	1	Mrs. John Bradley	38
1	4	0	3	Mr. Jacques Heath	35
1	5	0	3	Mr. William Henry	35

**Example: StratifyColumn Exists and Seed Does Not Exist**

```
SELECT * FROM TD_TrainTestSplit(
ON titanicDataset AS InputTable
USING
IDColumn('PassengerId')
trainSize(0.75)
testSize(0.25)
stratifyColumn('Survived')
)AS dt;
```

**StratifyColumn Exists and Seed Does Not Exist Output**

TD_IsTrainRow	PassengerId	Survived	Pclass	Name	Age
0	3	1	3	Miss Liana	26
0	6	0	3	Mr. James	38
1	1	0	3	Mr. Owen Harris	22
1	2	1	1	Mrs. John Bradley	38
1	4	0	3	Mr. Jacques Heath	35
1	5	0	3	Mr. William Henry	35

**Example: StratifyColumn and Seed Exist**

```
SELECT * FROM TD_TrainTestSplit(
ON titanicDataset AS InputTable
USING
IDColumn('PassengerId')
trainSize(0.75)
```

```

testSize(0.25)
Seed (42)
stratifyColumn('Survived')
)AS dt;

```

#### StratifyColumn and Seed Exist Output

TD_IsTrainRow	PassengerId	Survived	Pclass	Name	Age
0	3	1	3	Miss Liana	26
0	6	0	3	Mr. James	38
1	1	0	3	Mr. Owen Harris	22
1	2	1	1	Mrs. John Bradley	38
1	4	0	3	Mr. Jacques Heath	35
1	5	0	3	Mr. William Henry	35

# Text Analytic Functions

- [TD\\_NgramsSplitter](#)
- [TD\\_NaiveBayesTextClassifierPredict](#)
- [TD\\_NaiveBayesTextClassifierTrainer](#)
- [TD\\_NERExtractor](#)
- [TD\\_SentimentExtractor](#)
- [TD\\_TextMorph](#)
- [TD\\_TextParser](#)
- [TD\\_TFIDF](#)
- [TD\\_WordEmbeddings](#)

## TD\_NgramsSplitter

TD\_NgramsSplitter considers each input row to be one document, and returns a row for each unique  $n$ -gram in each document. TD\_NgramsSplitter also returns, for each document, the counts of each  $n$ -gram and the total number of  $n$ -grams.

TD\_NgramsSplitter is an algorithm used in natural language processing to divide text into smaller units known as *n-grams*. An *n*-gram is a sequence of  $n$  items, such as words, letters or characters, taken from a given sample of text or speech. The TD\_NgramsSplitter algorithm takes a string of text as input and returns a list of  $n$ -grams based on a specified value of  $n$ .

One potential limitation of the TD\_NgramsSplitter algorithm is that it can produce a large number of  $n$ -grams, especially when  $n$  is large. This can result in a high-dimensional feature space that can negatively impact the performance of NLP models. To address this issue, various techniques have been developed to reduce the number of  $n$ -grams used in NLP tasks.

### Function Information

- [TD\\_NgramsSplitter Usage Notes](#)
- [TD\\_NgramsSplitter Syntax](#)
- [Required Syntax Elements for TD\\_NgramsSplitter](#)
- [Optional Syntax Elements for TD\\_NgramsSplitter](#)
- [TD\\_NgramsSplitter Input](#)
- [TD\\_NgramsSplitter Output](#)
- [Examples: How to Use TD\\_NgramsSplitter](#)

## TD\_Ngramsplitter Usage Notes

TD\_Ngramsplitter is a technique used in analytics to break down text data into smaller components called *n*-grams. An *n*-gram is a sequence of *n* words from a given text.

For example, a 2-gram (or bigram) of the sentence "The quick brown fox jumps over the lazy dog" would be "The quick", "quick brown", "brown fox", "fox jumps", "jumps over", "over the", "the lazy", and "lazy dog".

Use TD\_Ngramsplitter in analytics for various purposes such as:

- Text classification: By breaking down text into n-grams, you can create features that represent the context of the text, which can be used for text classification tasks such as sentiment analysis, spam detection, and topic modeling.
- Language modeling: *N*-grams are used to build language models that predict the likelihood of a given sequence of words. For example, a trigram language model can predict the likelihood of the next word given the two previous words.
- Information retrieval: *N*-grams are also used in information retrieval systems such as search engines to match queries with relevant documents. By breaking down documents into *n*-grams, you can efficiently index the documents and quickly retrieve relevant documents for a given query.

## TD\_Ngramsplitter Syntax

```
TD_Ngramsplitter (
    ON { table | view | (query) }
    USING
        TextColumn ('text_column')
    [ Delimiter ({'delimiter_character' | '[regex_string]'} ) ]
    Grams ({'gram_number' | 'value_range' }[,...])
    [ OverLapping({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ ConvertToLowerCase ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ Reset ('reset_character...') ]
    [ Punctuation ({'punctuation_characters' | '[regex_string]'} ) ]
    [ Reset ({'reset_characters' | '[regex_string]'} ) ]
    [ OutputTotalGramCount ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
    [ TotalCountColName ('total_count_column') ]
    [ Accumulate ({'accumulate_column' | 'accumulate_column_range' }[,...]) ]
    [ NGramColName ('ngram_column') ]
    [ GramLengthColName ('gram_length_column') ]
    [ FrequencyColName ('frequency_column') ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_Ngramsplitter

### ON clause

Accepts the InputTable clause.

TD\_Ngramsplitter takes one input table that contains the documents for which  $n$ -grams have to be generated.

### TextColumn

Specify the name of the column that contains the input text. This column must have a SQL string data type.

### Grams

Specify the length, in words, of each  $n$ -gram (that is, the value of  $n$ ). A *value\_range* has the syntax *integer1-integer2*, where *integer1* is less than or equal to *integer2*. The values of  $n$ , *integer1*, and *integer2* must be positive.

## Optional Syntax Elements for TD\_Ngramsplitter

### Delimiter

Specify, with a regular expression the character or string that separates words in the input text.

Default: '' (space).

### OverLapping

Specify whether the function allows overlapping  $n$ -grams.

Default: 'true' (Each word in each sentence starts an  $n$ -gram, if enough words follow it in the same sentence to form a whole  $n$ -gram of the specified size. For information on sentences, see the Reset syntax element description.)

### ConvertToLowercase

Specify whether the function converts all letters in the input text to lowercase.

Default: 'true'

#### Punctuation

Specify, with a regular expression, the punctuation characters for the function to remove before evaluating the input text.

Default: '~#^&\*()-'

#### Reset

Specify, with a regular expression, the character or string that ends a sentence.

Default: '.', '?'

#### Punctuation

Specify, in a string, the punctuation characters for the function to remove before evaluating the input text.

Punctuation characters can be from both Unicode and Latin character sets.

Default: '`~#^&\*() - '

#### OutputTotalGramCount

Specify whether the function returns the total number of  $n$ -grams in the document (that is, in the row) for each length  $n$  specified in the Grams syntax element. If you specify 'true', the TotalCountColName syntax element determines the name of the output table column that contains these totals.

The total number of  $n$ -grams is not necessarily the number of unique  $n$ -grams.

Default: 'false'

#### TotalCountColName

Specify the name of the output table column that appears if the value of the OutputTotalGramCount syntax element is 'true'.

Default: 'totalcnt'

#### Accumulate

Specify the names of the input table columns to copy to the output table for each  $n$ -gram. These columns cannot have the same names as those specified by the syntax elements NGramColName, GramLengthColName, and TotalCountColName.

Default: All input columns for each  $n$ -gram

**NGramColName**

Specify the name of the output table column that is to contain the created *n*-grams.

Default: 'ngram'

**GramLengthColName**

Specify the name of the output table column that is to contain the length of *n*-gram (in words).

Default: 'n'

**FrequencyColName**

Specify the name of the output table column that is to contain the count of each unique *n*-gram (that is, the number of times that each unique *n*-gram appears in the document).

Default: 'frequency'

## TD\_Ngramsplitter Input

**Input Table Schema**

Each row of the table has a document to tokenize.

Column	Data Type	Description
text_column	VARCHAR or CLOB	Document to tokenize.

## TD\_Ngramsplitter Output

**Output Table Schema**

The table has a row for each unique *n*-gram in each input document.

Column Name	Data Type	Description
totalcnt	INTEGER	Total number of n-grams in the document. This column appears in the output table only if the value of the OutputTotalGramCount argument is 'true'.
Accumulate_columns	ANY	Column copied from input table. The output table can have many such columns.
Ngram	VARCHAR	Generated <i>n</i> -grams.
N	INTEGER	Length of <i>n</i> -gram in words (value <i>n</i> ).
frequency	INTEGER	Count of each unique <i>n</i> -gram in the document.

## Examples: How to Use TD\_Ngramsplitter

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

Splits an input stream of text and returns  $n$  multigrams based on selected Reset, Punctuation, and Delimiter syntax elements.

- [TD\\_Ngramsplitter Examples Input](#)
- [Example: Omit Accumulate](#)
- [Example: Specify Accumulate](#)
- [Examples: Using Regular Expressions with TD\\_Ngramsplitter](#)

### TD\_Ngramsplitter Examples Input

The input table, paragraphs\_input, contains sentences about commonly-used machine learning techniques.

**paragraphs\_input**

paraid	paratopic	paratext
1	Decision Trees	Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the items target value. It is one of the predictive modeling approaches used in statistics, data mining, and machine learning. Tree models where the target variable can take a finite set of values are called <i>classification trees</i> . In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called <i>regression trees</i> .
2	Simple Regression	In statistics, simple linear regression is the least squares estimator of a linear regression model with a single explanatory variable. In other words, simple linear regression fits a straight line through the set of $n$ points in such a way that makes the sum of squared residuals of the model (that is, vertical distances between the points of the data set and the fitted line) as small as possible.
...	...	...

### Example: Omit Accumulate

#### TD\_Ngramsplitter SQL Call

```
SELECT * FROM TD_Ngramsplitter (
  ON paragraphs_input
  USING
```

```

TextColumn ('paratext')
Grams ('4-6')
OutputTotalGramCount ('true')
) AS dt;

```

## TD\_Ngramsplitter Output

paraid	paratopic	paratext	ngram	n	frequency	totalcnt
1	Decision Trees	Decision tree lear...	a decision tree as	4	1	73
2	Simple Regression	In statistics, simp...	a linear regression model	4	1	55
1	Decision Trees	Decision tree lear...	a decision tree as	5	1	66
2	Simple Regression	In statistics, simp..	a linear regression model with	5	1	52
1	Decision Trees	Decision tree lear...	a decision tree as a predictive	6	1	60
2	Simple Regression	In statistics, simp...	a linear regression model with a	6	1	49
1	Decision Trees	Decision tree lear...	a finite set of	4	1	73
1	Simple Regression	In statistics, simp...	a single explanatory variable	4	1	55
...	...		...	...	...	...

## Example: Specify Accumulate

### TD\_Ngramsplitter SQL Call

```

SELECT * FROM TD_Ngramsplitter (
  ON paragraphs_input
  USING
  TextColumn ('paratext')
  Grams ('4-6')
  OutputTotalGramCount ('true')
  Accumulate ('[0:1]')
) AS dt;

```

## TD\_Ngramsplitter Output

paraid	paratopic	paratext	ngram	n	frequency	totalcnt
1	Decision Trees	Decision tree lear...	a decision tree as	4	1	73
2	Simple Regression	In statistics, simp...	a linear regression model	4	1	55
1	Decision Trees	Decision tree lear...	a decision tree as	5	1	66
2	Simple Regression	In statistics, simp..	a linear regression model with	5	1	52
1	Decision Trees	Decision tree lear...	a decision tree as a predictive	6	1	60
2	Simple Regression	In statistics, simp...	a linear regression model with a	6	1	49
1	Decision Trees	Decision tree lear...	a finite set of	4	1	73
1	Simple Regression	In statistics, simp...	a single explanatory variable	4	1	55
...	...		...	...	...	...

## Examples: Using Regular Expressions with TD\_Ngramsplitter

- [Input](#)
- [Example 1: Range A-Z and a-z](#)
- [Example 2: Remove all Punctuation Marks](#)
- [Example 3: Remove all Punctuation Marks and Reset at Punctuation](#)
- [Example 4: Remove all Punctuation Marks and Delimiter at Punctuation](#)
- [Example 5: Remove all Punctuation Marks from Unicode Data](#)

### Input

```
CREATE MULTISET TABLE text_data (id int, text_data varchar(500) character set
unicode, city varchar(100));
INSERT INTO text_data (1, 'Islamabad is known for its modern architecture and
lush greenery.', 'Islamabad');
INSERT INTO text_data (2, 'Karachi, the city of lights, is famous for its
vibrant culture and bustling streets.', 'Karachi');
INSERT INTO text_data (3, 'Lahore, the heart of Pakistan, is celebrated for
its rich history and beautiful gardens.', 'Lahore');
INSERT INTO text_data (4, 'Peshawar, a city with a deep cultural heritage, is
```

known for its ancient bazaars and historic landmarks.', 'Peshawar');

```
INSERT INTO text_data (5, 'Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.', 'Quetta');
```

```
INSERT INTO text_data (6, 'Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!', 'Islamabad');
```

```
INSERT INTO text_data (7, 'Karachi$$$ - The city of 5 million lights & countless stars!! #Vibrant', 'Karachi');
```

```
INSERT INTO text_data (8, 'Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!', 'Lahore');
```

```
INSERT INTO text_data (9, 'Peshawar&& - Home to 1,000+ bazaars & many historic landmarks!!!!', 'Peshawar');
```

```
INSERT INTO text_data (10, 'Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.', 'Quetta');
```

```
CREATE multiset TABLE input_data (docid int, content varchar(5000) char set UNICODE);
```

```
INSERT INTO input_data values
```

(1, 'ప్రార్థనలు : తొలంగాణ శాసనసభ ఎన్వెన్కల నగరా మౌగిందో. 119 శాసనసభ నీయోజకవర్గాల ఎన్వెన్కలకు గొబ్బట్టే నోటిఫికేషన్ వోడుదల్లోందో. కోండ్రోర ఎన్వెన్కల సంఘం నుంచో వచ్చేచూన ముసూర్యోదా ప్రెరత్కో గవర్నెన్చర్ నరసింహన్ ఆమోదముద్దోర వోశారు. అందుకు అనుగుణంగా ఎన్వెన్కల నోర్మెన్చా కోసం రొష్టోట్రోర నోటిఫికేషన్ జోర్టే చోశారు. నోటిఫికేషన్ వోడుదలతో నామోనోషన్ల ఫుల్ టం ప్రెరారంభమ్లోందో. ఉదయం 11 గంటల నుంచో మధ్యాహ్నానం 3 గంటల వరకు నామోనోషన్లు స్టోకరోంచనున్నారు. నామోనోషన్ల ద్వారాలుకు ఈ నోర్మెన్చ 19 చోరం తోదో. ఈ నోర్మెన్చ 20న నామోనోషన్లు పరోశ్టోలోంచనున్నారు. 22 వరకు నామోనోషన్ల ఉపసంహరణకు గడుమ ఉందో.

ప్రెరత్కో నీయోజకవర్గ కోండ్రోరంలో రోలర్ నోర్మెన్చా అధికారా నోత్రుత్తోవంలో నామోనోషన్ల స్టోకరణకు ఏర్పాట్లు చోశారు. కోండ్రోర ఎన్వెన్కల సంఘం నీయుంచోన వ్యాయా పరోశ్టోలకులు ఇప్పుడు జోల్ లోలకు చోరుకున్నారు. ఇవ్వాళ్లు నుంచో తమకు అప్పుగోంచోన నీయోజకవర్గాలో వారు తమ బొధ్యాయతలు నోర్మెన్చరోంచనున్నారు. డోస్టోంబర్ 7న ఎన్వెన్కలు జరగనున్నాయి. ఫలితాలు 11న వోల్ లడ్ కానున్నాయి.

**Title:** The Wendigo

**Author:** Algernon Blackwood

**Release Date:** January 31, 2004 [EBook #10897]

[Last updated: March 10, 2012]

**Language:** English

携帯電話大手のNTTドコモが運営する「ドコモオンラインショップ」で他人のIDとパスワード(PW)を使って不正購入された米アップルのスマートフォン「iPhone X(アイフォーンテン)」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者(32)ら、いずれも中国籍の21~32歳の男4人を窃盗の疑いで11日、逮捕した。捜査関係者への取材でわかった。

捜査関係者によると、逮捕容疑は何者かと共に謀して8月ごろ、ドコモの「dアカウント」と呼ばれるサービスの他人のIDとパスワード(PW)を使って機種変更手続きを行い、不正購入されたiPhone複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。

नई दिल्ली, माला दीक्षित। अखिल भारत हिन्दू महासभा के वकील ने सुप्रीम कोर्ट से मामले की अर्जन्सी बताते हुए जल्द सुनवाई की मांग की थी, लेकिन कोर्ट ने मांग ठुकरा दी। कोर्ट ने कहा कि वह इस मामले में पहले ही आदेश कर चुका है।

पिछली सुनवाई में सुप्रीम कोर्ट ने राम जन्मभूमि विवाद पर महज तीन मिनट में मामले की अगली सुनवाई जनवरी 2019 तक टाल दी थी। बता दें कि इस मामले की सुनवाई मुख्य न्यायाधीश जस्टिस रंजन गोगोई, जस्टिस एस.के. कौल और जस्टिस के.एम. जोसेफ की पीठ ने की थी। पहले इस मामले की सुनवाई चीफ जस्टिस दीपक मिश्रा, जस्टिस अशोक भूषण और जस्टिस अब्दुल नजीर की पीठ कर रही थी। मामले की सुनवाई किस तारीख से शुरू होगी, इसका फैसला भी जनवरी में ही होगा। सुनवाई हर रोज होगी या नहीं, इस पर भी अभी निर्णय नहीं हुआ है।');

Result:

<b>id</b>	<b>text_data</b>	<b>city</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi
8	Lahore@ @@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore
9	Peshawar&&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta

### Example 1: Range A-Z and a-z

#### Query Statement

```
SELECT * FROM TD_NgramsSplitter (
    ON text_data
    USING
        TextColumn ('text_data')
        Grams ('4-6')
        Punctuation('[^A-Za-z]')
        OutputTotalGramCount ('true')
) AS dt;
```

## Output

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its	4	1	8
9	Peshawar&&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar	bazaars many historic landmarks	4	1	5
7	Peshawar&&& - Home to 1,000+ bazaars & many historic landmarks!!!	Karachi	karachi the city of	4	1	3
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	a city with greenery	4	1	7
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	for its scenic beauty	4	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	the fruit garden of	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of million	4	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	city with greenery modern	4	1	3
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic	4	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty	4	1	8

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of million lights	4	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	a city with greenery modern	5	1	2
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	quette admired for its	4	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of million lights	5	1	4
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	with greenery modern architecture	4	1	3
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	the heart of pakistan	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic beauty	5	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic	4	1	4
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	karachi the city of million	5	1	4
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	city with greenery modern architecture	5	1	2
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich	4	1	7

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	its scenic beauty mountainous	4	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	fruit garden of pakistan	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	of million lights countless	4	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	a city with greenery modern architecture	6	1	1
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history	5	1	5
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	quette admired for its scenic	5	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty and	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	million lights countless stars	4	1	5
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its rich	5	1	5
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	for its scenic beauty mountainous	5	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its scenic	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of million lights countless	5	1	4

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	rich history and beautiful	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	quette admired for its scenic beauty	6	1	3
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	the fruit garden of pakistan	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	of million lights countless stars	5	1	4
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	history and beautiful gardens	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	scenic beauty mountainous landscapes	4	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic beauty	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	karachi the city of million lights	6	1	3
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and beautiful	5	1	5
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	its scenic beauty mountainous landscapes	5	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	scenic beauty and mountainous	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of million lights countless	6	1	3

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich history	5	1	5
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic beauty mountainous	6	1	3
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its scenic beauty	6	1	4
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of million lights countless stars	6	1	3
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history and beautiful	6	1	4
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	for its scenic beauty mountainous landscapes	6	1	3
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic beauty and	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its rich history	6	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	beauty and mountainous landscapes	4	1	8
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	rich history and beautiful gardens	5	1	5

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and mountainous	5	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich history and	6	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty and mountainous	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a deep	5	1	8
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and beautiful gardens	6	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	scenic beauty and mountainous landscapes	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	with a deep cultural	4	1	10
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and mountainous landscapes	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars	4	1	10

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a deep cultural heritage	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its modern	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its ancient	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for its	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep cultural	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a deep cultural	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars and	5	1	8

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	modern architecture	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	ancient bazaars and historic	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	architecture and lush greenery	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient bazaars	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture and	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	with a deep cultural heritage	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and lush	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	bazaars and historic landmarks	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for its modern	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its ancient bazaars	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern architecture	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and historic	5	1	6

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture and lush	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient bazaars and	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its modern architecture	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep cultural heritage	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern architecture and	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars and historic	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	modern architecture and lush greenery	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	ancient bazaars and historic landmarks	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and lush greenery	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and historic landmarks	6	1	6
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	for its years of	4	1	4
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its years	4	1	4
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	its years of history	4	1	4

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
8	Lahore@ @@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its years of	5	1	2
8	Lahore@ @@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	for its years of history	5	1	2
8	Lahore@ @@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	lahore heart of pakistan	4	1	4
8	Lahore@ @@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its years of history	6	1	1
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its	4	1	6
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	the city of lights	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its vibrant	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture and	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	culture and bustling streets	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	vibrant culture and bustling	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant culture	5	1	5

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and bustling	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its vibrant culture	6	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant culture and	6	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture and bustling	6	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	vibrant culture and bustling streets	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and bustling streets	6	1	4

## Example 2: Remove all Punctuation Marks

### Query Statement

```
SELECT * FROM TD_Ngrams splitter (
    ON text_data
    USING
        TextColumn ('text_data')
        Grams ('4-6')
        Punctuation('[[[:punct:]]]')
        OutputTotalGramCount ('true')
) AS dt;
```

### Output

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its	4	1	8
9	Peshawar&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar	peshawar home to 1	4	1	3

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of 5	4	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	a city with 20	4	1	6
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	its scenic beauty 50	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	the fruit garden of	4	1	8
9	Peshawar&&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar	000 bazaars many historic	4	1	3
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of 5 million	4	1	6
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	123 a city with	4	1	6
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	for its scenic beauty	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty	4	1	8
9	Peshawar&&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar	bazaars many historic landmarks	4	1	3
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	karachi the city of	4	1	6

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	123 a city with 20	5	1	5
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and	4	1	8
9	Peshawar&&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar	000 bazaars many historic landmarks	5	1	1
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the 5 million lights	4	1	6
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	with 20 greenery 80	4	1	6
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	the heart of pakistan	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	quetta admired for its	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	karachi the city of 5	5	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	20 greenery 80 modern	4	1	6
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich	4	1	7

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	beauty 50 mountainous 50	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	fruit garden of pakistan	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of 5 million	5	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	city with 20 greenery	4	1	6
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history	5	1	5
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	for its scenic beauty 50	5	1	6
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty and	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of 5 million lights	5	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	a city with 20 greenery	5	1	5
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its rich	5	1	5
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	50 mountainous 50 landscapes	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its scenic	5	1	6

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	5 million lights countless	4	1	6
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	city with 20 greenery 80	5	1	5
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	rich history and beautiful	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	scenic beauty 50 mountainous	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	the fruit garden of pakistan	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of 5 million lights	6	1	4
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	a city with 20 greenery 80	6	1	4
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	history and beautiful gardens	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic beauty	5	1	6
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic beauty	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	karachi the city of million	6	1	4
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	with 20 greenery 80 modern	5	1	5
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and beautiful	5	1	5

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	quette admired for its scenic	5	1	6
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	scenic beauty and mountainous	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	of 5 million lights countless	5	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	123 a city with 20 greenery	6	1	4
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich history	5	1	5
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	scenic beauty 50 mountainous 50	5	1	6
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its scenic beauty	6	1	4
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	million lights countless stars	4	1	6
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	city with 20 greenery 80 modern	6	1	4
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history and beautiful	6	1	4
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic beauty 50	6	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic beauty and	6	1	4

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	5 million lights countless stars	5	1	5
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	greenery 80 modern architecture	4	1	6
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its rich history	6	1	4
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	its scenic beauty 50 mountainous	5	1	6
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	beauty and mountainous landscapes	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of 5 million lights countless	6	1	4
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	20 greenery 80 modern architecture	5	1	5
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	rich history and beautiful gardens	5	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and mountainous	5	1	4
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	beauty 50 mountainous 50 landscapes	5	1	6
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and mountainous	5	1	6
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	of 5 million lights countless	6	1	4
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	with 20 greenery 80 modern architecture	6	1	4

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich history and	6	1	4
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	its scenic beauty 50 mountainous 50	6	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty and mountainous	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and beautiful gardens	6	1	4
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	for its scenic beauty 50 mountainous	6	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	scenic beauty and mountainous landscapes	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep	4	1	10
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	quette admired for its scenic beauty	6	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and mountainous landscapes	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its	4	1	10
10	Quetta*** - Admired for its scenic beauty	Quetta	scenic beauty 50 mountainous 50 landscapes	6	1	5

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
	(50%) & mountainous (50%) landscapes.					
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a deep	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	with a deep cultural	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its modern	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for its	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a deep cultural heritage	4	1	10

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its ancient	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	modern architecture and lush	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep cultural	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	architecture and lush greenery	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a deep cultural	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture and	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars and	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and lush	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	ancient bazaars and historic	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for its modern	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient bazaars	5	1	8

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern architecture	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	with a deep cultural heritage	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture and lush	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	bazaars and historic landmarks	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its modern architecture	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its ancient bazaars	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern architecture and	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and historic	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	modern architecture and lush greenery	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient bazaars and	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and lush greenery	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep cultural heritage	6	1	6

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	its 300 years of	4	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars and historic	6	1	6
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	for its 300 years	4	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	ancient bazaars and historic landmarks	5	1	8
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its 300	4	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and historic landmarks	6	1	6
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	300 years of history	4	1	6
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	for its 300 years of	5	1	4
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	lahore 202 heart of	4	1	6
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	2020 heart of pakistan	4	1	6
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its 300 years	5	1	4
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	its 300 years of history	5	1	4
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its 300 years of	6	1	2

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
8	Lahore@ @@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	for its 300 years of history	6	1	2
8	Lahore@ @@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	lahore 2020 heart of pakistan	5	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	the city of lights	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its vibrant	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture and	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	culture and bustling streets	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	vibrant culture and bustling	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant culture	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and bustling	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its vibrant culture	6	1	4

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant culture and	6	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture and bustling	6	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	vibrant culture and bustling streets	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and bustling streets	6	1	4

### Example 3: Remove all Punctuation Marks and Reset at Punctuation

#### Query Statement

```
SELECT * FROM TD_Ngrams splitter (
    ON text_data
    USING
        TextColumn ('text_data')
        Grams ('4-6')
        Punctuation('[[{:punct:}]]')
        Reset('[[{:punct:}]]')
        OutputTotalGramCount ('true')
) AS dt;
```

#### Output

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of 5	4	1	3
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	a city with 20	4	1	1

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	for its scenic beauty	4	1	2
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	the fruit garden of	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of 5 million	4	1	3
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and	4	1	7
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic	4	1	2
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	of 5 million lights	4	1	3
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its	4	1	3
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	admired for its scenic beauty	5	1	1
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of 5 million	5	1	2

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	the heart of pakistan	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic	4	1	8
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	city of 5 million lights	5	1	2
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich	4	1	7
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	the city of 5 million lights	6	1	1
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history	5	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty and	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its rich	5	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its scenic	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	rich history and beautiful	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired	Quetta	the fruit garden of pakistan	5	1	6

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
	for its scenic beauty and mountainous landscapes.					
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	history and beautiful gardens	4	1	7
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	admired for its scenic beauty	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a deep	5	1	8
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and beautiful	5	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	scenic beauty and mountainous	4	1	8
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	with a deep cultural	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich history	5	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its scenic beauty	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	for its rich history and beautiful	6	1	4
5	Quetta, the fruit garden of Pakistan, is admired	Quetta	admired for its scenic beauty and	6	1	4

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
	for its scenic beauty and mountainous landscapes.					
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its rich history	6	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	beauty and mountainous landscapes	4	1	8
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	rich history and beautiful gardens	5	1	5
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and mountainous	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a deep cultural heritage	4	1	10
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	celebrated for its rich history and	6	1	4
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	for its scenic beauty and mountainous	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its ancient	5	1	8
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	its rich history and beautiful gardens	6	1	4

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	scenic beauty and mountainous landscapes	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep cultural	5	1	8
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	its scenic beauty and mountainous landscapes	6	1	4
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	a city with a deep cultural	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars and	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	ancient bazaars and historic	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient bazaars	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its modern	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	with a deep cultural heritage	5	1	8

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for its	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	bazaars and historic landmarks	4	1	10
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is know for its ancient bazaars	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and historic	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	modern architecture and lush	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	known for its ancient bazaars and	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	architecture and lush greenery	4	1	7
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep cultural heritage	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture and	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	for its ancient bazaars and historic	6	1	6

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and	5	1	6
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	ancient bazaars and historic landmarks	5	1	8
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for its modern	6	1	5
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	its ancient bazaars and historic landmarks	6	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern architecture	5	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	for its modern architecture and lush	6	1	5
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	is known for its modern architecture	6	1	5
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	known for its modern architecture and	6	1	5
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	modern architecture and lush greenery	5	1	6
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	its modern architecture and lush greenery	6	1	5
8	Lahore@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its 300	4	1	1
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	the city of lights	4	1	7

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its vibrant	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture and	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	culture and bustling streets	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	vibrant culture and bustling	4	1	7
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant culture	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and bustling	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its vibrant culture	6	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	famous for its vibrant culture and	6	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	for its vibrant culture and bustling	6	1	4
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	vibrant culture and bustling streets	5	1	5
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	its vibrant culture and bustling streets	6	1	4

## Example 4: Remove all Punctuation Marks and Delimiter at Punctuation

### Query Statement

```
SELECT * FROM TD_NgramsSplitter (
    ON text_data
    USING
        TextColumn ('text_data')
        Grams ('1-2')
        Punctuation('[:punct:]')
        Delimiter('[,]')
        OutputTotalGramCount ('true')
) AS dt;
```

### Output

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	quette	1	1	3
9	Peshawar&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar	peshawar home to 1	1	1	2
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	vibrant	1	1	2
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	islamabad	1	1	2
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	lahore	1	1	3
10	Quetta*** - Admired for its scenic beauty (50%) & mountainous (50%) landscapes.	Quetta	quette admired for its scenic beauty 50 mountainous 50 landscapes	1	1	1
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	the fruit garden of pakistan	1	1	3
9	Peshawar&& - Home to 1,000+ bazaars & many historic landmarks!!!	Peshawar	000 bazaars many historic	1	1	2

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
7	Karachi\$\$\$ - The city of 5 million lights & countless stars!! #Vibrant	Karachi	karachi the city of 5 million lights countless stars	1	1	2
6	Islamabad! 123 - A city with 20% greenery & 80% modern architecture!!!	Islamabad	123 a city with 20 greenery 80 modern architecture	1	1	2
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	the heart of pakistan	1	1	3
5	Quetta, the fruit garden of Pakistan, is admired for its scenic beauty and mountainous landscapes.	Quetta	is admired for its scenic beauty and mountainous landscapes	1	1	3
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	peshawar	1	1	3
3	Lahore, the heart of Pakistan, is celebrated for its rich history and beautiful gardens.	Lahore	is celebrated for its rich history and beautiful gardens	1	1	3
1	Islamabad is known for its modern architecture and lush greenery.	Islamabad	islamabad is known for its modern architecture and lush greenery.	1	1	1
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	city with a deep cultural heritage	1	1	3
8	Lahore@@@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	known for its 300 years of	1	1	2
4	Peshawar, a city with a deep cultural heritage, is known for its ancient bazaars and historic landmarks.	Peshawar	is known for its ancient bazaars and historic landmarks	1	1	3
8	Lahore@@@ - 2020: Heart of Pakistan, known for its 300+ years of history!	Lahore	lahore 2020 heart of pakistan	1	1	2
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	karachi	1	1	3

<b>id</b>	<b>text_data</b>	<b>city</b>	<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	the city of lights	1	1	3
2	Karachi, the city of lights, is famous for its vibrant culture and bustling streets.	Karachi	is famous for its vibrant culture and bustling streets	1	1	3

## Example 5: Remove all Punctuation Marks from Unicode Data

### Query Statement

```
SELECT * FROM TD_Ngrams splitter (
    ON input_data
    USING
        TextColumn ('content')
        Grams ('4-6')
        Punctuation('[[:punct:]]')
        OutputTotalGramCount ('true')
        Accumulate('docid')
) AS dt ;
```

### Output

id=1

<b>ngram</b>	<b>n</b>	<b>frequency</b>	<b>totalcnt</b>
इस पर भी आभी	4	1	192
कहा कि वह इस	4	1	192
की पीठ ने की	4	1	192
ने कहा कि वह	4	1	192
की पीठ कर रही	4	1	192
की मांग की थी	4	1	192
तक टाल दी थी।	4	1	192
पीठ ने की थी।	4	1	192
बता दें कि इस	4	1	192
१९ नवंबर १९८६	4	1	192
कि इस मामले की	4	1	192

ngram	n	frequency	totalcnt
कि वह इस मामले	4	1	192
की थी। पहले इस	4	1	192
टाल दी थी। बता	4	1	192
थी। बता दें कि	4	1	192
दी थी। बता दें	4	1	192
ने की थी। पहले	4	1	192
पीठ कर रही थी।	4	1	192
2019 तक टाल दी	4	1	192
वह इस मामले मे	4	1	192
हर रोज होगी या	4	1	192
कोर्ट ने कहा कि	4	1	192
दें कि इस मामले	4	1	192
नज़ीर की पीठ कर	4	1	192
ने कहा कि वह इस	5	1	169
पर महज तीन मिनट	4	1	192
पहले ही आदेश कर	4	1	192
भी जनवरी में ही	4	1	192
मे पहले ही आदेश	4	1	192
ही आदेश कर चुका	4	1	192
आदेश कर चुका है।	4	1	192
इस मामले मे पहले	4	1	192
कर रही थी। मामले	4	1	192
की पीठ ने की थी।	5	1	169
कौल और जस्टिस के	4	1	192
जोसेफ की पीठ ने	4	1	192
दी। कोर्ट ने कहा	4	1	192
ଧ୍ୟାଳୁଙ୍କ ଠ୍ଠ ନ୍ତ୍ରେ 19	4	1	192
पर भी अभी निर्णय	4	1	192
पहले इस मामले की	4	1	192

ngram	n	frequency	totalcnt
महज तीन मिनट में	4	1	192
मामले में पहले ही	4	1	192
रही थी। मामले की	4	1	192
रोज होगी या नहीं	4	1	192
विवाद पर महज तीन	4	1	192
कर चुका है। पिछली	4	1	192
कि वह इस मामले में	5	1	169
किस तारीख से शुरू	4	1	192
की पीठ कर रही थी।	5	1	169
कोर्ट से मामले की	4	1	192
जनवरी 2019 तक टाल	4	1	192
तक टाल दी थी। बता	5	1	169
थी। पहले इस मामले	4	1	192
थी। बता दें कि इस	5	1	169
दी थी। बता दें कि	5	1	169
ने की थी। पहले इस	5	1	169
ने मांग तुकरा दी।	4	1	192
१९७ १९४७ ७४७	4	1	192
महासभा के वकील ने	4	1	192
मिनट में मामले की	4	1	192
में मामले की अगली	4	1	192
सुनवाई की मांग की	4	1	192
इस मामले की सुनवाई	4	2	192
१९४७ ११ १९४७ १९४७	4	1	192
कहा कि वह इस मामले	5	1	169
के वकील ने सुप्रीम	4	1	192
कोर्ट ने कहा कि वह	5	1	169
जनवरी में ही होगा।	4	1	192
टाल दी थी। बता दें	5	1	169

ngram	n	frequency	totalcnt
तुकरा दी। कोर्ट ने	4	1	192
तारीख से शुरू होगी	4	1	192
तीन मिनट में मामले	4	1	192
दें कि इस मामले की	5	1	169
पीठ ने की थी। पहले	5	1	169
फैसला भी जनवरी में	4	1	192
भी अभी निर्णय नहीं	4	1	192
मे पहले ही आदेश कर	5	1	169
2019 तक टाल दी थी।	5	1	169
सुनवाई हर रोज होगी	4	1	192
ही होगा। सुनवाई हर	4	1	192
हुए जल्द सुनवाई की	4	1	192
अब्दुल नजीर की पीठ	4	1	192
अभी निर्णय नहीं हुआ	4	1	192
अशोक भूषण और जस्टिस	4	1	192
इसका फैसला भी जनवरी	4	1	192
इस पर भी अभी निर्णय	5	1	169
इस मामले मे पहले ही	5	1	169
छः नौ ल 19 छःवरूँ छःदः	5	1	169
कर रही थी। मामले की	5	1	169
की सुनवाई किस तारीख	4	1	192
कोर्ट ने मांग तुकरा	4	1	192
जल्द सुनवाई की मांग	4	1	192
जस्टिस अशोक भूषण और	4	1	192
जोसेफ की पीठ ने की	5	1	169
थी। मामले की सुनवाई	4	1	192
दी। कोर्ट ने कहा कि	5	1	169
नजीर की पीठ कर रही	5	1	169
निर्णय नहीं हुआ है।	4	1	192

ngram	n	frequency	totalcnt
ने सुप्रीम कोर्ट से	4	1	192
पर महज तीन मिनट में	5	1	169
बता दें कि इस मामले	5	1	169
मामले की सुनवाई किस	4	1	192
में ही होगा। सुनवाई	4	1	192
लेकिन कोर्ट ने मांग	4	1	192
वह इस मामले में पहले	5	1	169
सुनवाई किस तारीख से	4	1	192
हर रोज होगी या नहीं	5	1	169
ही आदेश कर चुका है।	5	1	169
होगा। सुनवाई हर रोज	4	1	192
की अगली सुनवाई जनवरी	4	1	192
की थी। पहले इस मामले	5	1	169
जनवरी 2019 तक टाल दी	5	1	169
थी। पहले इस मामले की	5	1	169
दी थी। बता दें कि इस	6	1	148
पहले ही आदेश कर चुका	5	1	169
पीठ कर रही थी। मामले	5	1	169
मांग ठुकरा दी। कोर्ट	4	1	192
मामले की अगली सुनवाई	4	1	192
मामले की सुनवाई चीफ	4	1	192
में सुप्रीम कोर्ट ने	4	1	192
सुनवाई की मांग की थी	5	1	169
सुनवाई जनवरी 2019 तक	4	1	192
सुप्रीम कोर्ट ने राम	4	1	192
है। पिछली सुनवाई में	4	1	192
४ नोवेंबर 2019 को लगा	4	1	192
कहा कि वह इस मामले में	6	1	148
कि इस मामले की सुनवाई	5	1	169

ngram	n	frequency	totalcnt
की पीठ ने की थी। पहले	6	1	148
की सुनवाई चीफ जस्टिस	4	1	192
कोर्ट ने कहा कि वह इस	6	1	148
कोर्ट ने राम जन्मभूमि	4	1	192
चुका है। पिछली सुनवाई	4	1	192
जन्मभूमि विवाद पर महज	4	1	
टाल दी थी। बता दें कि	6	1	
तक टाल दी थी। बता दें	6	1	
तीन मिनट में मामले की	5	1	
ने कहा कि वह इस मामले	6	1	
ने राम जन्मभूमि विवाद	4	1	192
पर भी अभी निर्णय नहीं	5	1	169
पीठ ने की थी। पहले इस	6	1	148
फैसला भी जनवरी में ही	5	1	169
बताते हुए जल्द सुनवाई	4	1	192
भारत हिन्दू महासभा के	4	1	192
भी जनवरी में ही होगा।	5	1	169
भूषण और जस्टिस अब्दुल	4	1	192
मामले की सुनवाई मुख्य	4	1	192
मामले में पहले ही आदेश	5	1	169
మధ్య యూహా ను 3 గంటల వరకు	4	1	192
राम जन्मभूमि विवाद पर	4	1	192
2004 ebook 10897 last	4	1	192
वकील ने सुप्रीम कोर्ट	4	1	192
विवाद पर महज तीन मिनट	5	1	169
सुनवाई हर रोज होगी या	5	1	169
से मामले की अर्जेन्सी	4	1	192
हिन्दू महासभा के वकील	4	1	192
last updated march 10	4	1	192

ngram	n	frequency	totalcnt
अगली सुनवाई जनवरी 2019	4	1	192
अब्दुल नजीर की पीठ कर	5	1	169
आदेश कर चुका है। पिछली	5	1	169
और जस्टिस अब्दुल नजीर	4	1	192
कि वह इस मामले मे पहले	6	1	148
किस तारीख से शुरू होगी	5	1	169
की अर्जेन्सी बताते हुए	4	1	192
की सुनवाई किस तारीख से	5	1	169
गोप्य मुद्रण नं 3	4	1	192
जल्द सुनवाई की मांग की	5	1	169
जस्टिस अब्दुल नजीर की	4	1	192
तुकरा दी। कोर्ट ने कहा	5	1	169
दी। कोर्ट ने कहा कि वह	6	1	148
दृश्यमान 19 वर्ष	5	1	169
मुद्रण नं 3 गोप्य	4	1	192
बता दें कि इस मामले की	6	1	148
भी अभी निर्णय नहीं हुआ	5	1	169
महज तीन मिनट में मामले	5	1	169
मिनट में मामले की अगली	5	1	169
में ही होगा। सुनवाई हर	5	1	169
2019 तक टाल दी थी। बता	6	1	148
वह इस मामले मे पहले ही	6	1	148
सुप्रीम कोर्ट से मामले	4	1	192
ही होगा। सुनवाई हर रोज	5	1	169
अखिल भारत हिन्दू महासभा	4	1	192
अभी निर्णय नहीं हुआ है।	5	1	169
इसका फैसला भी जनवरी में	5	1	169
इस मामले की सुनवाई चीफ	5	1	169
की थी। पहले इस मामले की	6	1	148

ngram	n	frequency	totalcnt
की पीठ कर रही थी। मामले	6	1	148
कोर्ट ने मांग तुकरा दी।	5	1	169
चीफ जस्टिस दीपक मिश्रा	4	1	192
जोसेफ की पीठ ने की थी।	6	1	148
थी। बता दें कि इस मामले	6	1	148
थी। मामले की सुनवाई किस	5	1	169
नजीर की पीठ कर रही थी।	6	1	148
ने की थी। पहले इस मामले	6	1	148
ने मांग तुकरा दी। कोर्ट	5	1	169
पहले इस मामले की सुनवाई	5	1	169
पीठ कर रही थी। मामले की	6	1	148
मांग तुकरा दी। कोर्ट ने	5	1	169
माला दीक्षित। अखिल भारत	4	1	192
मे पहले ही आदेश कर चुका	6	1	148
रही थी। मामले की सुनवाई	5	1	169
11 अंडूल मूँचौ मुँझौ यूँ-पूँ न०	4	1	192
3 अंडूल वर्कु नौ-मूँनौ-पूँ ल०	4	1	192
सुनवाई चीफ जस्टिस दीपक	4	1	192
हुए जल्द सुनवाई की मांग	5	1	169
release date january 31	4	1	192
अर्जेंसी बताते हुए जल्द	4	1	192
इस पर भी अभी निर्णय नहीं	6	1	148
इस मामले की सुनवाई मुख्य	5	1	169
इस मामले मे पहले ही आदेश	6	1	148
कर चुका है। पिछली सुनवाई	5	1	169
के वकील ने सुप्रीम कोर्ट	5	1	169
जनवरी 2019 तक टाल दी थी।	6	1	148
ने राम जन्मभूमि विवाद पर	5	1	169
नौ-मूँनौ-पूँ ल० दौँ-झलुकु शै नौ०	4	1	192

ngram	n	frequency	totalcnt
पहले ही आदेश कर चुका है।	6	1	148
पिछली सुनवाई में सुप्रीम	4	1	192
बताते हुए जल्द सुनवाई की	5	1	169
मामले की अर्जन्सी बताते	4	1	192
मामले में पहले ही आदेश कर	6	1	148
में मामले की अगली सुनवाई	5	1	169
में सुप्रीम कोर्ट ने राम	5	1	169
10897 last updated march	4	1	192
वकील ने सुप्रीम कोर्ट से	5	1	169
सुनवाई किस तारीख से शुरू	5	1	169
सुनवाई जनवरी 2019 तक टाल	5	1	169
सुनवाई में सुप्रीम कोर्ट	4	1	192
हिन्दू महासभा के वकील ने	5	1	169
होगा। सुनवाई हर रोज होगी	5	1	169
ebook 10897 last updated	4	1	192
title the wendigo author	4	1	192
अगली सुनवाई जनवरी 2019 तक	5	1	169
॥नै॥ नै॥ कल ॥१०॥५० ॥१०॥५० नै॥ वच॥ च॥५	4	1	192
और जस्टिस अब्दुल नज़ीर की	5	1	169
की अगली सुनवाई जनवरी 2019	5	1	169
की सुनवाई मुख्य न्यायाधीश	4	1	192
॥५॥०॥८॥४ ॥नै॥ नै॥ कल ॥१०॥५० ॥१०॥५० नै॥	4	1	192
चुका है। पिछली सुनवाई में	5	1	169
जनवरी में ही होगा। सुनवाई	5	1	169
जन्मभूमि विवाद पर महज तीन	5	1	169
जल्द सुनवाई की मांग की थी	6	1	148
तुकरा दी। कोर्ट ने कहा कि	6	1	148
दीक्षित। अखिल भारत हिन्दू	4	1	192
दें कि इस मामले की सुनवाई	6	1	148

ngram	n	frequency	totalcnt
ने सुप्रीम कोर्ट से मामले	5	1	169
पर भी अभी निर्णय नहीं हुआ	6	1	148
पर महज तीन मिनट में मामले	6	1	148
महज तीन मिनट में मामले की	6	1	148
महासभा के वकील ने सुप्रीम	5	1	169
मामले की सुनवाई किस तारीख	5	1	169
राम जन्मभूमि विवाद पर महज	5	1	169
11 अ०६७ न००४९ मध्ये य०५०३	5	1	169
लेकिन कोर्ट ने मांग तुकरा	5	1	169
विवाद पर महज तीन मिनट में	6	1	148
सुप्रीम कोर्ट से मामले की	5	1	169
ही आदेश कर चुका है। पिछली	6	1	148
अखिल भारत हिन्दू महासभा के	5	1	169
अब्दुल नज़ीर की पीठ कर रही	6	1	148
अशोक भूषण और जस्टिस अब्दुल	5	1	169
इसका फैसला भी जनवरी में ही	6	1	148
कर रही थी। मामले की सुनवाई	6	1	148
की सुनवाई चीफ जस्टिस दीपक	5	1	169
जस्टिस अब्दुल नज़ीर की पीठ	5	1	169
जस्टिस अशोक भूषण और जस्टिस	5	1	169
तीन मिनट में मामले की अगली	6	1	148
ने मांग तुकरा दी। कोर्ट ने	6	1	148
भारत हिन्दू महासभा के वकील	5	1	169
भी अभी निर्णय नहीं हुआ है।	6	1	148
मामले की अगली सुनवाई जनवरी	5	1	169
में ही होगा। सुनवाई हर रोज	6	1	148
सुनवाई हर रोज होगी या नहीं	6	1	148
न०५१० न००४९ मध्ये चौन मुलाय०८०	4	1	192
हुए जल्द सुनवाई की मांग की	6	1	148



ngram	n	frequency	totalcnt
की अगली सुनवाई जनवरी 2019 तक	6	1	148
जनवरी में ही होगा। सुनवाई हर	6	1	148
छैलोगाइ छापनपढ़ एन्नॉक्ल नगौरा	4	1	192
ने राम जन्मभूमि विवाद पर महज	6	1	148
ने सुप्रीम कोर्ट से मामले की	6	1	148
पहले इस मामले की सुनवाई चीफ	6	1	148
भी जनवरी में ही होगा। सुनवाई	6	1	148
मामले की अर्जेन्सी बताते हुए	5	1	169
मामले की सुनवाई किस तारीख से	6	1	148
22 फरवरी नॉमूनैप्पनै ल छपनपॉरणकु	4	1	192
छापनपढ़ एन्नॉक्ल नगौरा मौगौंदौ	4	1	192
है। पिछली सुनवाई में सुप्रीम	5	1	169
अगली सुनवाई जनवरी 2019 तक टाल	6	1	148
आदेश कर चुका है। पिछली सुनवाई	6	1	148
बैठाकै थूं नूंचूं तमकु अपैपैगौंदौन	4	1	192
और जस्टिस अब्दुल नज़ीर की पीठ	6	1	148
कोर्ट ने मांग ठुकरा दी। कोर्ट	6	1	148
कौंदैर एन्नॉक्ल नूंफूं नौयूमौंचूंन	4	1	192
कौंनूं रौझै थैर नौंथूंफूंकौंप्पनै जौरौं	4	1	192
जस्टिस अब्दुल नज़ीर की पीठ कर	6	1	148
थी। मामले की सुनवाई किस तारीख	6	1	148
नूंचूं पचैचूंन मुस्तायौदौ पैरैतौकौं	4	1	192
बताते हुए जल्द सुनवाई की मांग	6	1	148
भारत हिन्दू महासभा के वकील ने	6	1	148
मिनट में मामले की अगली सुनवाई	6	1	148
राम जन्मभूमि विवाद पर महज तीन	6	1	148
2004 ebook 10897 last updated	5	1	169
लेकिन कोर्ट ने मांग ठुकरा दी।	6	1	148
सुनवाई किस तारीख से शुरू होगी	6	1	148

ngram	n	frequency	totalcnt
सुनवाई मुख्य न्यायाधीश जस्टिस	4	1	192
सुप्रीम कोर्ट ने राम जन्मभूमि	5	1	169
अनुग्रहांगा एन्‌नॉकल नॉर्डॅव्हॉड कॉन्सॉन	4	1	192
इस मामले की सुनवाई चीफ जस्टिस	6	1	148
छंदमें 11 गोंठल नूंचँ मध्यैयौर्हाँ नं 3	6	1	148
कोर्ट ने राम जन्मभूमि विवाद पर	6	1	192
जन्मभूमि विवाद पर महज तीन मिनट	6	1	192
पिछली सुनवाई में सुप्रीम कोर्ट	5	1	169
भूषण और जस्टिस अब्दुल नजीर की	6	1	148
माला दीक्षित। अखिल भारत हिन्दू	5	1	169
में मामले की अगली सुनवाई जनवरी	6	1	148
11 गोंठल नूंचँ मध्यैयौर्हाँ नं 3 गोंठल	6	1	148
वकील ने सुप्रीम कोर्ट से मामले	6	1	148
सुनवाई चीफ जस्टिस दीपक मिश्रा	5	1	169
पूँछमें नॉयमॉन्चॉनॉवैयैयौर्हाँ पैरॉक्टीलक्टुल	4	1	192
blackwood release date january	4	1	192
ebook 10897 last updated march	5	1	169
अखिल भारत हिन्दू महासभा के वकील	6	1	148
अर्जेन्सी बताते हुए जल्द सुनवाई	5	1	169
नॉमॉनॉप्स्नैल छंपसंपॉर्णकु गद्वमु छंदॉ	4	1	192
महासभा के वकील ने सुप्रीम कोर्ट	6	1	148
मामले की अगली सुनवाई जनवरी 2019	6	1	148
मामले की सुनवाई मुख्य न्यायाधीश	5	1	169
रॉप्सैटैर नॉटैटैफॉक्सैप्सैजैरै चैरैशैरु	4	1	192
वैचैचॉन मुमॉनॉयॉदॉवैरैरैतैकै गैरैनैरै	4	1	192
वरकु नॉमॉनॉप्स्नैल छंपसंपॉर्णकु गद्वमु	4	1	192
सुनवाई में सुप्रीम कोर्ट ने राम	6	1	148
से मामले की अर्जेन्सी बताते हुए	6	1	148
algernon blackwood release date	4	1	192





ngram	n	frequency	totalcnt
నుంచో తమకు అప్పేవగెంచొన నోయోజకవర్గాల్లో	4	1	192
నుంచో వచ్చేచోన ముసూయోదొప్పేరతోకో గపర్చేనర్చే	5	1	169
నొల 20న నొమోనోష్టన్లు పరోశోలోంచనున్నారు	4	1	192
మాలా దీక్షితా అఖిల భారత హిందూ మహాసభా	6	1	148
శాసనసభ నోయోజకవర్గాల్లో ఎన్నోకలకు గొజోట్లే	4	1	192
the wendigo author algernon blackwood	5	1	169
అనుగుణంగొ ఎన్నోకల నోర్చేవహాణ కోసం రొష్టుట్లో	5	1	169
ఎన్నోకల సంఘం నోయమోంచోన వ్యయ పరోశోలకులు	5	1	169
నోట్లోఫోకోష్టన్ వోడుదలతో నొమోనోష్టన్లు ఘుట్లోటం	4	1	192
ప్పేరతో నోయోజకవర్గా కోంద్రేరంలో రోటర్ నోంగ్	4	1	192
మామలె కి సునవాఈ ముఖ్య న్యాయాధీశ జస్టిస్	6	1	148
వోడుదలతో నొమోనోష్టన్లు ఘుట్లోటం ప్పేరొరంభమైందో	4	1	192
author algernon blackwood release date	5	1	169
అధికారో నోత్తృత్తేపంలో నొమోనోష్టన్లు స్టోకరణకు	4	1	192
శాసనొల 20న నొమోనోష్టన్లు పరోశోలోంచనున్నారు	5	1	169
గంటల వరకు నొమోనోష్టన్లు స్టోకరంచనున్నారు	4	1	192
తమకు అప్పేవగొంచొన నోయోజకవర్గాల్లో వారు తమ	5	1	169
నుంచో మధ్యొహాంనం 3 గంటల వరకు నొమోనోష్టన్లు	6	1	148
22 వరకు నొమోనోష్టన్లు ఉపసంహరణకు గడువు ఉందో	6	1	148
సంఘం నోయమోంచోన వ్యయ పరోశోలకులు ఇప్పుపట్టో	5	1	169
హైదరాబాద్ తొలంగాణ శాసనసభ ఎన్నోకల నగ్ారో	5	1	169

ngram	n	frequency	totalcnt
algernon blackwood release date january	5	1	169
ఎన్‌నోకల నోర్‌వహి కొన్‌సం రొష్‌ట్‌ర నోట్‌ఫోక్‌ప్స్‌	5	1	169
కోండ్‌రంల్‌ రోటర్‌ నోంగ్‌ అధోకొర్‌ నోత్తుత్‌ వంల్‌	4	1	192
నోయ్‌జకవర్‌ గ కోండ్‌రంల్‌ రోటర్‌ నోంగ్‌ అధోకొర్‌	4	1	192
పచ్‌చోన ముసొయోదొ ప్రతోక్‌ గవర్‌నర్‌ నరసింహాన్‌	5	1	169
సునవాఇ ముఖ్య న్యాయాధిశ జస్టిస రంజన గోగోఇ	6	1	148
ఎన్‌నోకలకు గోజోట్‌ నోట్‌ఫోక్‌ప్స్‌ వ్డుదల్‌టెండ్‌	4	1	192
కోండ్‌ర ఎన్‌నోకల సంఘం నుంచో పచ్‌చోన ముసొయోదొ	6	1	148
నోత్తుత్‌ వంల్‌ నొమోనోప్స్‌ల స్పెకరణకు ఏర్‌పొట్‌లు	4	1	192
ప్రతోక్‌ గవర్‌నర్‌ నరసింహాన్‌ ఆమ్‌దముద్‌ ర వోశారు	5	1	169
రోటర్‌ నోంగ్‌ అధోకొర్‌ నోత్తుత్‌ వంల్‌ నొమోనోప్స్‌ల	4	1	192
119 శాసనభ నోయ్‌జకవర్‌గాల ఎన్‌నోకలకు గోజోట్‌	5	1	169
3 గంటల వరకు నొమోనోప్స్‌లు స్పెకరంచనున్‌నారు	5	1	169
wendigo author algernon blackwood release	5	1	169
ఎన్‌నోకల సంఘం నుంచో పచ్‌చోన ముసొయోదొ ప్రతోక్‌	5	1	169
నుంచో తమకు అప్‌పగోంచోన నోయ్‌జకవర్‌గాల్‌లో వొరు	4	1	192
పరోళీలకులు ఇప్‌పట్‌కో జోల్‌లొలకు చోరుకున్‌నారు	6	1	148
సంఘం నుంచో పచ్‌చోన ముసొయోదొ ప్రతోక్‌ గవర్‌నర్‌	6	1	148
algernon blackwood release date january 31	5	1	169
అప్‌పగోంచోన నోయ్‌జకవర్‌గాల్‌లో వొరు తమ బాధ్‌యతలు	4	1	192

ngram	n	frequency	totalcnt
నోయ్‌జికవర్‌గాల ఎన్‌నోకలకు గొజీబ్ నోట్‌ఫోక్‌షెన్	5		169
ముసొయొదొ ప్‌రత్కొ గవర్‌నర్‌ నరసింహస్ ఆమ్‌దముద్‌ర	6	1	148
title the wendigo author algernon blackwood	5	1	169
నోయుమొంచొన ప్‌యయ పరోశ్‌లకులు ఇప్‌పట్‌క్‌జోల్‌లొలకు	6	1	148
నోర్‌వహణ కొసం రొష్‌ట్‌ర నోట్‌ఫోక్‌షెన్ జొర్థ్ చ్‌శారు	6	1	148
అందుకు అనుగుణంగొ ఎన్‌నోకల నోర్‌వహణ కొసం రొష్‌ట్‌ర	5	1	169
ఇవొళ్‌టు నుంచొ తమకు అప్‌పగొంచొన నోయ్‌జికవర్‌గాల్‌లొ	6	1	148
ఎన్‌నోకల నోర్‌వహణ కొసం రొష్‌ట్‌ర నోట్‌ఫోక్‌షెన్ జొర్థ్	6	1	148
క్‌ంద్‌ర ఎన్‌నోకల సంఘం నోయుమొంచొన ప్‌యయ పరోశ్‌లకులు	6	1	148
నుంచొ వచ్‌చొన ముసొయొదొ ప్‌రత్కొ గవర్‌నర్‌ నరసింహస్	5	1	169
ప్‌రత్కొ నోయ్‌జికవర్‌గ క్‌ంద్‌రంలొ రీలర్‌సింగ్ అఫ్‌కొర్	6	1	148
author algernon blackwood release date january	6	1	148
wendigo author algernon blackwood release date	6	1	148
ఎన్‌నోకల సంఘం నోయుమొంచొన ప్‌యయ పరోశ్‌లకులు ఇప్‌పట్‌క్	5	1	169
ప్‌యయ పరోశ్‌లకులు ఇప్‌పట్‌క్ జోల్‌లొలకు చ్‌రుకున్ నొరు	6	1	148
హొదరొబొద్‌ తొలంగొణ శాసనసభ ఎన్‌నోకల నగొరొ మొగొంద్	6	1	148
తమకు అప్‌పగొంచొన నోయ్‌జికవర్‌గాల్‌లొ వొరు తమ బొధ్‌యతలు	5	1	169

ngram	n	frequency	totalcnt
నోత్యత్వంలో నామునోష్టస్ ల స్వప్తికరణకు ఏర్పాట్ లు చేశారు	5	1	169
అధికారి నోత్యత్వంలో నామునోష్టస్ ల స్వప్తికరణకు ఏర్పాట్ లు	6	1	148
సంఘం నోయమించొన వ్యయయ పరోశీలకులు ఇప్పటిక్కే జోల్ లాలకు	6	1	148
అనుగుణంగూ ఎన్ నోకల నోర్ వహణ కోసం రాష్ట్రట్ ర నోట్టప్పక్కష్టస్	6	1	148
ఇవొళ్ టు నుంచో తమకు అప్పగించోన నోయోజకవర్ గొల్ లో వారు	6	1	148
ముసొయొదొ ప్రత్తిక్ గవర్ నర్ నరసింహాన్ ఆమోదముద్ ర వేశారు	6	1	148
వచ్ చోన ముసొయొదొ ప్రత్తిక్ గవర్ నర్ నరసింహాన్ ఆమోదముద్ ర	5	1	169
శాసనసభ నోయోజకవర్ గొల ఎన్ నోకలకు గొజోట్ నోట్టప్పక్కష్టస్	5	1	169
కోండ్ రంలో రోట్రెన్సింగ్ అధికారి నోత్యత్వంలో నామునోష్టస్ ల	5	1	169
నోయోజకవర్ గ కోండ్ రంలో రోట్రెన్సింగ్ అధికారి నోత్యత్వంలో	5	1	169
నోట్టప్పిక్కష్టస్ వోడుదలతో నామునోష్టస్ ల ఘట్ టుం ప్రారంభమైందో	6	1	148
మధ్ యొప్పిం నోయోజకవర్ నోకలల నామునోష్టస్ ల	5	1	169
రోట్రెన్సింగ్ అధికారి నోత్యత్వంలో నామునోష్టస్ ల స్వప్తికరణకు	5	1	169
నోయోజకవర్ గొల్ లో వారు తమ బొధ్ యతలు నోర్ వర్ తోంచన్ నారు	5	1	169
నోయోజకవర్ గొల ఎన్ నోకలకు గొజోట్ నోట్టప్పిక్కష్టస్ వోడుదలైందో	6	1	148
119 శాసనసభ నోయోజకవర్ గొల ఎన్ నోకలకు గొజోట్ నోట్టప్పక్కష్టస్	6	1	148
అధికారి నోత్యత్వంలో నామునోష్టస్ ల స్వప్తికరణకు ఏర్పాట్ లు చేశారు	6	1	148
నోయమించొన వ్యయయ పరోశీలకులు ఇప్పటిక్కే జోల్ లాలకు చేరుకున్ నారు	6	1	148
ప్రత్త నోయోజకవర్ గ కోండ్ రంలో రోట్రెన్సింగ్ అధికారి నోత్యత్వంలో	6	1	148

ngram	n	frequency	totalcnt
கீங்கரமான சிறுவன் அதிகாரி நடித்து வங்கி நாமு நடிப்பு ல ஸ்டிக்ரஷன்கு	6	1	148
ரிடர்ம் நாம் அதிகாரி நடித்து வங்கி நாமு நடிப்பு ல ஸ்டிக்ரஷன்கு பிரைப்ளட் லு	6	1	148
சுபநஷ்ட நீயோஜிகார்ம் என் நீகலகு கீஜிட் நீயோஜிகார்ம் வீடு திட்டம்	6	1	148
நீயோஜிகார்ம் கீங்கரமான சிறுவன் நாம் நடிப்பு ல அதிகாரி நடித்து வங்கி நாமு நடிப்பு ல	6	1	148
அப்புக்கு நீயோஜிகார்ம் கீங்கரமான சிறுவன் நாம் நடித்து வங்கி நாமு நடிப்பு ல அதிகாரி நடித்து வங்கி நாமு நடிப்பு ல	4	1	192
2012 language english 携帯電話大手の n t t ドコモが運営する「ドコモオンラインショップ」で他人の i d とパスワード ( p w ) を使って不正購入された米アップルのスマートフォン「iphonex (アイフォーンテン)」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者 ( 32 ) ら、いずれも中国籍の 21 ~ 32 歳の男 4 人を窃盗の疑いで 11 日、逮捕した。捜査関係者への取材でわかった。	4	1	192
携帯電話大手の n t t ドコモが運営する「ドコモオンラインショップ」で他人の i d とパスワード ( p w ) を使って不正購入された米アップルのスマートフォン「iphonex (アイフォーンテン)」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者 ( 32 ) ら、いずれも中国籍の 21 ~ 32 歳の男 4 人を窃盗の疑いで 11 日、逮捕した。捜査関係者によると、逮捕容疑は何者かと共に謀して 8 月ごろ、ドコモの「d アカウント」と呼ばれるサービスの他人の i d とパスワード ( p w ) を使って機種変更手続きを行い、不正購入された iphonex 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。 नई दिल्ली	4	1	192
english 携帯電話大手の n t t ドコモが運営する「ドコモオンラインショップ」で他人の i d とパスワード ( p w ) を使って不正購入された米アップルのスマートフォン「iphonex (アイフォーンテン)」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者 ( 32 ) ら、いずれも中国籍の 21 ~ 32 歳の男 4 人を窃盗の疑いで 11 日、逮捕した。捜査関係者によると、逮捕容疑は何者かと共に謀して 8 月ごろ、ドコモの「d アカウント」と呼ばれるサービスの他人の i d とパスワード ( p w ) を使って機種変更手続きを行い、不正購入された iphonex 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。 नई दिल्ली	4	1	192

ngram	n	frequency	totalcnt
ウント」と呼ばれるサービスの他人の i dとパスワード( p w )を使って機種変更手続きを行い、不正購入された i p h o n e x 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。 <small>नई</small>			
language english 携帯電話大手の n t t ドコモが運営する「ドコモオンラインショップ」で他人の i dとパスワード( p w )を使って不正購入された米アップルのスマートフォン「 i p h o n e x ( アイフォーンテン ) 」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者( 32 )ら、いずれも中国籍の 21 ~ 32 歳の男4人を窃盗の疑いで 11 日、逮捕した。捜査関係者への取材でわかった。捜査関係者によると、逮捕容疑は何者かと共謀して 8 月ごろ、ドコモの「 d アカウント」と呼ばれるサービスの他人の i dとパスワード( p w )を使って機種変更手続きを行い、不正購入された i p h o n e x 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。	5	1	169
english 携帯電話大手の n t t ドコモが運営する「ドコモオンラインショップ」で他人の i dとパスワード( p w )を使って不正購入された米アップルのスマートフォン「 i p h o n e x ( アイフォーンテン ) 」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者( 32 )ら、いずれも中国籍の 21 ~ 32 歳の男4人を窃盗の疑いで 11 日、逮捕した。捜査関係者によると、逮捕容疑は何者かと共謀して 8 月ごろ、ドコモの「 d アカウント」と呼ばれるサービスの他人の i dとパスワード( p w )を使って機種変更手続きを行い、不正購入された i p h o n e x 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。 <small>नई दिल्ली</small>	5	1	169
language english 携帯電話大手の n t t ドコモが運営する「ドコモオンラインショップ」で他人の i dとパスワード( p w )を使って不正購入された米アップルのスマートフォン「 i p h o n e x ( アイフォーンテン ) 」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者( 32 )ら、いずれも中国籍の 21 ~ 32 歳の男4人を窃盗の疑いで 11 日、逮捕した。捜査関係者によると、逮捕容疑は何者かと共謀して 8 月ごろ、ドコモの「 d アカウント」と呼ばれるサービスの他人の	5	1	169

ngram	n	frequency	totalcnt
i dとパスワード ( p w )を使って機種変更手続きを行い、不正購入された i p h o n e x 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。 <small>ね</small>			
2012 language english 携帯電話大手の n t t ドコモが運営する「ドコモオンラインショッピング」で他人の i dとパスワード ( p w )を使って不正購入された米アップルのスマートフォン「i p h o n e x ( アイフォーンテン )」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者 ( 32 ) ら、いずれも中国籍の 21 ~ 32 歳の男4人を窃盗の疑いで 11 日、逮捕した。捜査関係者への取材でわかった。 捜査関係者によると、逮捕容疑は何者かと共謀して 8 月ごろ、ドコモの「d アカウント」と呼ばれるサービスの他人の i dとパスワード ( p w )を使って機種変更手続きを行い、不正購入された i p h o n e x 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。	6	1	148
2012 language english 携帯電話大手の n t t ドコモが運営する「ドコモオンラインショッピング」で他人の i dとパスワード ( p w )を使って不正購入された米アップルのスマートフォン「i p h o n e x ( アイフォーンテン )」を受け取ったとして、警視庁と千葉県警は、東京都荒川区荒川5丁目の会社役員、全英杰容疑者 ( 32 ) ら、いずれも中国籍の 21 ~ 32 歳の男4人を窃盗の疑いで 11 日、逮捕した。捜査関係者への取材でわかった。 捜査関係者によると、逮捕容疑は何者かと共謀して 8 月ごろ、ドコモの「d アカウント」と呼ばれるサービスの他人の i dとパスワード ( p w )を使って機種変更手続きを行い、不正購入された i p h o n e x 複数台を、東京都内のコンビニエンスストアで受け取ったというもの。同庁と県警は不正アクセス禁止法違反などの疑いでも調べる。 <small>ね</small>	6	1	148

## TD\_NaiveBayesTextClassifierPredict

This function uses the model output by TD\_NaiveBayesTextClassifierTrainer function to analyze the input data and make predictions.

### Function Information

- [TD\\_NaiveBayesTextClassifierPredict Syntax](#)

- [TD\\_NaiveBayesTextClassifierPredict Syntax Elements](#)
- [TD\\_NaiveBayesTextClassifierPredict Input](#)
- [TD\\_NaiveBayesTextClassifierPredict Output](#)
- [TD\\_NaiveBayesTextClassifierPredict Examples](#)

## TD\_NaiveBayesTextClassifierPredict Syntax

```
TD_NaiveBayesTextClassifierPredict (
    ON { table | view | (query) } AS PredictorValues PARTITION BY
    doc_id_column [, ...]
    ON { table | view | (query) } AS Model DIMENSION
    USING
        InputTokenColumn ('input_token_column')
        [ ModelType ({ 'Multinomial' | 'Bernoulli' }) ]
        DocIDColumns ({ 'doc_id_column' | 'doc_id_column_range' }[, ...])
        [ ModelTokenColumn ('model_token_column')
            ModelCategoryColumn ('model_category_column')
            ModelProbColumn ('model_probability_column') ]
        [ TopK ({ 'num_of_top_k_predictions' | 'num_of_top_k_predictions' }) |
            Responses ('response' [, ...]) ]
        [ OutputProb ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'}) ]
        [ Accumulate ({ 'accumulate_column' | 'accumulate_column_range' }[, ...]) ]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## TD\_NaiveBayesTextClassifierPredict Syntax Elements

### **InputTokenColumn**

Specify the name of the PredictorValues column that contains the tokens.

### **ModelType**

[Optional] Specify the model type of the text classifier.

Default: 'Multinomial'

**DocIDColumns**

Specify the names of the PredictorValues columns that contain the document identifier.

**ModelTokenColumn**

[Optional] Specify the name of the Model table column that contains the tokens.

Default: First column of Model table

**ModelCategoryColumn**

[Optional] Specify the name of the Model table column that contains the prediction categories.

Default: Second column of Model table

**ModelProbColumn**

[Optional] Specify the name of the Model table column that contains the probability values.

Default: Third column of Model table

**TopK**

[Disallowed with Responses, otherwise optional.] Specify the number of most likely prediction categories to output with their log-likelihood values (for example, the top 10 most likely prediction categories). To see the probability of each class, use OutputProb ('true').

Default: All prediction categories

**Responses**

[Disallowed with TopK, otherwise optional.] Specify the labels for which to output log-likelihood values and probabilities (with OutputProb ('true')).

**OutputProb**

Specify whether to output the calculated probability for each observation.

Default: 'false'

**Accumulate**

Specify the names of the PredictorValues table columns to copy to the output table.

**Note:**

- Specify either all or none of the syntax elements ModelTokenColumn, ModelCategoryColumn, and ModelProbColumn.
- Specifying neither TopK nor Responses is equivalent to specifying TopK.

**TD\_NaiveBayesTextClassifierPredict Input**

Table	Description
PredictorValues	Contains test data, for which to predict outcomes, in document-token pairs. To transform the input document into this form, input it to TD_TextParser or ML Engine function, TextTokenizer, or TextParser. TextTokenizer and TextParser have language-processing limitations that might limit support for Unicode input data (see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003).
Model	Model output by TD_NaiveBayesTextClassifier or ML Engine NaiveBayesTextClassifierTrainer2 function. If the latter, then for schema, see <i>Teradata Vantage™ Machine Learning Engine Analytic Function Reference</i> , B700-4003.

**PredictorValues Schema**

Column	Data Type	Description
<i>doc_id_column</i>	CHARACTER, VARCHAR, INTEGER, or SMALLINT	Identifier of document that contains classified testing tokens.
<i>token_column</i>	CHARACTER or VARCHAR	Testing token.
<i>accumulate_column</i>	Any	Column to copy to output table.

**Model Schema**

For CHARACTER and VARCHAR columns, CHARACTER SET must be either UNICODE or LATIN.

Column	Data Type	Description
token	CHARACTER or VARCHAR	Classified training token.
category	CHARACTER or VARCHAR	Prediction category for token.
prob	DOUBLE PRECISION	Probability that token is in category.

## TD\_NaiveBayesTextClassifierPredict Output

### Output Table Schema

Column	Data Type	Description
doc_id	CHARACTER, VARCHAR, INTEGER, or SMALLINT	Single- or multiple-column document identifier.
prediction	VARCHAR	Prediction category.
loglik	DOUBLE PRECISION	Loglikelihood that document belongs to category.
loglik_response	DOUBLE PRECISION	[Column appears only with Responses syntax element.] Loglikelihood that document belongs to class label <i>response</i> .
prob	DOUBLE PRECISION	[Column appears only when you both specify OutputProb ('true') and omit Responses.] Probability that document belongs to class label in prediction column, which is $\max(\text{softmax}(\text{loglik}))$ .
prob_response	DOUBLE PRECISION	[Column appears only when you specify both OutputProb ('true') and Responses. Column appears once for each specified <i>response</i> .] Probability that document belongs to class label <i>response</i> , which is $\text{softmax}(\text{loglik\_response})$ .
accumulate_column	Same as in PredictorValues table	Column copied from PredictorValues table.

## TD\_NaiveBayesTextClassifierPredict Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [TD\\_NaiveBayesTextClassifierPredict Example: TopK Specified](#)
- [TD\\_NaiveBayesTextClassifierPredict Example: Responses Specified](#)

### TD\_NaiveBayesTextClassifierPredict Example: TopK Specified

#### Input

- PredictorValues: complaints\_test\_tokenized, created by applying ML Engine TextTokenizer function to the table complaints\_test, a log of vehicle complaints, as follows:

```
CREATE MULTISET TABLE complaints_test_tokenized AS (
  SELECT doc_id, doc_name, lower(cast(token AS VARCHAR(20))) AS token
  FROM TextTokenizer (
```

```

ON complaints_test PARTITION BY ANY
USING
TextColumn ('text_data')
OutputByWord ('true')
Accumulate ('doc_id', doc_name)
) AS dt
) WITH DATA;

```

- Model: Use the following query to create the complaints\_tokens\_model:

```

CREATE TABLE complaints_tokens_model(
    token VARCHAR(100),
    category VARCHAR(100),
    prob DOUBLE PRECISION
);

```

**complaints\_test**

doc_id	doc_name	text_data
1	A	ELECTRICAL CONTROL MODULE IS SHORTENING OUT, CAUSING THE VEHICLE TO STALL. ENGINE WILL BECOME TOTALLY INOPERATIVE. CONSUMER HAD TO CHANGE ALTERNATOR/ BATTERY AND STARTER, AND MODULE REPLACED 4 TIMES, BUT DEFECT STILL OCCURRING CANNOT DETERMINE WHAT IS CAUSING THE PROBLEM.
2	B	ABS BRAKES FAIL TO OPERATE PROPERLY, AND AIR BAGS FAILED TO DEPLOY DURING A CRASH AT APPROX. 28 MPH IMPACT. MANUFACTURER NOTIFIED.
3	C	WHILE DRIVING AT 60 MPH GAS PEDAL GOT STUCK DUE TO THE RUBBER THAT IS AROUND THE GAS PEDAL.
4	D	THERE IS A KNOCKING NOISE COMING FROM THE CATALYTIC CONVERTER , AND THE VEHICLE IS STALLING. ALSO, HAS PROBLEM WITH THE STEERING.
5	E	CONSUMER WAS MAKING A TURN ,DRIVING AT APPROX 5- 10 MPH WHEN CONSUMER HIT ANOTHER VEHICLE. UPON IMPACT, DUAL AIRBAGS DID NOT DEPLOY . ALL DAMAGE WAS DONE FROM ENGINE TO TRANSMISSION,TO THE FRONT OF VEHICLE, AND THE VEHICLE CONSIDERED A TOTAL LOSS.
6	F	WHEEL BEARING AND HUBS CRACKED, CAUSING THE METAL TO GRIND WHEN MAKING A RIGHT TURN. ALSO WHEN APPLYING THE BRAKES, PEDAL GOES TO THE FLOOR, CAUSE UNKNOWN. WAS ADVISED BY MIDAS NOT TO DRIVE VEHICLE- WHEEL COULD COME OFF.
7	G	DRIVING ABOUT 5-10 MPH, THE VEHICLE HAD A LOW FRONTAL IMPACT IN WHICH THE OTHER VEHICLE HAD NO DAMAGES. UPON IMPACT, DRIVER'S AND THE PASSENGER'S AIR BAGS DID NOT DEPLOY, RESULTING IN INJURIES. PLEASE PROVIDE FURTHER INFORMATION AND VIN#.

doc_id	doc_name	text_data
8	H	THE AIR BAG WARNING LIGHT HAS COME ON. INDICATING AIRBAGS ARE INOPERATIVE.THEY WERE FIXED ONE AT THE TIME, BUT PROBLEM HAS REOCCURRED.
9	I	CONSUMER WAS DRIVING WEST WHEN THE OTHER CAR WAS GOING EAST. THE OTHER CAR TURNED IN FRONT OF CONSUMER'S VEHICLE, CONSUMER HIT OTHER VEHICLE AND STARTED TO SPIN AROUND , COULDNT STOP, RESULTING IN A CRASH. UPON IMPACT, AIRBAGS DIDN'T DEPLOY.
10	J	WHILE DRIVING ABOUT 65 MPH AND THE TRANSMISSION MADE A STRANGE NOISE, AND THE LEFT FRONT AXLE LOCKED UP. THE DEALER HAS REPAIRED THE VEHICLE.

## SQL Call

```
SELECT * FROM TD_NaiveBayesTextClassifierPredict (
    ON complaints_test_tokenized AS PredictorValues PARTITION BY doc_id
    ON complaints_tokens_model AS Model DIMENSION
    USING
        ModelType ('Bernoulli')
        InputTokenColumn ('token')
        DocIDColumns ('doc_id')
        OutputProb ('true')
        Accumulate ('doc_name')
        TopK ('2')
) AS dt ORDER BY doc_id;
```

## Output

doc_id	prediction	loglik	prob	doc_name
1	crash	-1.38044220625651E 002	1.41243173571687E-009	A
1	no_crash	-1.17666267644292E 002	9.9999998587568E-001	A
2	crash	-1.04652470718918E 002	1.70704288519507E-003	B
2	no_crash	-9.82811865081127E 001	9.98292957114805E-001	B
3	crash	-1.03026451289745E 002	2.26862573862878E-012	C
3	no_crash	-7.62146044204976E 001	9.99999999997731E-001	C
4	crash	-1.10830711173169E 002	1.42026355157382E-011	D
4	no_crash	-8.58531176043404E 001	9.9999999985797E-001	D
5	no_crash	-1.23936921216052E 002	3.43620138383542E-002	E
5	crash	-1.20601083912966E 002	9.65637986161646E-001	E
6	crash	-1.30310015371040E 002	2.61074198636704E-006	F
6	no_crash	-1.17454141890718E 002	9.99997389258014E-001	F

7	no_crash	-1.23123774759574E 002	4.23603936872661E-002	G
7	crash	-1.20005517060745E 002	9.57639606312734E-001	G
8	crash	-1.08617321658980E 002	8.92398441816595E-009	H
8	no_crash	-9.00827983614664E 001	9.99999991076016E-001	H
9	crash	-1.19919230739025E 002	2.24857954852037E-002	I
9	no_crash	-1.16147101713878E 002	9.77514204514796E-001	I
10	crash	-1.06104244132225E 002	7.57068462691010E-008	J
10	no_crash	-8.97078469668254E 001	9.99999924293154E-001	J

## TD\_NaiveBayesTextClassifierPredict Example: Responses Specified

### Input

As in [TD\\_NaiveBayesTextClassifierPredict Example: TopK Specified](#)

- PredictorValues: complaints\_test\_tokenized
- Model: complaints\_tokens\_model

### SQL Call

```
SELECT * FROM TD_NaiveBayesTextClassifierPredict (
    ON complaints_test_tokenized AS PredictorValues PARTITION BY doc_id
    ON complaints_tokens_model AS Model DIMENSION
    USING
        ModelType ('Bernoulli')
        InputTokenColumn ('token')
        DocIDColumns ('doc_id')
        OutputProb ('true')
        Accumulate ('doc_name')
        Responses ('crash', 'no crash')
) AS dt ORDER BY doc_id;
```

### Output

doc_id	prediction	loglik_crash	loglik_no_crash	prob_crash	prob_no_crash	doc_name
1	no_crash	-1.38044220625651E 002	-1.17666267644292E 002	1.41243173571687E-009	9.9999998587568E-001	A
2	no_crash	-1.04652470718918E 002	-9.82811865881127E 001	1.70704288519507E-003	9.98292957114805E-001	B
3	no_crash	-1.03026451289745E 002	-7.62146044204976E 001	2.26862573862878E-012	9.9999999997731E-001	C
4	no_crash	-1.10830711173169E 002	-8.58531176043404E 001	1.42026355157382E-011	9.99999999985797E-001	D
5	crash	-1.20601083912966E 002	-1.23936921216052E 002	9.65637986161646E-001	3.43620138383542E-002	E
6	no_crash	-1.30310015371040E 002	-1.17454141890718E 002	2.61074198636704E-006	9.99997389258014E-001	F
7	crash	-1.20005517060745E 002	-1.23123774759574E 002	9.57639606312734E-001	4.236039336872661E-002	G
8	no_crash	-1.08617321658980E 002	-9.08827983614664E 001	8.92398441816595E-009	9.9999991076016E-001	H
9	no_crash	-1.19919230739025E 002	-1.16147101713878E 002	2.24857954852037E-002	9.77514204514796E-001	I
10	no_crash	-1.06104244132225E 002	-8.97078469668254E 001	7.57068462691010E-008	9.99999924293154E-001	J

## TD\_NaiveBayesTextClassifierTrainer

The TD\_NaiveBayesTextClassifierTrainer function calculates the conditional probabilities for token-category pairs, the prior probabilities, and the missing token probabilities for all categories. The trainer

function trains the model with the probability values, and the predict function uses the values to classify documents into categories.

## Function Information

- [TD\\_NaiveBayesTextClassifierTrainer Syntax](#)
- [TD\\_NaiveBayesTextClassifierTrainer Syntax Elements](#)
- [TD\\_NaiveBayesTextClassifierTrainer Input](#)
- [TD\\_NaiveBayesTextClassifierTrainer Output](#)
- [Example: Using TD\\_NaiveBayesTextClassifierTrainer with Multinomial Model Type](#)

## TD\_NaiveBayesTextClassifierTrainer Syntax

```
TD_NaiveBayesTextClassifierTrainer (
    ON { table | view | (query) } AS InputTable
    [ OUT [ PERMANENT | VOLATILE ] TABLE ModelTable (model_table_name) ]
    USING
        TokenColumn ('token_column')
        DocCategoryColumn ('doc_category_column')
    [{{
        [ ModelType ('Multinomial') ]
    }}
    |
    {
        ModelType ('Bernoulli')
        DocIDColumn ('doc_id_column')
    }]
)
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## TD\_NaiveBayesTextClassifierTrainer Syntax Elements

### TokenColumn

[Required]: Specify the InputTable column name that contains the classified tokens.

**DocCategoryColumn**

[Required]: Specify the InputTable column name that contains the document category.

**DocIDColumn**

[Required for Bernoulli model type]: Specify the InputTable column name that contains the document identifier.

**ModelType**

[Optional]: Specify the model type of the text classifier.

Supported Model Types: Bernoulli and Multinomial

Default: Multinomial

**TD\_NaiveBayesTextClassifierTrainer Input****Input Table Schema**

Column	Data Type	Description
doc_id_column	BYTEINT/SMALLINT/ INTEGER/BIGINT or CHAR /VARCHAR	The InputTable column name that contains the document identifier.  <b>Note:</b> This column is required for 'Bernoulli' model type.
token_column	CHAR or VARCHAR	The column name that contains the classified training tokens from a tokenization function.
doc_category_column	CHAR or VARCHAR	The category of the document.

**Note:**

The following vocabulary token names are reserved:

- NAIVE\_BAYES\_TEXT\_MODEL\_TYPE
- NAIVE\_BAYES\_PRIOR\_PROBABILITY
- NAIVE\_BAYES\_MISSING\_TOKEN\_PROBABILITY

**TD\_NaiveBayesTextClassifierTrainer Output****Output Table Schema**

Column	Data Type	Description
token	VARCHAR (UNICODE)	The classified training tokens.

Column	Data Type	Description
category	VARCHAR (UNICODE)	The category of the token.
prob	DOUBLE PRECISION	The probability of the token in the category.

## Example: Using TD\_NaiveBayesTextClassifierTrainer with Multinomial Model Type

### TD\_NaiveBayesTextClassifierTrainer InputTable

```
doc_id category token
-----
1 no_crash vehicl
1 no_crash motor
1 no_crash separ
1 no_crash from
1 no_crash frame
2 crash deploy
2 crash anoth
2 crash end
2 crash vehicl
2 crash 70mph
2 crash airbag
2 crash rear
3 no_crash sunroof
3 no_crash leak
4 crash driver
4 crash sustain
4 crash injuri
```

### TD\_NaiveBayesTextClassifierTrainer SQL Call

```
SELECT * FROM TD_NaiveBayesTextClassifierTrainer (
    ON complaints_tokenized AS InputTable
    USING
        TokenColumn ('token')
        DocCategoryColumn ('category')
        ModelType ('Multinomial')
) AS dt;
```

## TD\_NaiveBayesTextClassifierTrainer Output Table

token	category	prob
driver	crash	0.076923077
vehicl	no_crash	0.086956522
leak	no_crash	0.086956522
anoth	crash	0.076923077
deploy	crash	0.076923077
airbag	crash	0.076923077
from	no_crash	0.086956522
70mph	crash	0.076923077
NAIVE_BAYES_PRIOR_PROBABILITY	crash	0.588235294
separ	no_crash	0.086956522
end	crash	0.076923077
sunroof	no_crash	0.086956522
injuri	crash	0.076923077
NAIVE_BAYES_MISSING_TOKEN_PROBABILITY	crash	0.038461538
rear	crash	0.076923077
vehicl	crash	0.076923077
NAIVE_BAYES_PRIOR_PROBABILITY	no_crash	0.411764706
NAIVE_BAYES_MISSING_TOKEN_PROBABILITY	no_crash	0.043478261
NAIVE_BAYES_TEXT_MODEL_TYPE	MULTINOMIAL	1.000000000
frame	no_crash	0.086956522
motor	no_crash	0.086956522
sustain	crash	0.076923077

## TD\_NERExtractor

The TD\_NERExtractor function performs Named Entity Recognition (NER) on input text according to user-defined dictionary words or regular expression (regex) patterns.

TD\_NERExtractor can be used to match and label specific elements within a given input text. The function identifies the contents of the text by matching extracts from the text to specific words (dictionary) or string patterns using regular expressions.

### Function Information

- [TD\\_NERExtractor Syntax](#)
- [Required Syntax Elements for TD\\_NERExtractor](#)
- [Optional Syntax Elements for TD\\_NERExtractor](#)
- [TD\\_NERExtractor Input](#)
- [TD\\_NERExtractor Output](#)
- [Example: How to Use TD\\_NERExtractor](#)

**Note:**

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).  
For information about PTCs, see *International Character Set Support*, B035-1125.
- This function does not support KanjiSJIS or Graphic data types.
- SELECT TOP gives non-deterministic results. Therefore, identical queries including this instruction may produce different results.

## TD\_NERExtractor Syntax

```
TD_NERExtractor (
    ON { table | view | (query) } as InputTable [ PARTITION BY ANY ]
    [ ON { table | view | (query) } as Dict DIMENSION ]
    [ ON { table | view | (query) } as Rules DIMENSION ]
    USING
        TextColumn ('text_column')
        [ InputLanguage ('en') ]
        [ ShowContext (context_num) ]
        [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
)

```

**USING Clause limitation:** The maximum number of rules and dictionary words (rows) is allowed to be up to 90% of the available memory of the cluster in use. There is no fixed limit in the number of rows between the two of them.

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_NERExtractor

**ON Clause**

Accepts the InputTable clause.

The InputTable in TD\_NERExtractor query supports PARTITION BY clause, so it can have no partition at all or have PARTITION BY ANY, or PARTITION BY 'column name'.

**TextColumn**

Specify the name of the input column.

Maximum 1 text column is supported. If more than 1 text column is provided, an error is returned.

## Optional Syntax Elements for TD\_NERExtractor

**ON Clause**

Accepts the Dict and Rules clauses.

**InputLanguage**

Specify the language of the input text.

Default - 'en' (English).

**ShowContext**

Specify the number of "context" words before and after the entity found. In cases where the match is close to the beginning or end of the input text, leading or trailing '...' will be included.

**Accumulate**

Specifies the names of the input columns to copy to the output table.

## TD\_NERExtractor Input

The input text table, dictionary table, and regex rules table can be specified in any order.

- The input text table should be aliased with InputTable.
- The Rules table should be aliased with Rules.
- The dictionary table should be aliased as Dict.
- Rules and Dict table should be DIMENSION. If aliases are not specified, an error is returned.
- If incorrect aliases are passed, an error is returned.

**Note:**

Both UNICODE and LATIN character set are allowed, however, all input tables must share the same character set. For example, if TextColumn is UNICODE, the contents of Dict and Rules table must also be UNICODE.

## InputTable Schema

Column Name	Data Type	Description
text_column	VARCHAR	Column contains input text.
accumulate_column	ANY	Column to copy to output table. CLOB and BLOB columns cannot be part of Accumulate columns.

## Rules Table Schema

Column names for Rules table must be type\_ner and regex; otherwise, an error will be thrown.

Column Name	Data Type	Description
type_ner	VARCHAR	Name of the entity.
regex	VARCHAR	<p>Regular Expression pattern of the entity.</p> <p><b>Note:</b></p> <p>No escape characters are needed for some special characters. For example to find '\$' character, a valid regular expression would be '\\$', and not '\\\$'.</p> <p>The following characters need to be escaped with one backslash for literal match:</p> <ul style="list-style-type: none"> <li>'\` - Backslash</li> <li>'\^ - Caret</li> <li>'\\$' - dollar sign</li> <li>'.' - Period or dot</li> <li>' ' - Vertical bar or pipe</li> <li>'?' - Question mark</li> <li>'*' - Asterisk or star</li> <li>'+' - Plus sign</li> <li>'(' ')' - Opening and closing parenthesis</li> <li>'[' ']' - Opening and closing square bracket</li> <li>'{' '}' - opening and closing curly brackets</li> </ul>

## Dict Table Schema

Column names for Dict table must be type\_ner and dict, otherwise, an error will be thrown.

## TD\_NERExtractor Output

Column Name	Data Type	Description
Accumulate columns	ANY	Columns copied from input to output.
sn	INTEGER	Match number in a given row.

Column Name	Data Type	Description
entity	VARCHAR CHARACTER SET LATIN VARCHAR CHARACTER SET UNICODE	Matched string for a given dictionary word or regex pattern. Column size is fixed to 1000 characters (UNICODE and LATIN). Character set will be the same as input tables.
type	VARCHAR CHARACTER SET LATIN VARCHAR CHARACTER SET UNICODE	User-defined entity name (tag/label) for matched dictionary word or regex pattern. Column size will be the largest of type_ner columns from rules or dict table. Character set will be the same as input tables.
start	INTEGER	Word number in row where the match starts.
end	INTEGER	Word number in row where the match ends.
context	VARCHAR CHARACTER SET LATIN VARCHAR CHARACTER SET UNICODE	[Optional] Only appears when argument ShowContext > 0 is passed. Column size will be the same as TextColumn, up to 16000 characters (UNICODE or LATIN), whichever is smaller. Character set will be the same as input tables.
approach	VARCHAR CHARACTER SET LATIN VARCHAR CHARACTER SET UNICODE	Category of matched string RULE for regex rules, or DICT for dictionary word. Column size if 4 characters fixed (UNICODE or LATIN). Character set will be the same as input tables.

## Example: How to Use TD\_NERExtractor

### Input Table

```

DROP TABLE ner_input_eng;
create multiset table ner_input_eng(
    id INTEGER,
    txt VARCHAR(500) CHARACTER SET LATIN NOT CASESPECIFIC
);
insert into ner_input_eng values (1, 'At end of August, the Janus Unconstrained
fund held only 45 debt issues with 70 percent of its assets in U.S.
government debt.');
insert into ner_input_eng values (2, 'One Treasury issue due June 2016 alone
was worth 43 percent of the fund''s total assets.');
insert into ner_input_eng values (3, 'Most of the bonds have short durations,

```

```

with the average maturity of just over three years, indicating a generally
defensive posture.');
insert into ner_input_eng values (4, 'For Bill Gross, quitting Pimco''s $222
billion Total Return Fund to take over a $13 million fund at Janus Capital is
like resigning the U.S. presidency to become city manager of Ashtabula, Ohio,
population 18,800.');
insert into ner_input_eng values (5, 'Gross stunned the investing world on
Friday with his abrupt departure from Pimco, the $2 trillion asset manager he
co-founded in 1971 and where he had run the Total Return Fund, the world''s
biggest bond fund, for more than 27 years.');
insert into ner_input_eng values (6, '[0-9]+');

```

## Rules Table

```

DROP TABLE ner_rule;
create multiset table ner_rule(
    type_ner VARCHAR(500) CHARACTER SET LATIN NOT CASESPECIFIC,
    regex VARCHAR(500) CHARACTER SET LATIN NOT CASESPECIFIC
);
insert into ner_rule values ('email', '[\w\-\]([\.\.\w])+\[\w\]+\@([\w\-\]+\.)+[a-zA-Z]{2,4}');
insert into ner_rule values ('Money', '\s\$[0-9]+\s');
insert into ner_rule values ('Digits', '\s[0-9]+\s');
insert into ner_rule values ('Name', '[A-Z][a-z]+\s+[A-Z][a-z]+');

```

## Dict Table

```

DROP TABLE ner_dict;
create multiset table ner_dict(
    type_ner VARCHAR(500) CHARACTER SET LATIN NOT CASESPECIFIC,
    dict VARCHAR(500) CHARACTER SET LATIN NOT CASESPECIFIC
);
insert into ner_dict values('location', 'Arkansas');
insert into ner_dict values('location', 'Dublin');
insert into ner_dict values('MISC', ' average maturity');
insert into ner_dict values('location', 'Ohio ');
insert into ner_dict values('month', ' June ');
insert into ner_dict values('Last Name', ' Gross');
insert into ner_dict values('digit regex', '[0-9]+');

```

## Query Statement

Input Table : ner\_input\_eng

Rules Table: ner\_rule

Dict Table: ner\_dict

```
SELECT id, entity, "type", "start", "end", context, approach
FROM TD_NERExtractor(
    ON ner_input_eng as InputTable
    ON ner_rule as rules DIMENSION
    ON ner_dict as dict DIMENSION
    USING
        TextColumn('txt')
        InputLanguage('en')
        ShowContext(3)
        Accumulate('id')
) as dt order by id, "start";
```

Output:

<b>id</b>	<b>entity</b>	<b>type</b>	<b>start</b>	<b>end</b>	<b>context</b>	<b>approach</b>
1	Janus Unconstrained	Name	6	7	of August, the Janus Unconstrained fund held only	RULE
1	45	Digits	11	11	fund held only 45 debt issues with	RULE
1	70	Digits	15	15	debt issues with 70 percent of its	RULE
2	One Treasury	Name	1	2	..... One Treasury issue due June	RULE
2	June	month	5	5	Treasury issue due June 2016 alone was	DICT
2	2016	Digits	6	6	issue due June 2016 alone was worth	RULE
2	43	Digits	10	10	alone was worth 43 percent of the	RULE
3	average maturity	MISC	10	11	durations, with the average maturity of just over	DICT
4	For Bill	Name	1	2	.... For Bill Gross, quitting Pimco's	RULE
4	Gross	Last Name	3	3	... For Bill Gross, quitting Pimco's \$222	DICT
4	\$222	Money	6	4	Gross, quitting Pimco's \$222 billion Total Return	RULE
4	Total Return	Name	8	9	Pimco's \$222 billion Total Return Fund to take	RULE
4	\$13	Money	15	15	take over a \$13 million fund at	RULE

<b>id</b>	<b>entity</b>	<b>type</b>	<b>start</b>	<b>end</b>	<b>context</b>	<b>approach</b>
4	Janus Capital	Name	19	20	million fund at Janus Capital is like resigning	RULE
4	Ohio	location	33	33	manager of Ashtabula, Ohio, population 18,800. ...	DICT
5	Gross	Last Name	1	1	... .... Gross stunned the investing	DICT
5	\$2	Money	15	15	from Pimco, the \$2 trillion asset manager	RULE
5	1971	Digits	22	22	he co-founded in 1971 and where he	RULE
5	Total Return	Name	29	30	had run the Total Return Fund, the world's	RULE
5	27	Digits	40	40	for more than 27 years. ....	RULE
6	[0-9]+	digit regex	1	1	... .... [0-9]+ ... ....	DICT

## TD\_SentimentExtractor

TD\_SentimentExtractor uses a dictionary model to extract the sentiment (positive, negative, or neutral) of each input document or sentence.

The dictionary model consists of WordNet, a lexical database of the English language, and these negation words such as no, not, neither, never, and so on.

The function handles negated sentiments as follows:

- -1 if the sentiment is negated. For example, I am not happy.
- -1 if one word separates the sentiment and a negation word. For example, I am not very happy.
- +1 if two or more words separate the sentiment and a negation word. For example, I am not saying I am happy.
- You can omit the dimension ON clauses of the dictionary tables from the query if you want to use the default sentiment dictionary.
- You can use your dictionary table and provide it as a CustomDictionaryTable ON clause.
- You can provide additional dictionary entries through the AdditionalDictionaryTable ON clause to add more entries to either the CustomDictionaryTable or default dictionary.
- You can access the dictionary through the OutputDictionaryTable OUT clause to check the dictionary contents used during sentiment analysis.
- Only supports the English language.
- The maximum length supported for sentiment word in the dictionary table is 128 characters.

- The maximum length of the sentiment\_words output column is 32000 characters. If the sentiment\_words output column value exceeds this limit, then a ellipsis (...) displays at the end of the string.
- The maximum length of the content output column is 32000 characters; the supported maximum length of a sentence is 32000.
- There can be up to 10 words in a sentiment phrase.

## Function Information

- [TD\\_SentimentExtractor Syntax](#)
- [Required Syntax Elements for TD\\_SentimentExtractor](#)
- [Optional Syntax Elements for TD\\_SentimentExtractor](#)
- [TD\\_SentimentExtractor Input](#)
- [TD\\_SentimentExtractor Output](#)
- [TD\\_SentimentExtractor Usage Notes](#)
- [Examples: How to Use TD\\_SentimentExtractor](#)

## TD\_SentimentExtractor Syntax

```
TD_SentimentExtractor (
    ON { table | view | (query) } AS InputTable PARTITION BY ANY
    [ ON { table | view | (query) } AS CustomDictionaryTable DIMENSION ]
    [ ON { table | view | (query) } AS AdditionalDictionaryTable DIMENSION ]
    [ OUT PERMANENT TABLE OutputDictionaryTable (output_table_name) ]
    USING
        TextColumn ('text_column')
    [ Accumulate ({ 'accumulate_column' | accumulate_column_range }[,...]) ]
    [ AnalysisType ({ 'DOCUMENT' | 'SENTENCE' }) ]
    [ Priority ({ 'NONE' | 'NEGATIVE_RECALL' | 'NEGATIVE_PRECISION' |
        'POSITIVE_RECALL' | 'POSITIVE_PRECISION'}) ]
    [ OutputType ({ 'ALL' | 'POS' | 'NEG' | 'NEU' }) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
-

## Required Syntax Elements for TD\_SentimentExtractor

### ON clause

Accepts the InputTable clause.

### TextColumn

Specifies the input column name that contains text for sentiment analysis.

## Optional Syntax Elements for TD\_SentimentExtractor

### ON clause

Accepts the CustomDictionaryTable, AdditionalDictionaryTable, and OutputDictionaryTable clauses.

### Accumulate

Specifies the input table column names to copy to the output table.

### AnalysisType

Specifies the analysis level to analyze each document or sentence.

Default: Document.

### Priority

Specifies one of the following priorities for results:

- None (Default): Provide all results the same priority.
- Negative\_Recall: Provide the highest priority to negative results, including those with lower-confidence sentiment classifications (maximizes number of negative results returned).
- Negative\_Precision: Provide the highest priority to negative results with high-confidence sentiment classifications.
- Positive\_Recall: Provide the highest priority to positive results, including those with lower-confidence sentiment classifications (maximizes number of positive results returned).
- Positive\_Precision: Provide the highest priority to positive results with high confidence sentiment classifications.

### OutputType

Specifies one of the following result types:

- All (Default): Returns all results.

- Pos: Returns only results with positive sentiments.
- Neg: Returns only results with negative sentiments.
- Neu: Returns only results with neutral sentiments.

## TD\_SentimentExtractor Input

### Input Table Schema

Column	Data Type	Description
text_column	CHAR, VARCHAR, CLOB	The InputTable column name that contains text for sentiment analysis.
accumulate_column	ANY	The InputTable column names to copy to the output table.

### Custom/Additional Dictionary Table Schema

Column	Data Type	Description
sentiment_word	CHAR, VARCHAR	The column name that contains the sentiment word.
polarity_strength	BYTEINT, SMALLINT, INTEGER	The column name that contains the strength of the sentiment word.

## TD\_SentimentExtractor Output

### Output Table Schema

Column	Data Type	Description
AccumulateColumns	ANY	The specified InputTable column names copied to the output table.
content	VARCHAR	The column contains the sentence extracted from the document. The column displays if you use Sentence as the AnalysisType.
polarity	VARCHAR	The sentiment value of the result. Values are POS (positive), NEG (negative), or NEU (neutral).
sentiment_score	INTEGER	The sentiment score of polarity. Values are 0 (neutral), 1 (higher than neutral), or 2 (higher than 1).
sentiment_words	VARCHAR	The string that contains a total positive score, total negative score, and sentiment words with their polarity_strength and frequency enclosed in parenthesis.

## Output Dictionary Table Schema

Column	Data Type	Description
sentiment_word	VARCHAR	The column that contains the sentiment word.
polarity_strength	INTEGER	The column that contains the strength of the sentiment word.

## TD\_SentimentExtractor Usage Notes

Sentiment extraction in machine learning is a natural language processing (NLP) technique used to analyze and classify the emotional tone of text data into positive, negative or neutral categories. This technique involves training a model on labeled data to learn the patterns and features that distinguish between different segments in text.

The main goal of sentiment extraction is to determine the underlying sentiment, emotion, or opinion expressed in each text. This information is valuable for businesses and organizations to understand their customer's preferences, opinions, and reactions toward their products, services, or brand.

There are some common algorithms and techniques for sentiment extraction such as rule-based modeling, machine learning, lexicon-based approaches, and so on.

- Rule-based modeling, defines a set of predefined rules to classify the sentiment of the text data.
- Machine Learning (ML) algorithms use a supervised approach to train a classifier on labeled data.
- Lexicon-based approaches use sentiment lexicons or dictionaries that contain a list of words and their associated sentiment polarity.

Sentiment extraction in machine learning has different applications, including sentiment analysis of customer reviews, social media monitoring, and market research. The accuracy of the sentiment extraction model depends on several factors, such as the quality of the labeled data, the choice of features, and the design of the model. Techniques such as ensemble learning, and transfer learning can be used to improve the performance of the model and generalize to new domains.

To train the sentiment extraction model, pre-processing of the text data is performed by cleaning and tokenizing the text data. Next, labeled data is used for training to recognize the sentiment of the text. This model uses various features, such as word frequency, sentiment lexicons, and part-of-speech tags, to predict the sentiment of the text. The accuracy of the model is evaluated on a validation set and the hyperparameters are tuned to optimize the performance of the model.

However, it is important to note that the accuracy of the sentiment extraction model depends on the quality of the training data, the choice of features, and the design of the model. It is essential to perform rigorous testing and evaluation of the model before deploying it in real-world applications.

## Examples: How to Use TD\_SentimentExtractor

### TD\_SentimentExtractor Input Table: `sentiment_extract_input`

id	product	category	review
...	...	...	...

## 9: Text Analytic Functions

1 camera POS we primarily bought this camera for high image quality and excellent video capability without paying the price for a dslr .it has excelled in what we expected of it , and consequently represented excellent value for me .all my friends want my camera for their vacations . i would recommend this camera to anybody .definitely worth the price .plus , when you buy some accessories , it becomes even more powerful .

2 office suite POS it is the best office suite i have used to date . it is launched before office 2010 and it is ages ahead of it already . the fact that i could comfortable import xls , doc , ppt and modify them , and then export them back to the doc , xls , ppt is terrific . i needed the compatibility .it is a very intuitive suite and the drag drop functionality is terrific .

3 camera POS this is a nice camera , delivering good quality video images decent photos . light small , using easily obtainable , high quality minidv i love it . minor irritations include touchscreen based menu only digital photos can only be transferred via usb , requiring ilink and usb if you use ilink .

4 gps POS it is a fine gps . outstanding performance , works great . you can even get incredible coordinate accuracy from streets and trips to compare .

5 gps POS nice graphs and map route info .i would not run outside again without this unique gadget . great job. big display , good backlight , really watertight , training assistant .i use in trail running and it worked well through out the race

6 gps NEG most of the complaints i have seen in here are from a lack of rtfm. i have never seen so many mistakes do to what i think has to be none update of data to the system . i wish i could make all the rating stars be empty .

7 gps NEG this machine is all screwed up . on my way home from a friends house it told me there is no possible route . i found their website support difficult to navigate . i am is so disappointed and just returned it and now looking for another one

8 camera NEG i hate my camera , and im stuck with it . this camera sucks so bad , even the dealers on ebay have difficulty selling it. horrible indoors , does not capture fast action, screwy software , no suprise , and screwy audio/video codec that does not work with hardly any app

9 television NEG \$3k is way too much money to drop onto a piece of crap .poor customer support . after about 1 and a half years and hardly using the tv , a big yellow pixilated stain appeared. product is very inferior and subject to several lawsuits . i expressed my dissatisfaction with the situation as this is a known issue

10 camera NEG i returned my camera to the vendor as i will not tolerate a sub standard product that is a known issue especially from vendor who will not admst that this needs to be removed from the shelf due to failing parts updated . due to the constant need for repair , i would never recommend this product .

**TD\_SentimentExtractor InputTable: sentiment\_word**

sentiment_word	polarity_strength
big	0
constant	0
crap	-2
difficulty	-1
disappointed	-1
excellent	2
fun	1
incredible	2
love	1
mistake	-1
nice	1
not tolerate	-1
outstanding	2
screwed	2
small	0
stuck	-1
terrific	2
terrible	-2
update	0

**TD\_SentimentExtractor InputTable: sentiment\_word\_add**

sentiment_word	polarity_strength
love	2
need for repair	-2
repair	-1

**Example: TD\_SentimentExtractor SQL Call with Default Dictionary**

```
SELECT * FROM TD_SentimentExtractor (
ON sentiment_extract_input AS InputTable PARTITION BY ANY
USING
TextColumn ('review')
Accumulate ('id', 'product')
AnalysisType ('DOCUMENT')
) AS dt ORDER BY id;
```

## TD\_SentimentExtractor Output with Default Dictionary

```

id product      polarity    sentiment_score sentiment_words
----- -----
1 camera        POS         2             In total, positive score:7 negative score:0. exceeded 1 (1), excellent 1 (2), powerful 1 (1), worth 1 (1), recommend 1 (1), capability 1 (1).
2 office suite  POS         2             In total, positive score:5 negative score:-1. drag -1 (1), intuitive 1 (1), best 1 (1), comfortable 1 (1), terrific 1 (2).
3 camera        POS         2             In total, positive score:5 negative score:-1. decent 1 (1), good 1 (1), irritations -1 (1), nice 1 (1), love 1 (1), obtainable
1 (1).
4 gps           POS         2             In total, positive score:5 negative score:0. incredible 1 (1), outstanding 1 (1), fine 1 (1), great 1 (1), works 1 (1).
5 gps           POS         2             In total, positive score:5 negative score:0. good 1 (1), worked 1 (1), nice 1 (1), great 1 (1), well 1 (1).
6 gps           NEG         2             In total, positive score:0 negative score:-3. lack -1 (1), complaints -1 (1), mistakes
-1 (1).
7 gps           NEG         2             In total, positive score:1 negative score:-3. disappointed -1 (1), doomed -1 (1), difficult -1 (1), support 1 (1).
8 camera        NEG         2             In total, positive score:0 negative score:-10. stuck -1 (1), sucks -1 (1), screwy -1 (2), not fast -1 (1), bad -1 (1), difficulty -1 (1), horrible -1 (1), not work -1 (1), hate
-1 (1).
9 television    NEG         2             In total, positive score:1 negative score:-5. crap -1 (1), issue -1 (1), stain -1 (1), inferior -1 (1), poor -1 (1), support
1 (1).
10 camera       NEG        2             In total, positive score:0 negative score:-3. failing -1 (1), issue -1 (1), never recommend -1 (1).

```

## Example: TD\_SentimentExtractor SQL Call with Custom Dictionary

```

SELECT * FROM TD_SentimentExtractor (
ON sentiment_extract_input AS InputTable PARTITION BY ANY
ON sentiment_word AS CustomDictionaryTable DIMENSION
USING
TextColumn ('review')
Accumulate ('id', 'product')
AnalysisType ('SENTENCE')
) AS dt ORDER BY id;

```

## TD\_SentimentExtractor Output with Custom Dictionary

```

id product      content      polarity    sentiment_score    sentiment_words
----- -----
1 camera       i would recommend this camera to anybody .definitely worth the price .plus , when you buy some accessories , it
becomes even more powerful          NEU         0
1 camera       we primarily bought this camera for high image quality and excellent video capability without paying the price for a
dslr .it has excelled in what we expected of it , and consequently represented excellent value for me .all my friends want my camera
for their vacations .          POS         2             In total, positive score:4 negative score:0. excellent 2 (2).
2 office suite the fact that i could comfortable import xls , doc , ppt and modify them , and then export them back to the doc ,
xls , ppt is terrific .          POS         2             In total, positive score:2 negative score:0. terrific 2 (1).
2 office suite i needed the compatibility .it is a very intuitive suite and the drag drop functionality is terrific .
          POS         2             In total, positive score:2 negative score:0. terrific 2 (1).
2 office suite it is launched before office 2010 and it is ages ahead of it already .
          NEU         0
2 office suite it is the best office suite i have used to date .
          NEU         0
3 camera       minor irritations include touchscreen based menu only digital photos can only be transferred via usb , requiring ilink
and usb if you use ilink .          NEU         0
3 camera       light small , using easily obtainable , high quality minidv i love it .
          POS         2             In total, positive score:1 negative score:0. small 0 (1), love 1 (1).
3 camera       this is a nice camera , delivering good quality video images decent photos .
          POS         2             In total, positive score:1 negative score:0. nice 1 (1).
4 gps          it is a fine gps .
          NEU         0
4 gps          outstanding performance , works great .
          POS         2             In total, positive score:2 negative score:0. outstanding 2 (1).
4 gps          you can even get incredible coordinate accuracy from streets and trips to compare .
          POS         2             In total, positive score:2 negative score:0. incredible 2 (1).
5 gps          big display , good backlight , really watertight , training assistant .i use in trail running and it worked well
through out the race          NEU         0             In total, positive score:0 negative score:0. big 0 (1).
5 gps          great job .
          NEU         0
5 gps          nice graphs and map route info .i would not run outside again without this unique gadget .
          POS         2             In total, positive score:1 negative score:0. nice 1 (1).
6 gps          i wish i could make all the rating stars be empty .

```

```

          NEU      0
6 gps     i have never seen so many mistakes do to what i think has to be none update of data to the system .
          NEU      0           In total, positive score:0 negative score:0. update 0 (1).
6 gps     most of the complaints i have seen in here are from a lack of rtfm.
          NEU      0

7 gps     i found their website support difficult to navigate .
          NEU      0
7 gps     i am is so disapointed and just returned it and now looking for another one
          NEU      0

7 gps     on my way home from a friends house it told me there is no possible route .
          NEU      0
7 gps     this machine is all screwed up .
          POS      2           In total, positive score:2 negative score:0. screwed 2 (1).
8 camera   this camera sucks so bad , even the dealers on ebay have difficulty selling it.
          NEG      2           In total, positive score:0 negative score:-1. difficulty -1 (1).
8 camera   i hate my camera , and im stuck with it
          NEG      2           In total, positive score:0 negative score:-1. stuck -1 (1).
8 camera   horrible indoors , does not capture fast action, screwy software , no suprise , and screwy audio/video codec that does
not work with hardly any app
          NEU      0
9 television product is very inferior and subject to several lawsuits .
          NEU      0
9 television i expressed my dissatisfaction with the situation as this is a known issue
          NEU      0
9 television after about 1 and a half years and hardly using the tv , a big yellow pixilated stain appeared.
          NEU      0           In total, positive score:0 negative score:0. big 0 (1).
9 television $3k is way too much money to drop onto a piece of crap .poor customer support .
          NEG      2           In total, positive score:0 negative score:-2. crap -2 (1).
10 camera   due to the constant need for repair , i would never recommend this product .
          NEU      0           In total, positive score:0 negative score:0. constant 0 (1).
10 camera   i returned my camera to the vendor as i will not tolerate a sub standard product that is a known issue especially from
vendor who will not admitt that this needs to be removed from the shelf due to failing parts updated .
          NEG      2           In total, positive score:0 negative score:-1. not tolerate -1 (1).

```

## Example: TD\_SentimentExtractor SQL Call with Default Dictionary and Additional Dictionary

```

SELECT * FROM TD_SentimentExtractor (
ON sentiment_extract_input AS InputTable PARTITION BY ANY
ON sentiment_word_add AS AdditionalDictionaryTable DIMENSION
USING
TextColumn ('review')
Accumulate ('id', 'product')
AnalysisType ('DOCUMENT')
) AS dt order by id;

```

## TD\_SentimentExtractor Output with Default Dictionary and Additional Dictionary

id	product	polarity	sentiment_score	sentiment_words
1	camera	POS	2	In total, positive score:7 negative score:0. excelled 1 (1), excellent 1 (2), powerful 1 (1), worth 1 (1), recommend 1 (1), capability 1 (1).
2	office suite	POS	2	In total, positive score:5 negative score:-1. drag -1 (1), intuitive 1 (1), best 1 (1), comfortable 1 (1), terrific 1 (2).
3	camera	POS	2	In total, positive score:6 negative score:-1. decent 1 (1), good 1 (1), irritations -1 (1), nice 1 (1), love 2 (1), obtainable 1 (1).
4	gps	POS	2	In total, positive score:5 negative score:0. incredible 1 (1), outstanding 1 (1), fine 1 (1), great 1 (1), works 1 (1).
5	gps	POS	2	In total, positive score:5 negative score:0. good 1 (1), worked 1 (1), nice 1 (1), great 1 (1), well 1 (1).
6	gps	NEG	2	In total, positive score:0 negative score:-3. lack -1 (1), complaints -1 (1), mistakes -1 (1).
7	gps	NEG	2	In total, positive score:1 negative score:-3. disappointed -1 (1), screwed -1 (1), difficult -1 (1), support 1 (1).
8	camera	NEG	2	In total, positive score:0 negative score:-10. stuck -1 (1), sucks -1 (1), screwy -1 (2), not fast -1 (1), bad -1 (1), difficulty -1 (1), horrible -1 (1), not work -1 (1), hate -1 (1).
9	television	NEG	2	In total, positive score:1 negative score:-5. crap -1 (1), issue -1 (1), stain -1 (1), inferior -1 (1), poor -1 (1), support 1 (1).
10	camera	NEG	2	In total, positive score:0 negative score:-5. failing -1 (1), need for repair -2 (1), issue -1 (1), never recommend -1 (1).

## TD\_TextMorph

The TD\_TextMorph function generates morphs (standard form/Dictionary form) of the given tokens in the input dataset using lemmatization algorithm based on English Dictionary. You can specify part of speech (POS) to generate morphs of specified POS for the given input token. This function also supports single output such that if the input token has multiple morphs of different part of speech, it will generate only one output.

### Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).  
For information about PTCs, see *International Character Set Support*, B035-1125.
- This function does not support KanjiSJIS, Graphic, or character large object (CLOB) data types.

### Function Information

- [TD\\_TextMorph Syntax](#)
- [Required Syntax Elements for TD\\_TextMorph](#)
- [Optional Syntax Elements for TD\\_TextMorph](#)
- [TD\\_TextMorph Input](#)
- [TD\\_TextMorph Output](#)
- [Examples: How to Use TD\\_TextMorph](#)

## TD\_TextMorph Syntax

```
TD_TextMorph (
  ON {table | view | (query)} as InputTable
  USING
    WordColumn ('word_column')
    [POSTagColumn ('pos_tag_column')]
    [SingleOutput({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'})]
    [POS({'noun' | 'verb' | 'adj' | 'adv'})]
    [Accumulate({'accumulate_column' | 'accumulate_column_range'}[, ...])]
)
```

---

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_TextMorph

The following element is required when using TD\_TextMorph.

**ON clause**

Accept the InputTable clause.

**Word Column**

Specify the name of the input column that contains words for which morphs to be generated. If WordColumn is not specified, an error is returned.

Maximum 1 column is allowed.

---

**Note:**

WordColumn supports only English language with LATIN or UNICODE characters input. Otherwise, an error is returned.

---

## Optional Syntax Elements for TD\_TextMorph

The following elements are optional when using TD\_TextMorph

**POSTagColumn**

Specify the name of the input table column that contains the part of speech (POS) tags of the words.

If you specify this syntax element, the function outputs each morph according to its POS tag.

Maximum 1 column is allowed.

---

**Note:**

POSTagColumn supports only English language with LATIN or UNICODE characters input.

---

**SingleOutput**

Specify whether to output only one morph for each word. If you specify 'false', the function outputs all morphs for each word.

Default: true.

**POS**

Specify the parts of speech to output. A pos can be 'noun', 'verb', 'adj', or 'adv'. Specification order is irrelevant, this is the order of precedence: 'noun', 'verb', 'adj', 'adv'. If you specify this syntax element and SingleOutput ('true'), then the function outputs only the first pos.

The function does not determine the part of speech of the word from its context, it uses all possible parts of speech for the word in the dictionary

Default: All parts of speech.

**Accumulate**

Specify the names of the input columns to copy to the output table.

- Maximum 2043 names are allowed (with POSTagColumn argument)
- Maximum 2044 names are allowed (without POSTagColumn argument)

**TD\_TextMorph Input****Input Table Schema**

Column Name	Data Type	Description
word_column	VARCHAR (LATIN or UNICODE)	The input tokens for which morphs to be generated.
pos_tag_column	VARCHAR (LATIN or UNICODE)	The POS tag of word. pos_tag_column column in inputTable is required when POSTagColumn argument is passed. The following POSTag Table summarizes the English POSTagger tags and gives their corresponding TextMorph parts of speech.
Accumulate	ANY	Column to copy to output table.

**POSTag Table**

Tag Number	POSTagger Tag	Description	Examples	TextMorph POS
1	CC	Coordinating conjunction	and	
2	CD	Cardinal number	1, third	

Tag Number	POSTagger Tag	Description	Examples	TextMorph POS
3	DT	Determiner	the	
4	EX	Existential there	there is	
5	FW	Foreign word	hors d'oeuvre	
6	IN	Preposition or coordinating conjunction	in, of, like	
7	JJ	Adjective	green	adj
8	JJR	Adjective, comparative	greener	adj
9	JJB	Adjective, superlative	greenest	adj
10	LS	List item marker	1.	
11	MD	Modal	could, will	
12	NN	Noun, singular or mass	table	noun
13	NNS	Noun, plural	tables	noun
14	NNP	Noun, proper, singular	John	noun
15	NNPS	Noun, proper, plural	Vikings	noun
16	PDT	Predeterminer	both boys	
17	POS	Possessive	friend's	noun
18	PRP	Pronoun, personal	I, he, it	
19	PRPS	Pronoun, possessive	my, his	
20	RB	Adverb	however, usually, normally, here, good	adv
21	RBR	Adverb, comparative	better	adv
22	RBS	Adverb, superlative	best	adv
23	RP	Particle	give up	
24	SYM	Symbol		
25	TO	to	to go, to him	
26	UH	Interjection	uh, huh, hmm	
27	VB	Verb, base form	take	verb
28	VBD	Verb, past tense	took	verb

Tag Number	POSTagger Tag	Description	Examples	TextMorph POS
29	VBG	Verb, gerund or present participle	taking	verb
30	VBN	Verb, past participle	taken	verb
31	VBP	Verb, non-third-person singular present tense	take	verb
32	VBZ	Verb, third-person singular present tense	takes	
33	WDT	Wh- determiner	which	
34	WP	Wh- pronoun	who, what	
35	WP	Wh- pronoun, possessive	whose	
36	WRB	Wh- adverb	where, when	

## TD\_TextMorph Output

Column Name	Data Type	Description
Accumulate columns	ANY	Column copied from input to output.
word_column	VARCHAR	Tokens/words for which morphs to be generated.
TD_Morph	VARCHAR	Standard form or Dictionary form of the given tokens generated based on English Dictionary.
POS	VARCHAR	A pos can be 'noun', 'verb', 'adj', or 'adv'.

## Examples: How to Use TD\_TextMorph

This section shows the input table, SQL query, and output tables of examples using TD\_TextMorph.

- [TD\\_TextMorph Examples Input](#)
- [Example 1: TD\\_TextMorph with SingleOutput set to 'True'](#)
- [Example 2: TD\\_TextMorph with SingleOutput set to 'False'](#)
- [Example 3: TD\\_TextMorph with POS Argument Specified and SingleOutput Set to 'False'](#)
- [Example 4: TD\\_TextMorph with POS Argument Specified and SingleOutput Set to 'True'](#)
- [Example 5: TD\\_TextMorph with POSTagColumn Argument Specified](#)

## TD\_TextMorph Examples Input

Except for [Example 5](#), the following input applies to the examples.

```

CREATE TABLE words_input (
    id INTEGER,
    word VARCHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC)
PRIMARY INDEX ( id );
insert into words_input values(1, 'regression');
insert into words_input values(2, 'Roger');
insert into words_input values(3, 'better');
insert into words_input values(4, 'datum');
insert into words_input values(5, 'quickly');
insert into words_input values(6, 'proud');
insert into words_input values(7, 'father');
insert into words_input values(8, 'juniors');
insert into words_input values(9, 'doing');
insert into words_input values(10, 'being');
insert into words_input values(11, 'negating');
insert into words_input values(12, 'yearly');

```

### **InputTable: words\_input**

<b>id</b>	<b>word</b>
1	regression
2	Roger
3	better
4	datum
5	quickly
6	proud
7	father
8	juniors
9	doing
10	being
11	negating
12	yearly

## Example 1: TD\_TextMorph with SingleOutput set to 'True'

```
SELECT * FROM TD_TextMorph(
ON words_input as inputtable
USING
WordColumn('word')
SingleOutput('True')
accumulate('id')
)as dt order by 1,2;
```

Result:

<b>id</b>	<b>word</b>	<b>TD_Morph</b>	<b>POS</b>
1	regression	regression	NOUN
2	roger	Roger	NULL
3	better	better	NOUN
4	datum	datum	NOUN
5	quickly	quickly	ADV
6	proud	proud	ADJ
7	father	father	NOUN
8	juniors	junior	NOUN
9	doing	do	VERB
10	being	being	NOUN
11	negating	negate	VERB
12	yearly	yearly	NOUN

## Example 2: TD\_TextMorph with SingleOutput set to 'False'

```
SELECT * FROM TD_TextMorph (
ON words_input as inputtable
USING
WordColumn('word')
SingleOutput('false')
Accumulate('id')
) AS dt order by 1,2;
```

Result:

<b>id</b>	<b>word</b>	<b>TD_Morph</b>	<b>POS</b>
1	regression	regression	NOUN
2	Roger	Roger	NULL
3	better	better	VERB
3	better	well	ADV
3	better	good	ADJ
3	better	well	ADJ
3	better	better	NOUN
4	datum	datum	NOUN
5	quickly	quickly	ADV
6	proud	proud	ADJ
7	father	father	NOUN
7	father	father	VERB
8	juniors	junior	NOUN
9	doing	do	VERB
10	being	be	VERB
10	being	being	NOUN
11	negating	negate	VERB
12	yearly	yearly	ADV
12	yearly	yearly	ADJ
12	yearly	yearly	NOUN

### Example 3: TD\_TextMorph with POS Argument Specified and SingleOutput Set to 'False'

```
SELECT * FROM TD_TextMorph (
  ON words_input as inputtable
  USING
    WordColumn('word')
    SingleOutput('false')
    POS('noun', 'verb')
```

```
Accumulate('id')
) AS dt order by 1,2;
```

Result:

<b>id</b>	<b>word</b>	<b>TD_Morph</b>	<b>POS</b>
1	regression	regression	NOUN
3	better	better	VERB
3	better	better	NOUN
4	datum	datum	NOUN
7	father	father	NOUN
7	father	father	VERB
8	juniors	junior	NOUN
9	doing	do	VERB
10	being	be	VERB
10	being	being	NOUN
11	negating	negate	VERB
12	yearly	yearly	NOUN

#### Example 4: TD\_TextMorph with POS Argument Specified and SingleOutput Set to 'True'

```
SELECT * FROM TD_TextMorph (
  ON words_input as inputtable
  USING
    WordColumn('word')
    SingleOutput('true')
    POS('noun', 'verb')
    Accumulate('id')
) AS dt order by 1,2;
```

Result:

<b>id</b>	<b>word</b>	<b>TD_Morph</b>	<b>POS</b>
1	regression	regression	NOUN

<b>id</b>	<b>word</b>	<b>TD_Morph</b>	<b>POS</b>
3	better	better	NOUN
4	datum	datum	NOUN
7	father	father	NOUN
8	juniors	junior	NOUN
9	doing	do	VERB
10	being	being	NOUN
11	negating	negate	VERB
12	yearly	yearly	NOUN

## Example 5: TD\_TextMorph with POSTagColumn Argument Specified

### Input

<b>id</b>	<b>word</b>	<b>pos_tag</b>
3	born	VBN
5	8	CD
6	august	NN
7	1981	CD
9	is	VBZ
10	a	DT
11	greatest	JJS
12	tennis	NN
13	player	NN
14	,	O
15	who	WP
17	been	VBN
18	continuously	RB
19	ranked	VBN
20	inside	IN
21	the	DT

<b>id</b>	<b>word</b>	<b>pos_tag</b>
22	top	JJ
23	10	CD
24	since	IN
25	october	JJ
26	2002	CD
27	and	CC
28	has	VBZ
29	won	VBN
30	wimbledon	NN
31	,	O
33	,	O
34	australian	JJ
35	and	CC
36	frenchopen	JJ
37	titles	NNS
38	multiple	JJ
39	times	NNS
32	usopen	JJ
16	has	VBZ
8	,	O
4	on	IN
2	federer	NN
1	roger	NN

## Query

```
SELECT * FROM TD_TextMorph (
  ON pos_input as inputTable
  USING
    WordColumn ('word')
    POSTagColumn ('pos_tag')
```

```
Accumulate ('id', 'pos_tag')
) AS dt;
```

## Output

<b>id</b>	<b>pos_tag</b>	<b>word</b>	<b>TD_Morph</b>	<b>POS</b>
1	NN	roger	roger	NOUN
2	NN	federer	federer	NOUN
3	VBN	born	bear	VERB
4	IN	on	on	NULL
5	CD	8	8	NULL
6	NN	august	august	NOUN
7	CD	1981	1981	NULL
8	O	,	,	NULL
9	VBZ	is	be	VERB
10	DT	a	a	NULL
11	JJS	greatest	great	ADJ
12	NN	tennis	tennis	NOUN
13	NN	player	player	NOUN
14	O	,	,	NULL
15	WP	who	who	NULL
16	VBZ	has	have	VERB
17	VBN	been	be	VERB
18	RB	continuously	continuously	ADV
19	VBN	ranked	rank	VERB
20	IN	inside	inside	NULL
21	DT	the	the	NULL
22	JJ	top	top	ADJ
23	CD	10	10	NULL
24	IN	since	since	NULL
25	JJ	october	october	ADJ
26	CD	2002	2002	NULL

<b>id</b>	<b>pos_tag</b>	<b>word</b>	<b>TD_Morph</b>	<b>POS</b>
27	CC	and	and	NULL
28	VBZ	has	have	VERB
29	VBN	won	win	VERB
30	NN	wimbledon	wimbledon	NOUN
31	O	,	,	NULL
32	JJ	usopen	usopen	ADJ
33	O	,	,	NULL
34	JJ	australian	australian	ADJ
35	CC	and	and	NULL
36	JJ	frenchopen	frenchopen	ADJ
37	NNS	titles	title	NOUN
38	JJ	multiple	multiple	ADJ
39	NNS	times	time	NOUN

## TD\_TextParser

A text parser, also known as a text tokenizer, is a software component that breaks a text into its constituent parts, such as words, phrases, sentences, or other meaningful units. Text parsing is an important technique in natural language processing (NLP) and is used in a wide range of applications, from search engines and chatbots to email filters and data analysis tools.

In text analytics, a text parser is often used as the first step in processing text data to extract useful insights. By breaking the text into smaller units, a parser makes it easier to analyze the text and identify patterns, trends, and relationships among the data.

Text parsers can be simple or complex, depending on the type of text data being processed and the level of detail required for analysis. For example, a basic text parser might split a sentence into individual words, while a more advanced parser might recognize parts of speech, identify named entities, or recognize patterns in the text that suggest a particular sentiment or tone.

By breaking text into its constituent parts and analyzing its structure, text parsers enable a variety of tasks, from information extraction and sentiment analysis to machine translation and chatbot dialog generation. Overall, text parser function is a powerful tool for extracting structured information from unstructured or semi-structured text data, making it easier for analysts and data scientists to work with large amounts of text data and gain insights from it.

The TD\_TextParser performs the following operations:

- This function tokenizes a text with single-character delimiter values, or through using a PCRE regular expression as the token delimiter to parse it.
- Removes the punctuations from the text and converts the text to lowercase
- Removes stop words from the text and converts the text to their root forms
- Creates a row for each word in the output table
- Performs stemming; that is, the function identifies the common root form of a word by removing or replacing word suffixes
- Counts the occurrences of each token or stem
- Obtains a comma separated list of positions for each token occurrence
- Outputs all parsed tokens in a single row

**Note:**

The stems resulting from stemming may not be actual words. For example, the stem for 'communicate' is 'commun' and the stem for 'early' is 'earli' (trailing 'y' is replaced by 'i').

**Function Information**

- [TD\\_TextParser Syntax](#)
- [Required Syntax Elements for TD\\_TextParser](#)
- [Optional Syntax Elements for TD\\_TextParser](#)
- [TD\\_TextParser Input](#)
- [TD\\_TextParser Output](#)
- [TD\\_TextParser Usage Notes](#)
- [Examples: How to Use TD\\_TextParser](#)

**TD\_TextParser Syntax**

```
TD_TextParser (
    ON { table | view | (query) } AS InputTable
    [ON {table | view | (query)} AS StopWordsTable DIMENSION]
    USING
    TextColumn ('text_column')

    [ConvertToLowercase ({'true'|'t'|'yes'||'y'||'1'||'false'||'f'||'no'||'n'||'0'})]
    [StemTokens ({'true'|'t'|'yes'||'y'||'1'||'false'||'f'||'no'||'n'||'0'})]
    [RemoveStopwords ({'true'|'t'|'yes'||'y'||'1'||'false'||'f'||'no'||'n'||'0'})]
    [Delimiter ('delimiter_expression')]
        [DelimiterRegex ('delimiter_PCRE_regular_expression')]
    [Punctuation ('punctuation_expression')]
    [TokenColumnName ('token_column')]
    [Accumulate ({'accumulate_column' | 'accumulate_column_range'} [, ...])]
```

```
[DocIDColumn('docIdColumn')]
[ListPositions ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
[TokenFrequency ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
[OutputByWord ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_TextParser

**ON clause**

Accept the InputTable clause.

**TextColumn**

Specify the input table column name.

## Optional Syntax Elements for TD\_TextParser

**ON clause**

Accept the StopWordsTable clause.

**ConvertToLowerCase**

Convert the text in the input table column name to lowercase.

When StemTokens is set to 'true', TD\_TextParser behaves as if ConvertToLowerCase had the value 'true' regardless of the actual value.

Default value: true

**StemTokens**

Convert the text in the input table column name to their root forms.

Default value: false

**Delimiter**

Specify single-character delimiter values to apply to the text in the specified column in the TextColumn element.

Default values: '\t\n\f\r'

#### **DelimiterRegex**

Specifies a PCRE regular expression that represents the token delimiter.

No default value, when used, the user must provide a valid PCRE regex.

#### **RemoveStopWords**

Specify the value, true to remove the stop words before parsing the text in the specified column in the TextColumn element.

Default value: false

#### **Punctuation**

Specify the punctuation characters that you want to replace in the text of the specified column in the TextColumn element with space.

Default values: '!#\$%&()\*+,-./:;?@|^\_`{|}~'

#### **TokenColName**

Specify a name for the output column that contains the individual words from the text of the specified column in the TextColumn element.

Default value: token

#### **Accumulate**

Specify the input table column names to copy to the output table.

#### **DocIdColumn**

Specify the column name containing the unique identifier of input rows.

If ListPositions is 'true' and/or TokenFrequency is 'true' then DocIdColumn is required only if OutputByWord is 'true'.

#### **ListPositions**

Specify whether to output a list of comma separated positions for each occurrence of a token. The list is arranged in ascending order. The function ignores this argument if OutputByWord has the value 'false'.

Default value: false. The function outputs a row for each occurrence of the word.

#### **TokenFrequency**

Specify whether to output a count of the total of occurrences for each token.

TD\_TextParser ignores this argument if OutputByWord has the value 'false'.

Default value: false.

#### **OutputByWord**

Specifies whether to output all tokens in a single cell ('false') or each token in a separate row ('true').

Default value: true.

## **TD\_TextParser Input**

#### **Note:**

CLOB LATIN/UTF16 is only supported on the Block File System on the primary cluster. It is not available for the Object File System.

#### **InputTable Schema**

Column	Data Type	Description
text_column	CHAR/CLOB/VARCHAR CHARACTER SET LATIN /UNICODE	The column name that contains the text to parse.
accumulate_column	Any	The input table column names to copy to the output table.
docIdColumn	BYTEINT, SMALLINT, INTEGER, BIGINT, CHAR/VARCHAR	Unique identifier of input rows.

#### **StopWordsTable Schema**

Column	Data Type	Description
words	CHAR/CLOB/VARCHAR CHARACTER SET LATIN/UNICODE	The column name that contains the stopwords.

## **TD\_TextParser Output**

#### **Output Table Schema**

Column	Data Type	Description
docIdColumn	BYTEINT, SMALLINT, INTEGER,	Unique identifier of input rows. If provided, it is always the first column in output table.

Column	Data Type	Description
	BIGINT, CHAR /VARCHAR	
AccumulateColumns	ANY	Columns to be copied from input to output. Default: All input columns are copied to output.
TokenColumn	VARCHAR	Column containing individual tokens.
frequency	INTEGER	(Optional) Value indicating the total occurrences of a token.
locations	VARCHAR, BIGINT	(Optional) Comma separated list of values, sorted in ascending order. When ListPositions is set to false and OutputByWord is true, then the position in the input text of each single token is displayed in a separate row. However, the type for this column is BIGINT instead.
tokens	CHAR/VARCHAR /CLOB	(Optional) When OutputByWord argument is 'false': Space separated list of tokens in a single cell.

## TD\_TextParser Usage Notes

- The 'locations' column will accommodate a list of comma separated values up to the maximum capacity of a VARCHAR 64000 column. Once this limit is reached or if the number of digits to be written exceeds it, any additional values will be ignored and not included in the output.
- TokenColName has the following limitations when using ListPositions and TokenFrequency:
  - Cannot have spaces.
  - Do not use any reserved SQL word
- In TokenColName, do not use column names which require the use of double quotations, e.g., containing special characters, reserved keywords, numerics only are not allowed for TokenColName. Some examples of names not allowed: '1,2,3' has numbers and special characters, '123' has only numbers, 'Order' is a reserved keyword.
- When StemTokens('true') now the function ignores ConvertToLowerCase and behaves as if it had the default value 'true'. Instead of previous behavior: Error in function TD\_TextParser: ConvertToLowerCase needs to be true for stemming.
- You cannot use Delimiter and DelimiterRegex at the same time.
- If neither Delimiter nor DelimiterRegex are provided, default Delimiter value is used for tokenization.
- When using DelimiterRegex, empty tokens are not part of the output and are silently discarded.

## Examples: How to Use TD\_TextParser

- [Example 1: Using TD\\_TextParser with StopWords Table](#)
- [Example 2: Using TD\\_TextParser to Get Token Frequency and List of Positions](#)

## Example 1: Using TD\_TextParser with StopWords Table

### Input table: stopwords

```
word
-----
a
an
the
```

### Input table: test\_table

```
id paragraph
-----
1 Programmers program with programming languages
2 The quick brown fox jumps over the lazy dog
```

### SQL Call

```
SELECT * FROM TD_TextParser (
ON test_table AS InputTable
ON stopwords As StopWordsTable DIMENSION
USING
TextColumn ('paragraph')
StemTokens ('true')
RemoveStopWords ('true')
Accumulate ('id')
) as dt ORDER BY id,token;
```

The query performs the following operations:

- Removes the stopwords from the text in the Paragraph column
- Splits the text in the Paragraph column and creates a row for each word in the output table
- Copies the ID column from the input table to the output table

### Output Table

```
id token
-----
1 languag
1 program
1 program
1 programm
```

```

1 with
2 brown
2 dog
2 fox
2 jump
2 lazi
2 over
2 quick

```

## Example 2: Using TD\_TextParser to Get Token Frequency and List of Positions

### **InputTable: test\_table data**

```

CREATE TABLE test_table (
    id INTEGER, paragraph VARCHAR(100)
);
INSERT INTO test_table (id, paragraph) VALUES(1, 'Programmers program with
program, as.as programming languages a program');
INSERT INTO test_table (id, paragraph) VALUES(2, 'The quick brown fox jumps
over the lazy dog');

```

### **SELECT Statement**

```
SELECT * from test_table;
```

Result:

paragraph	-----
-----	
Programmers program with program, as.as programming languages a program	
The quick brown fox jumps over the lazy dog	

### **StopWords table: Custom set of words to be removed when parsing**

```

CREATE TABLE stopwords (word varchar(10));
INSERT INTO stopwords('a');
INSERT INTO stopwords('an');
INSERT INTO stopwords('and');
INSERT INTO stopwords('the');

```

## SELECT Statement

```
SELECT * from stopwords
```

Result:

```
word
-----
the
and
an
a
```

## Query 1 (Tokenizing with default delimiter)

```
SELECT * FROM TD_TextParser (
ON test_table AS InputTable
USING
TextColumn ('paragraph')
RemoveStopWords ('true')
) as dt ORDER BY 1,4
```

Result:

<b>id</b>	<b>paragraph</b>	<b>token</b>	<b>locations</b>
1	Programmers program with program, as.as programming languages a program	programmers	1
1	Programmers program with program, as.as programming languages a program	program	2
1	Programmers program with program, as.as programming languages a program	programming	5
1	Programmers program with program, as.as programming languages a program	language	6
1	Programmers program with program, as.as programming languages a program	program	8
2	The quick brown fox jumps over the lazy dog	quick	1
2	The quick brown fox jumps over the lazy dog	brown	2
2	The quick brown fox jumps over the lazy dog	fox	3
2	The quick brown fox jumps over the lazy dog	jumps	4
2	The quick brown fox jumps over the lazy dog	over	5

<b>id</b>	<b>paragraph</b>	<b>token</b>	<b>locations</b>
2	The quick brown fox jumps over the lazy dog	lazy	7
2	The quick brown fox jumps over the lazy dog	dog	8

**Query 2 (Using StopWordsTable, ListPositions, and TokenFrequency)**

```
SELECT * FROM TD_TextParser (
ON test_table AS InputTable
ON stopwords as StopWordsTable DIMENSION
USING
TextColumn ('paragraph')
RemoveStopWords ('true')
DocIDColumn('id')
ListPositions('t')
TokenFrequency('t')
) as dt ORDER BY 1,2
```

Result:

<b>id</b>	<b>paragraph</b>	<b>token</b>	<b>frequency</b>	<b>locations</b>
1	Programmers program with program, as.as programming languages a program	programmers	1	1
1	Programmers program with program, as.as programming languages a program	program	2	2,8
1	Programmers program with program, as.as programming languages a program	with	1	3
1	Programmers program with program, as.as programming languages a program	the	1	4
1	Programmers program with program, as.as programming languages a program	programming	1	5
1	Programmers program with program, as.as programming languages a program	language	1	6
2	The quick brown fox jumps over the lazy dog	the	2	0,6
2	The quick brown fox jumps over the lazy dog	quick	1	1
2	The quick brown fox jumps over the lazy dog	brown	1	2
2	The quick brown fox jumps over the lazy dog	fox	1	3
2	The quick brown fox jumps over the lazy dog	jumps	1	4
2	The quick brown fox jumps over the lazy dog	over	1	5

<b>id</b>	<b>paragraph</b>	<b>token</b>	<b>frequency</b>	<b>locations</b>
2	The quick brown fox jumps over the lazy dog	lazy	1	7
2	The quick brown fox jumps over the lazy dog	dog	1	8

**Query 3 (Using OutputByWords set to false and Delimiter is a blank space)**

```
SELECT * FROM TD_TextParser (
ON test_table AS InputTable
ON stopwords as StopWordsTable DIMENSION
USING
TextColumn ('paragraph')
RemoveStopWords ('true')
Delimiter(' ')
OutputByWord('false')
) as dt ORDER BY 1,2
```

Result:

<b>id</b>	<b>paragraph</b>	<b>tokens</b>
1	Programmers program with program, as.as programming languages a program	programmers program program programming languages program
2	The quick brown fox jumps over the lazy dog	quick brown fox jumps over lazy dog

**Query 4 (Using DelimiterRegex)**

```
SELECT * FROM TD_TextParser (
ON test_table AS InputTable
USING
TextColumn ('paragraph')
RemoveStopWords ('true')
DocIDColumn('id')
DelimiterRegex('[ \t\f\r\n]+')
ListPositions('true')
) as dt ORDER BY 1,4
```

Result:

<b>id</b>	<b>paragraph</b>	<b>tokens</b>	<b>locations</b>
1	Programmers program with program, as.as programming languages a program	programmers	1

<b>id</b>	<b>paragraph</b>	<b>tokens</b>	<b>locations</b>
1	Programmers program with program, as.as programming languages a program	program	2,8
1	Programmers program with program, as.as programming languages a program	programming	5
1	Programmers program with program, as.as programming languages a program	language	6
2	The quick brown fox jumps over the lazy dog	quick	1
2	The quick brown fox jumps over the lazy dog	brown	2
2	The quick brown fox jumps over the lazy dog	fox	3
2	The quick brown fox jumps over the lazy dog	jumps	4
2	The quick brown fox jumps over the lazy dog	over	5
2	The quick brown fox jumps over the lazy dog	lazy	7
2	The quick brown fox jumps over the lazy dog	dog	8

## TD\_TFIDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a technique for evaluating the importance of a specific term in a specific document in a document set. Term frequency (tf) is the number of times that the term appears in the document and inverse document frequency (idf) is the number of times that the term appears in the document set. The TF-IDF score for a term is  $tf * idf$ . A term with a high TF-IDF score is relevant to the specific document.

You can use the TF-IDF scores as input for documents clustering and classification algorithms, including:

- Cosine-similarity
- Latent Dirichlet allocation
- K-means clustering
- K-nearest neighbors

TD\_TFIDF function represents each document as an N-dimensional vector, where N is the number of terms in the document set (therefore, the document vector is sparse). Each entry in the document vector is the TF-IDF score of a term.

### Function Information

- [TD\\_TFIDF Syntax](#)
- [Required Syntax Elements for TD\\_TFIDF](#)
- [Optional Syntax Elements for TD\\_TFIDF](#)
- [TD\\_TFIDF Input](#)

- [TD\\_TFIDF Output](#)
- [Example: How to Use TD\\_TFIDF](#)

## TD\_TFIDF Syntax

```
TD_TFIDF (
    ON { table | view | (query) } AS InputTable
    USING
        DocIdColumn ('docid_column')
        TokenColumn ('token_column')
        [ TFNormalization ({'BOOL' | 'COUNT' | 'NORMAL' | 'LOG' | 'AUGMENT'}) ]
        [ IDFNormalization ({'UNARY' | 'LOG' | 'LOGNORM' | 'SMOOTH'}) ]
        [ Regularization ({'L2' | 'L1' | 'NONE'}) ]
        [ Accumulate ({'accumulate_column'|'accumulate_column_range'}[,...]) ]
)
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Required Syntax Elements for TD\_TFIDF

### ON clause

Specifies the table name, view name or query as an InputTable.

### DocIdColumn

Specifies the column with the document identifier.

### TokenColumn

Specifies the column with the document tokens.

## Optional Syntax Elements for TD\_TFIDF

### TFNormalization

Specifies the normalization method for calculating the term frequency (TF).

Default: NORMAL

The argument must have one of the following values:

Values	Description
BOOL	Boolean frequency: $tf(t,d) = 1$ if t occurs in d; otherwise $tf(t,d) = 0$
COUNT	Raw frequency $tf(t,d) = f(t,d)$ where $f(t,d)$ is the number of times t occurs in d (that is, raw frequency, rf).
NORMAL	Normalized frequency $tf(t,d) = f(t,d) / \sum \{w : w \in d\}$ This value is rf divided by the number of terms in the document.
LOG	Logarithmically-scaled frequency: $tf(t,d) = 1 + \log(f(t,d))$ This value is the natural logarithm of rf.
AUGMENT	Augmented frequency, which prevents bias towards longer documents: $tf(t,d) = 0.5 + (0.5 \times f(t,d)) / \max \{f(w,d) : w \in d\}$ This value is rf divided by the maximum raw frequency of any term in the document.

### IDFNormalization

Specifies the normalization method for calculating the inverse document frequency (IDF).

Default: LOG

The argument must have one of the following values:

Values	Description
UNARY	$idf(t,D) = 1$ Used to disable IDF calculation.
LOG	$idf(t,D) = \log(N/N_t)$ where N is the total number of documents in the corpus, that is, $N =  D $ and $N_t$ is the number of documents d where the term t appears, that is, $N_t =  \{d \in D : t \in d\} $
LOGNORM	$idf(t,D) = 1 + \log(N / N_t)$
SMOOTH	$idf(t,D) = 1 + \log((1 + N) / (1 + N_t))$

### Regularization

Specifies the regularization method for calculating the TF-IDF score.

Default: NONE

The argument must have one of the following values:

Values	Description
L2	Euclidean regularization: $\text{tfidf}(t,d) = \text{tf}(t,d) * \text{idf}(t,D) / \sqrt{\sum \{(\text{tf}(w,d) * \text{idf}(w,D))^2 : w \in d\}}$ The product of tf and idf values for a term t in document d is divided by the square root of the sum of the squared products of tf and idf values for each term in the document.
L1	Manhattan regularization: $\text{tfidf}(t,d) = \text{tf}(t,d) * \text{idf}(t,D) / \sum \{ \text{tf}(w,d) * \text{idf}(w,D)  : w \in d\}$ The product of tf and idf values for a term t in document d is divided by the sum of the absolute products of tf and idf values for each term in the document.
NONE	No regularization: $\text{tfidf}(t,d) = \text{tf}(t,d) * \text{idf}(t,D)$

**Accumulate**

Specifies the input columns to copy to the output table.

**TD\_TFIDF Input****Input Table Schema**

Column	Data Type	Description
doc_id_column	BYTEINT, SMALLINT, INTEGER, BIGINT	Column with the document identifier.
token_column	CHAR, VARCHAR	Column with the document tokens.
accumulate_column	Any	Column which needs to be copied to the output table.

**TD\_TFIDF Output****Output Table Schema**

Column	Data Type	Description
doc_id_column	BYTEINT, SMALLINT, INTEGER, BIGINT	Document identifier of document d.
token_column	CHAR, VARCHAR	Term t.
TD_TF	FLOAT	Term frequency of term t in document d, calculated as specified by TFNormalization formula.
TD_IDF	FLOAT	Inverse document frequency of term t in document corpus D, calculated as specified by IDFNormalization formula.

Column	Data Type	Description
TD_TF_IDF	FLOAT	TFIDF score of term t in document d in corpus D, calculated as specified by the Regularization formula.
accumulate_column	Any	Accumulate column copied from input to output.

## Example: How to Use TD\_TFIDF

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### TD\_TFIDF Input

Input table before tokenization creation:

```
CREATE TABLE tfidf_input (docid integer, content varchar(100),
category varchar(10));
INSERT INTO tfidf_input (1,'The quick brown fox jumps over the
lazy fox.', 'Animals');
INSERT INTO tfidf_input (2,'Scientists conducted experiments in the lab to
analyze the chemical reactions.', 'Science');
INSERT INTO tfidf_input (3,'Using advanced equipments in the lab, scientists
observed unexpected reactions in the lab.', 'Science');
```

Tokenization call:

```
CREATE MULTISET TABLE tfidf_input_tokenized AS (
SELECT docid, cast(token as varchar(15)) as token, category FROM TD_TextParser (
ON tfidf_input AS InputTable
USING
TextColumn ('content')
ConvertToLowercase ('true')
OutputByWord ('true')
Punctuation ('\[.,?!\]\'')
RemoveStopwords ('true')
StemTokens ('true')
Accumulate ('docid','category')
) AS dt ) WITH DATA;
```

Input table after tokenization:

docid	token	category
1	brown	Animals
1	fox	Animals
1	fox	Animals
1	jump	Animals
1	lazi	Animals
1	over	Animals
1	quick	Animals
2	analyz	Science
2	chemic	Science
2	conduct	Science
2	experi	Science
2	lab	Science
2	reaction	Science
2	scientist	Science
3	advanc	Science
3	equip	Science
3	lab	Science
3	observ	Science
3	reaction	Science
3	scientist	Science
3	unexpect	Science
3	use	Science

## TD\_TFIDF SQL Call

```
SELECT * FROM TD_TFIDF (
    ON tfidf_input_tokenized AS InputTable
    USING
        DocIdColumn ('docid')
        TokenColumn ('token')
        TFNormalization ('LOG')
        IDFNormalization ('SMOOTH')
        Regularization ('L2')
        Accumulate ('category')
) AS dt ORDER BY docid, token;
```

## TD\_TFIDF Output

docid	token	TD_TF	TD_IDF	TD_TF_IDF	category
1	brown	1.00000000000000E+00	1.69314718055995E+00	3.56535187467553E-01	Animals
1	fox	1.69314718055995E+00	1.69314718055995E+00	6.03666547431099E-01	Animals
1	jump	1.00000000000000E+00	1.69314718055995E+00	3.56535187467553E-01	Animals

1	lazi	1.00000000000000E+00	1.69314718055995E+00	3.56535187467553E-01	Animals
1	over	1.00000000000000E+00	1.69314718055995E+00	3.56535187467553E-01	Animals
1	quick	1.00000000000000E+00	1.69314718055995E+00	3.56535187467553E-01	Animals
2	analyz	1.00000000000000E+00	1.69314718055995E+00	4.17566623878192E-01	Science
2	chemic	1.00000000000000E+00	1.69314718055995E+00	4.17566623878192E-01	Science
2	conduct	1.00000000000000E+00	1.69314718055995E+00	4.17566623878192E-01	Science
2	experi	1.00000000000000E+00	1.69314718055995E+00	4.17566623878192E-01	Science
2	lab	1.69314718055995E+00	1.28768207245178E+00	3.17570180428344E-01	Science
2	reaction	1.00000000000000E+00	1.28768207245178E+00	3.17570180428344E-01	Science
2	scientist	1.00000000000000E+00	1.28768207245178E+00	3.17570180428344E-01	Science
3	advanc	1.00000000000000E+00	1.69314718055995E+00	3.57715385483810E-01	Science
3	equip	1.00000000000000E+00	1.69314718055995E+00	3.57715385483810E-01	Science
3	lab	1.00000000000000E+00	1.28768207245178E+00	4.60623688927680E-01	Science
3	observ	1.00000000000000E+00	1.69314718055995E+00	3.57715385483810E-01	Science
3	reaction	1.00000000000000E+00	1.28768207245178E+00	2.72051770936621E-01	Science
3	scientist	1.00000000000000E+00	1.28768207245178E+00	2.72051770936621E-01	Science
3	unexpected	1.00000000000000E+00	1.69314718055995E+00	3.57715385483810E-01	Science
3	use	1.00000000000000E+00	1.69314718055995E+00	3.57715385483810E-01	Science

## TD\_WordEmbeddings

Word embedding is the representation of a word/token in multi-dimensional space such that words/tokens with similar meanings have similar embeddings. Each word/token is mapped to a vector of real numbers that represent the word/token. The Analytics Database function TD\_WordEmbeddings produces vectors for each piece of text and can find the similarity between the texts. The options are token-embedding, doc-embedding, token2token-similarity, and doc2doc-similarity.

The ModelTable contains pretrained words/tokens and their corresponding vector mappings in multidimensional space. You can use pre-defined vectors from [Word Vectors](#), or train your own using packages such as GloVe or Word2Vec. Note that the ModelTable format expects the vectors in GloVe format (one word/token vector pair per row). To convert a Word2Vec file, simply delete the first row which contains the number of words or tokens and the number of dimensions.

### Note:

- This function supports CHARACTER SET LATIN.
- This function does not support CHARACTER SET UNICODE.

### Function Information

- [TD\\_WordEmbeddings Syntax](#)
- [Required Syntax Elements for TD\\_WordEmbeddings](#)
- [Optional Syntax Elements for TD\\_WordEmbeddings](#)
- [TD\\_WordEmbeddings Input](#)
- [TD\\_WordEmbeddings Output](#)
- [Example: How to Use TD\\_WordEmbeddings](#)

## TD\_WordEmbeddings Syntax

```
TD_WordEmbeddings (
  ON { table | view | (query) } AS InputTable PARTITION BY ANY
  ON { table | view | (query) } AS ModelTable DIMENSION
```

USING

```

IDColumn('IDColumn')
ModelVectorColumns({ 'vector_columns' | 'vector_columns_range' }[,...])
ModelTextColumn('text_columns')
PrimaryColumn ('text_column')
[ SecondaryColumn ('text_column')]
[ Accumulate ('text_column') ]
[ Operation({ 'token-embedding' | 'doc-embedding' | 'token2token-
similarity' | 'doc2doc-similarity'}) ]
[ RemoveStopWords({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'})]
[ ConvertToLowercase({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'})]
[ StemTokens ({'true' | 't' | 'yes' | 'y' | '1' | 'false' | 'f' | 'no' | 'n' | '0'})]
)

```

#### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_WordEmbeddings

### IDColumn

Identifier that uniquely identifies the row of the input table.

### ModelVectorColumns

Range of columns in the model table that contains real value vector.

### ModelTextColumn

Column that contains the token in the model.

### PrimaryColumn

Name of the input table column that contains the text.

## Optional Syntax Elements for TD\_WordEmbeddings

### SecondaryColumn

Name of the input table column that contains the text. This field is applicable for the token2token-similarity and doc2doc-similarity operations only.

**Accumulate**

List of columns to be added to the output from the input table. This is not applicable with the token-embedding operation.

**Operation**

Operation to be performed on the data. Options are:

- token-embedding: Emits vectors to all tokens in the column. Each token present in the specified text column is mapped to a vector of real numbers that represents the semantic meaning of that token. For example, the word "dog" might be represented by the vector [0.1, 0.2, 0.3, 0.4, 0.5], where each number represents a different aspect of the meaning of the word.
- doc-embedding: Vectorizes each token in the document and combines them. For example, the document "The dog ran across the street" might be represented by the vector [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9], where each number represents a different aspect of the meaning of the document.
- token2token-similarity: Computes the similarity between tokens and quantifies the result value. It measures how similar or related two tokens are based on their word embeddings. If the word embeddings of two tokens are close in the multi-dimensional space, the similarity value will be higher, indicating a semantic similarity between the tokens. For example, the similarity between the words "dog" and "cat" would be higher than the similarity between the words "dog" and "table".
- doc2doc-similarity: Computes the similarity between documents and quantifies the result value. It considers the embeddings of two entire documents, which are created using the "doc-embedding" operation. The similarity value reflects how similar or related two documents are in terms of their content. For example, the doc2doc-similarity between the documents "The dog ran across the street" and "The cat sat on the mat" would be higher than the doc2doc-similarity between the documents "The dog ran across the street" and "The apple fell from the tree".

Default value: token-embedding

**RemoveStopWords**

Stop words in English include words such as "the", "and", "in", "of", "to", "is", "it", "on", "at", and so on. All stop words present in the input table text are removed before any operation is performed. Applicable to all operations except token2token-similarity. Default is False.

**ConvertToLowercase**

All operations are performed after converting input table text to lowercase letters. Default is True.

**StemTokens**

Converts word to its root word in the input table, such as converting going to go. Default is False.

**TD\_WordEmbeddings Input****Input Table Schema**

Column	Data Type	Description
IDColumn	INTEGER, CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE for CHAR and VARCHAR)	Identifier that uniquely identifies the row.
PrimaryColumn	CHAR, VARCHAR (CHARACTER SET LATIN)	Name of the input column that contains the text.
SecondaryColumn	CHAR, VARCHAR (CHARACTER SET LATIN)	Name of the input column that contains the text. This field is applicable for the token2token-similarity and doc2doc-similarity operations only.

**TD\_WordEmbeddings Output****Output Table Schema**

Column	Data Type	Description
IDColumn	INTEGER, CHAR, VARCHAR (CHARACTER SET LATIN or UNICODE for CHAR and VARCHAR)	Unique row identifier.
tokenColumn	VARCHAR (CHARACTER SET LATIN)	Tokens from the target column data. Applicable for token-embedding only.
v1	Real	Coordinate point of the first dimension. Applicable for token-embedding and doc-embedding only.
v2	Real	Coordinate point of the second dimension. Applicable for token-embedding and doc-embedding only. <b>Note:</b> There are as many v columns as there are dimensions.
similarity	Real	Similarity value between two text values. Applicable for token2token-similarity and doc2doc-similarity.

Column	Data Type	Description
Accumulate columns	Same as input data type	Columns added from the input. Applicable for doc-embedding, token2token-similarity and doc2doc-similarity.

## Example: How to Use TD\_WordEmbeddings

The InputTable, wordEmb\_inputTable, used for the token-embedding operation and doc-embedding operation examples, is as follows:

doc_id	doc1	doc2
1	I like pizza	I love pizza
2	single_token	token
3	food is delicious	dinner is yummy
4	tokyo hosting olympics	food is delicious
5	person xyz was assisted by nurses	few medics helped person xyz

The model table, wordEmbedModel, used for the token-embedding operation and doc-embedding operation examples, is as follows:

doc_id	v1	v2	v3	v4
assisted	0.10058	0.1914	0.28125	0.17382
by	-0.11572	-0.03149	0.15917	0.13867
delicious	-0.18164	-0.13281	0.03906	0.31445
dinner	-0.06152	-0.08496	-0.15039	0.42382
few	0.13867	0.02941	-0.18652	0.15039
food	-0.18164	-0.16503	-0.16601	0.35742
hosting	-0.06396	0.25585	0.04321	0.01721
i	-0.22558	-0.01953	0.09082	0.2373
is	0.00704	-0.07324	0.17187	0.02258
helped	0.12695	0.09033	0.26367	0.08544
like	0.10351	0.13769	-0.00297	0.18164
love	0.10302	-0.15234	0.02587	0.16503
nurses	-0.04638	-0.14257	-0.34179	0.21582

doc_id	v1	v2	v3	v4
olympics	-0.39648	0.02038	0.07275	0.24414
person	0.27539	0.24707	0.01721	0.16796
pizza	-0.12597	0.02539	0.16699	0.55078
medics	0.05981	0.26171	0.16894	0.60156
token	0.04174	0.2041	-0.26757	0.29882
tokyo	-0.05664	-0.05029	-0.0075	0.23828
was	0.026	-0.00189	0.18554	-0.05175
xyz	-0.01574	-0.13476	0.1582	0.11328
yummy	-0.18945	0.06591	-0.00417	0.43359

### Example: TD\_WordEmbeddings SQL Call Using token-embedding Operation

```
SELECT * FROM TD_wordembeddings (
ON wordEmb_inputTable AS InputTable
ON wordEmbedModel AS ModelTable DIMENSION
USING
IDColumn('doc_id')
ModelVectorColumns('[1:4]')
PrimaryColumn('doc1')
Operation('token-embedding')
MODELTEXTCOLUMN('token')
)AS dt ORDER BY doc_id ASC;
```

### TD\_WordEmbeddings Output Table Using token-embedding Operation

id	token	v1	v2	v3	v4
--	--	--	--	--	--
1	i	-0.22558	-0.01953	0.09082	0.2373
1	like	0.10351	0.13769	-0.00297	0.18164
1	pizza	-0.12597	0.02539	0.16699	0.55078
2	single_token	0	0	0	0
3	delicious	-0.18164	-0.13281	0.03906	0.31445
3	is	0.00704	-0.07324	0.17187	0.02258
3	food	-0.18164	0.16503	-0.16601	0.35742
4	olympics	-0.39648	0.02038	0.07275	0.24414
4	hosting	-0.06396	0.25585	0.04321	0.01721
4	tokyo	-0.05664	-0.05029	-0.0075	0.23828
5	nurses	-0.04638	-0.14257	-0.34179	0.21582

```

5 person      0.27539  0.24707  0.01721  0.16796
5 assisted    0.10058  0.1914   0.28125  0.17382
5 was         0.026    -0.00189  0.18554  -0.05175
5 by          -0.11572 -0.03149  0.15917  0.13867
5 xyz         -0.01574 -0.13476  0.1582   0.11328

```

### **Example: TD\_WordEmbeddings SQL Call Using doc-embedding Operation**

```

SELECT * FROM TD_wordembeddings (
ON wordEmb_inputTable AS InputTable
ON wordEmbedModel AS ModelTable DIMENSION
USING
IDColumn('doc_id')
ModelVectorColumns('[1:4]')
PrimaryColumn('doc1')
Operation('doc-embedding')
MODELTEXTCOLUMN('token')
Accumulate('doc1')
)AS dt ORDER BY doc_id ASC;

```

### **TD\_WordEmbeddings Output Table Using doc-embedding Operation**

doc_id	v1	v2	v3	v4	doc
1	-0.08268	0.04785	0.08494	0.32324	i like pizza
2	0	0	0	0	single_token
3	-0.11874	-0.01367	0.01497	0.23148	food is delicious
4	-0.17236	0.07531	0.03615	0.16654	tokyo hosting olympics
5	0.03735	0.02129	0.07659	0.1263	person xyz was assisted by nurses

The InputTable, wordEmb\_inputTable2, used for the token2token-similarity operation and doc2doc-similarity operation examples, is as follows:

doc_id	Token1	Token2
1	food	delicious
2	pizza	food
3	love	like
4	nurses	olympics

### **Example: TD\_WordEmbeddings SQL Call Using token2token-similarity Operation**

```
SELECT * FROM TD_wordembeddings (
ON wordEmb_inputTable2 AS InputTable
ON wordEmbedModel AS ModelTable DIMENSION
USING
IDColumn('token_id')
ModelVectorColumns('[1:4]')
PrimaryColumn('token1')
SECONDARYCOLUMN('token2')
Operation('token2token-similarity')
MODELTEXTCOLUMN('token')
Accumulate('token1','token2')
)AS dt ORDER BY token_id ASC;
```

### **TD\_WordEmbeddings Output Table Using token2token-similarity Operation**

doc_id	Similarity	Token1	Token2
1	0.64836	food	delicious
2	0.71667	pizza	food
3	0.31491	love	like
4	0.21295	nurses	olympics

### **Example: TD\_WordEmbeddings SQL Call Using doc2doc-similarity Operation**

```
SELECT * FROM TD_wordembeddings (
ON wordEmb_inputTable AS InputTable
ON wordEmbedModel AS ModelTable DIMENSION
USING
IDColumn('doc_id')
ModelVectorColumns('[1:4]')
PrimaryColumn('doc1')
SECONDARYCOLUMN('doc2')
Operation('doc2doc-similarity')
MODELTEXTCOLUMN('token')
Accumulate('doc1','doc2')
)AS dt ORDER BY token_id ASC;
```

### **TD\_WordEmbeddings Output Table Using doc2doc-similarity Operation**

doc_id	Similarity	doc1	doc2
--------	------------	------	------

1	0.96055	i like pizza	i love pizza
2	0	single_token	token
3	0.97761	food is delicious	dinner is yummy
4	0.88368	tokyo hosting olympics	food is delicious
5	0.94299	person xyz was assisted by nurses few medics helped person xyz	

# Hypothesis Testing Functions

Hypothesis testing functions find the relative likelihood of hypotheses. You can accept the most likely hypotheses and reject the least likely.

- [Hypothesis Test Components](#)
- [Hypothesis Test Types](#)
- [TD\\_ANOVA](#)
- [TD\\_Chisq](#)
- [TD\\_FTest](#)
- [TD\\_ZTest](#)

## Hypothesis Test Components

All hypothesis tests have the following components:

Component	Description
Null hypothesis ( $H_0$ )	The null hypothesis is known as a hypothesis of no difference. Example: Experimental drug is no better than placebo. The null hypothesis is accepted or rejected based on a statistical test of the hypothesis.
Alternate hypothesis ( $H_1$ )	Hypothesis accepted if null hypothesis is rejected. Example: Experimental drug is more effective than placebo.
Alpha ( $\alpha$ ) (Also called <i>significance level</i> or <i>Type I error</i> .)	The Null Hypothesis is rejected if the P-value is smaller than the specified Alpha value (where Alpha is the probability of rejecting the null hypothesis when it is true). Most common $\alpha$ values are 0.01, 0.05, and 0.10, corresponding to 99%, 95%, and 90% confidence, respectively. Results are "statistically significant at $\alpha$ ."
Test statistic	Value to which data set is reduced, used in hypothesis test. Its sampling distribution under null hypothesis must be calculable (exactly or approximately), making p_values calculable.
Degrees of freedom	Number of independent pieces of information needed to estimate a population parameter (for example, $\mu$ or $\sigma^2$ ) for sample of specified size.
Critical value	Quantile of distribution of test statistic under null hypothesis. Used to determine rejection region.
p_value	Probability of test results at least as extreme as test statistic results observed under assumption that null hypothesis is true. The smaller the p_value, the stronger the evidence against the null hypothesis.

Component	Description
Hypothesis test conclusion	Acceptance or rejection of null hypothesis.

## Hypothesis Test Types

A hypothesis test is either:

- *One-tailed or two-tailed*  
A one-tailed test can be either *lower-tailed* or *upper-tailed*.
- *One-sample or two-sample*
- *Paired or unpaired*

### Hypothesis Test Term Definitions

Term	Description
One-sample test	Uses one test sample.
One-tailed test	Rejection region is the lower tail or the upper tail of the sampling distribution under the null hypothesis $H_0$ .
Lower-tailed test	Alternate hypothesis ( $H_1$ ): $\mu < \mu_0$
Upper-tailed test	Alternate hypothesis ( $H_1$ ): $\mu > \mu_0$
Two-tailed test	The null hypothesis assumes that $\mu = \mu_0$ where $\mu_0$ is a specified value. Two-tailed test considers both lower and upper tails of distribution of test statistic. Alternate hypothesis ( $H_1$ ): $\mu \neq \mu_0$
Two-sample test	Uses two test samples.
Paired test	Compares study subjects at two different times. The null and alternative hypotheses are the same as one sample test. The paired test becomes a one-sample test because the test considers the differences between sample values before and after the subjects are exposed to treatment.
Unpaired test	Compares different subjects drawn from two independent populations. $H_0$ : $\mu_1 = \mu_2$ The alternate hypotheses are as follows: <ul style="list-style-type: none"> <li>• Alternate hypothesis for upper-tailed test (<math>H_1</math>): <math>\mu_1 &gt; \mu_2</math></li> <li>• Alternate hypothesis for lower-tailed test (<math>H_1</math>): <math>\mu_1 &lt; \mu_2</math></li> <li>• Two-tailed test <math>\mu_1 \neq \mu_2</math></li> </ul>

## TD\_ANOVA

Analysis of variance (ANOVA) is a statistical test that analyzes the difference among the means of multiple groups. Analysis works by comparing the variation between groups with the variation within the groups. If the variation between the groups is much larger than the variation within the groups, then it suggests that there is a significant difference between the means of the groups.

The difference in means is due to at least one group's distribution being different from the others. If all groups means were the same with approximately same variance, then they would be governed by the same distribution. With all groups belonging to the same distribution, the ratio (F-stat) of between-group variance and within-group variance is approximately 0. But if at least one of the groups is from a different distribution, then the ratio is larger than 0.

The null hypothesis of ANOVA is that there is no difference among group means. However, if any one of the group means is significantly different from the overall mean, then the null hypothesis is rejected.

A one-way analysis involves one factor or independent variable. The factor may have two or more categorical groups and analysis is used to establish if there is any significant difference between those categorical groups.

The TD\_ANOVA function is used in a variety of contexts, such as comparing the performance of different treatments in a clinical trial, comparing the effectiveness of different marketing strategies, or comparing the mean salary of employees in different departments of a company.

To conduct an analysis, the data must meet certain assumptions, including that the data are normally distributed and that the variances of the groups are approximately equal.

### Function Information

- [TD\\_ANOVA Syntax](#)
- [Required Syntax Elements for TD\\_ANOVA](#)
- [Optional Syntax Elements for TD\\_ANOVA](#)
- [TD\\_ANOVA Input](#)
- [TD\\_ANOVA Output](#)
- [Examples: How to Use TD\\_ANOVA](#)

## TD\_ANOVA Syntax

```
TD_ANOVA (
  ON { table | view | (query) } AS InputTable
  USING
  {
    [ GroupColumns({'group_col1' | 'group_col2'[,...] | group_col_range[,...]}) ]
    |
    GroupNameColumn('group_name_column')
    GroupValueColumn('group_value_column')
  }
```

```

    GroupNames('group_name1' | 'group_name2'[,...])
  |
  NumGroups(number_of_groups)
}
}
[ Alpha(alpha) ]
)

```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_ANOVA

### **ON clause**

Specifies the table name, view name or query as an InputTable.

### **GroupNameColumn**

Required for group-value input. Specifies the column that contains the groups names.

### **GroupValueColumn**

Required for group-value input. Specifies the column that contains the values for each group.

### **GroupNames**

Required for group-value input. Specifies the groups names to be used for ANOVA.

**Note:**

Cannot be specified if NumGroups is used.

Unicode and double quotes cannot be used.

### **NumGroups**

Required for group-value input. Specifies the groups number to be used for ANOVA.

**Note:**

Cannot be specified if GroupNames is used.

## Optional Syntax Elements for TD\_ANOVA

### GroupColumns

Specifies the input table column names or a column range.

**Note:**

Unicode and double quotes are supported.

### Alpha

Specifies the probability of rejecting the null hypothesis when the null hypothesis is true.

Default: 0.05

Valid range: [0,1]

## TD\_ANOVA Input

TD\_ANOVA accepts one input table which can be formatted either as multiple columns or group-value.

### Option 1 - Multiple Column Input Using GroupColumns Argument

Input	Description	Comments
InputTable	A table that contains columns of samples. Each column represents a different sample with rows containing the value of each instance of that sample.	In a dense format.

### InputTable Schema

Column	Data Type	Description
group_column<i>	DOUBLE	Numerical value used to calculate ANOVA.

### Option 2 - Group-Value Input Using GroupNames or NumGroups Arguments

Input	Description	Comments
InputTable	A table that contains instances per row, specifying its group names in the GroupNameColumn and its corresponding value in GroupValueColumn.	In a dense format.

### InputTable Schema

Column	Data Type	Description
group_name_column	VARCHAR	Instance group name.

Column	Data Type	Description
group_value_column	DOUBLE	Instance numerical value used to calculate ANOVA.

## TD\_ANOVA Output

### Output Table Schema

Column	Data Type	Description
sum_of_squares (between groups)	DOUBLE	The sum of squares (variation) between groups.
sum_of_squares (within groups)	DOUBLE	The sum of squares (variation) within groups.
Df (between groups)	INTEGER	The degrees of freedom corresponding to the between groups sum of squares.
Df (within groups)	INTEGER	The degrees of freedom corresponding to the within group sum of squares.
mean_square (between groups)	DOUBLE	The mean of sum of squares, which is calculated by dividing the sum of squares (between groups) by the degrees of freedom (between groups).
mean_square (within groups)	DOUBLE	The mean of sum of squares, which is calculated by dividing the sum of squares (within groups) by the degrees of freedom (within groups).
F_statistic	DOUBLE	The F-statistic is calculated as the ratio of between group sum of squares/(k-1) and within group sum of squares/(N-k+1). The null hypothesis is rejected when F-statistic is greater than the critical value of F.
alpha	DOUBLE	The level of significance of the test.
critical_f	DOUBLE	The critical value of F denoted by (k-1, N-k) where: <ul style="list-style-type: none"> <li>• k is the number of groups</li> <li>• N is the total number of observations</li> <li>• k-1 is the degrees of freedom between groups</li> <li>• N-k+1 is the degrees of freedom within groups</li> </ul>
p_value	DOUBLE	The probability value associated with the F-statistic value. The null hypothesis is rejected when p_value is less than alpha.
conclusion	VARCHAR	The result of the test based on if the null hypothesis is rejected or not.

## Examples: How to Use TD\_ANOVA

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Using TD\\_ANOVA to Determine Difference of Insect Sprays](#)
- [Using GroupNames Argument](#)
- [Using NumGroups Argument](#)

### Using TD\_ANOVA to Determine Difference of Insect Sprays

Your independent variable is insect spray type, and you have data on spray type A, B, C, D, E, and F. You can use one-way analysis to determine whether there is any difference among these spray types based on their means.

#### TD\_ANOVA Input: Insect\_sprays

groupA	groupB	groupC	groupD	groupE	groupF
7	17	1	5	6	9
10	17	2	5	1	13
10	11	0	3	3	11
12	14	1	3	6	16
13	13	4	4	4	13
14	11	2	6	5	22
14	16	3	4	3	15
14	7	1	2	6	24
17	19	3	5	3	26
20	21	0	5	2	26
20	21	7	12	3	15
23	17	1	5	1	10

#### TD\_ANOVA SQL Call

```
SELECT * from TD_ANOVA (
ON insect_sprays as InputTable
USING
ALPHA (0.05)
) AS dt;
```

## TD\_ANOVA Output Table

```

sum_of_squares(between groups) sum_of_squares(within groups) df(between groups)
df(within groups) mean_square(between groups) mean_square(within groups)
f_statistic alpha critical_f p_value conclusion
-----
-----
-----
2656.902778000 1019.083333000 5 66 531.380556000 15.440657000
34.414376000 0.050000000 2.353809000 0.000000000 Reject Null hypothesis

```

## Using GroupNames Argument

### TD\_ANOVA Input: insect2cols

```

groupName groupValue
-----
groupE    34
groupF    15
groupE    31
groupF    26
groupE    21
groupF    26
groupE    11
groupF    10
...
groupD    12
groupB    21
groupD    12
groupB    19
groupD    5
groupB    21
...
groupC    13
groupC    11
groupC    10
groupC    21
groupC    11
groupA    20
groupA    17
groupA    17
groupA    20
groupA    23
groupA    14
...

```

### TD\_ANOVA SQL Call

```

SELECT * FROM td_anova(
  ON insect2cols AS InputTable
  USING
    GroupValueColumn('groupValue')
    GroupNameColumn('groupName')
    GroupNames('groupA', 'groupB', 'groupC')
) AS dt;

```

## TD\_ANOVA Output

Sum of Squares (between groups)	Sum of Squares (within groups)	DF (between groups)	DF (within groups)	Mean Square (between groups)	Mean Square (within groups)	F_statistic	Alpha	Critical_f	P_value
68.6	534.6	2	27	34.3	19.8	1.732323	0.05	3.35413082	0.19595795

## Using NumGroups Argument

### TD\_ANOVA Input: insect2cols

```
groupName groupValue
-----
groupE    34
groupF    15
groupE    31
groupF    26
groupE    21
groupF    26
groupE    11
groupF    10
...
groupD    12
groupB    21
groupD    12
groupB    19
groupD    5
groupB    21
...
groupC    13
groupC    11
groupC    10
groupC    21
groupC    11
groupA    20
groupA    17
groupA    20
groupA    23
groupA    14
...
```

### TD\_ANOVA SQL Call

```
SELECT * FROM TD_ANOVA(
  ON insect2cols AS InputTable
  USING
    GroupValueColumn('groupValue')
    GroupNameColumn('groupName')
    NumGroups(6)
) AS dt;
```

## TD\_ANOVA Output

Sum of Squares (between groups)	Sum of Squares (within groups)	DF (between groups)	DF (within groups)	Mean Square (between groups)	Mean Square (within groups)	F_statistic	Alpha	Critical_f	P_value
284.28333	5,149.9	5	54	56.856666	95.368515	0.5961785	0.05	2.38606986	0.702928314

## TD\_Chisq

Analytics Database provides built-in support for performing chi-square tests, which you can use to analyze relationships between categorical variables.

TD\_Chisq performs Pearson's chi-squared ( $\chi^2$ ) test for independence, which determines if there is a statistically significant difference between the expected and observed frequencies in one or more categories of a contingency table (also called a *cross tabulation*). This function takes two or more columns as input and returns a result set that contains the chi-square statistic, the degrees of freedom, and the p-value for the test.

The supported test types follow:

- One-tailed, upper-tailed
- One-sample
- Unpaired

### Why Use Chi-Square Test?

The chi-square test is a statistical method used in analytics to determine whether there is a significant relationship between two categorical variables.

In analytics, you can work with data that is organized into categories or groups, such as the gender of individuals, educational qualifications, or income levels. The chi-square test allows you to determine whether there is a significant relationship between two such categorical variables.

- For example, imagine you want to determine whether there is a relationship between the level of education and the employment status of a group of individuals. You can use the chi-square test to analyze the data and determine whether there is a significant association between the two variables.
- The chi-square test compares the observed frequencies of each category with the expected frequencies. If there is a significant difference between the observed and expected frequencies, you can conclude that there is a relationship between the two variables.

Overall, the chi-square test is a tool for analyzing categorical data and identifying relationships between variables. Used in fields like marketing, social sciences, healthcare, and finance, and so on.

## Function Information

- [TD\\_ChiSq Syntax](#)
- [Required Syntax Elements for TD\\_ChiSq](#)
- [Optional Syntax Elements for TD\\_ChiSq](#)
- [TD\\_ChiSq Input](#)
- [TD\\_ChiSq Output](#)
- [TD\\_ChiSq Usage Notes](#)
- [Example: How to Use TD\\_ChiSq](#)

## TD\_ChiSq Syntax

```
TD_ChiSq (
    ON { table | view | (query) } AS CONTINGENCY
    [ OUT [ PERMANENT | VOLATILE ] TABLE EXPCOUNTS (expected_values_table) ]
    USING
    [ Alpha (alpha) ]
)
```

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_ChiSq

### ON clause

Name of a table, view, or query that contains the actual frequencies for the categories.

## Optional Syntax Elements for TD\_ChiSq

### *alpha*

The significance level at which the test will be undertaken. *alpha* must be a numeric value in the range [0, 1].

Default: 0.05

**expected\_values\_table**

Name for the table of expected values. *expected\_values\_table* cannot be the name of an existing table.

Default behavior: Function does not output this table.

**TD\_ChiSq Input**

A contingency table, also known as a two-way frequency table, is a tabular mechanism with at least two rows and two columns used in statistics to present categorical data in terms of frequency counts.

A contingency table shows the observed frequency of two variables arranged into rows and columns. A cell is the intersection of a row and a column of a contingency table.

For example, a cell count  $n_{ij}$  represents a joint occurrence of row  $i$  and column  $j$  where  $i$  is a value between 1 to  $r$  (total number of rows) and  $j$  is a value between 2 to  $c$  (total number of columns).

You can interpret the contingency table in the example as follows:

- The first column represents the first category, gender, which has two labels, female and male represented by two rows.
- The second and third columns represent the second category, habits, which has two labels, smokers and non-smokers.

The second category can have at most 2046 unique labels. The function ignores NULL values in the table.

Maximum label length is 64000 for *category\_1*, 128 for all other columns.

For a valid test output, the value of each observed frequency in the CONTINGENCY table must be at least 5.

**CONTINGENCY Table Schema**

Column	Data Type	Description
<i>Name of categorical column 1</i>	Any	Columns can have one or multiple labels. Can either be an INTEGER, LATIN, or UTF8 code.
<i>category_2_label_1</i>	INTEGER, SMALLINT, BYTEINT, or BIGINT	Joint frequency of category 1 label $i$ and category 2 label 1, where $i$ has a value between 1 to $r$ .
<i>category_2_label_2</i>	INTEGER, SMALLINT, BYTEINT, or BIGINT	Joint frequency of category 1 label $i$ and category 2 label 2, where $i$ has a value between 1 to $r$ .
.		
.		
.		
.		

Column	Data Type	Description
<i>category_2_label_c</i>	INTEGER, SMALLINT, BYTEINT, or BIGINT	[Column appears zero or more times.] Joint frequency of category 1 label i and category 2 label c, where i has a value between 1 to r.

## TD\_Chisq Output

### Output Table Schema

Column	Data Type	Description
chi_square	DOUBLE PRECISION	Chi-squared statistic.
cramers_v	DOUBLE PRECISION	Cramer's V statistic.
df	INTEGER	Degrees of freedom.
alpha	DOUBLE PRECISION	alpha (see <a href="#">Optional Syntax Elements for TD_Chisq</a> ).
p_value	DOUBLE PRECISION	Probability associated with chi-squared statistic.
criticalvalue	DOUBLE PRECISION	Critical value calculated using Alpha for test.
conclusion	VARCHAR	Chi-squared test result, either 'reject null hypothesis' or 'fail to reject null hypothesis'.

### Table of Expected Values

The function outputs this table only if you include the OUT clause in the function call. The OUT clause specifies its name.

This table contains the expected frequencies calculated under the assumption that the null hypothesis is true.

This table has the same schema as the CONTINGENCY table that contains the observed frequencies, except that all columns but the first have the data type DOUBLE PRECISION.

## TD\_Chisq Usage Notes

### Computational Method

The Chi-Square test finds statistically significant associations between categorical variables. The test determines if the categorical variables are statistically independent or not.

The contingency tables organize the data for analysis. A two-way contingency table consists of r rows and c columns wherein:

- Variable 1 corresponds rows that consists of r categories.
- Variable 2 corresponds columns that consists of c categories.

Each cell of the contingency table is the count of the joint occurrence of particular levels of variable 1 and variable 2.

For example, the following two-way contingency table shows the categorical variable Gender with two levels (Male, Female) and the categorical variable Affiliation with two levels (Smokers, Non-smokers).

**Gender Affiliation Table**

Gender	Affiliation	
	Smokers	Non-Smokers
Male	$n_{11}$	$n_{12}$
Female	$n_{21}$	$n_{22}$

The cell counts  $n_{ij}$ ,  $i = 1, 2; j = 1, 2$  are number of joint occurrences of Gender and Affiliation at their  $i^{\text{th}}$  and the  $j^{\text{th}}$  levels, respectively. The Null and alternative hypotheses  $H_0$  and  $H_1$  corresponding to a  $\chi^2$  test of independence is as follows:

$H_0$ : The two categorical variables are independent

vs

$H_1$ : The two categorical variables are not independent

Use the previous table to calculate the expected cell counts:

$$e_{11} = n_{11} + n_{21}$$

$$e_{12} = n_{11} + n_{12}$$

$$e_{21} = n_{21} + n_{22}$$

$$e_{22} = n_{12} + n_{22}$$

The following formula calculates the  $\chi^2$  test statistic:

$$\chi_{\text{stat}}^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(n_{ij} - e_{ij})^2}{e_{ij}}$$

The  $\chi^2$  statistic follows a Chi-Square distribution with  $r - 1$  and  $c - 1$  degrees of freedom. In the Gender Affiliation table,  $r=2$  and  $c=2$ . The Null hypothesis  $H_0$  is rejected if  $\chi_{\text{stat}}^2 > \chi^2_{r-1,c-1,\alpha}$  where  $\alpha \in \{0.10, 0.05, 0.01\}$ .

The following formula calculates the Cramer's V statistic:

$$V = \sqrt{\frac{\varphi^2}{\min(c - 1, r - 1)}} = \sqrt{\frac{\chi^2/n}{\min(c - 1, r - 1)}}$$

where:

- $\varphi$  is the phi coefficient
- $\chi^2$  derives from the Pearson's chi-squared test
- $n$  is the grand total of observations
- $c$  is the number of columns
- $r$  is the number of rows

Used rules to compute the hypothesis conclusion:

- If the chi-square statistic is greater than the critical value, then the function rejects the Null hypothesis.
- If the chi-square statistic is lesser than or equal to the critical value, then the function fails to reject the Null hypothesis.

## Example: How to Use TD\_ChiSq

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

This example tests whether gender influences party affiliation. The null hypothesis is that gender and party affiliation are independent. TD\_ChiSq compares the null hypothesis (expected frequencies) to the contingency table (observed frequencies).

### TD\_ChiSq Input: contingency1

The contingency table contains the frequencies of men and women affiliated with each party. *category\_1*, gender, has the following labels: female and male. *category\_2*, party, has the following labels: Democrats and Republicans.

This example illustrates a two-way contingency table with two categories, *category\_1* and *category\_2*, respectively.

Each row has a label, *i* which has a value between 1 to *r*, and each column has a label, *j* which has a value between 2 to *c*. The values of *c* and *r* are 3 and 2, respectively.

In this example, *category\_1\_label1* corresponds to females and *category\_1\_label2* corresponds to males. Similarly, *category2\_label1* corresponds to Democrats and *category2\_label2* corresponds to Republicans.

Query to create the contingency table is as follows:

### Query to construct table contingency1

```
CREATE MULTISET TABLE contingency1 AS
(
SELECT gender AS gender
, sum((case when party = 'dem' then 1 else 0 end)) as dem_party_cnt
, sum((case when party = 'rep' then 1 else 0 end)) as rep_party_cnt
FROM mytesttable
GROUP BY gender
) with data;
```

The input table, contingency1, can be like this:

gender	dem	rep
male	8	5
female	6	9

### Alternate Query to construct table contingency1 using PIVOT

```
DROP TABLE contingency1;
CREATE MULTISET TABLE contingency1 AS
(
SELECT *
FROM (select gender AS gender, party, count(party) AS party_count
FROM mytesttable group by gender, party) AS mytesttable
PIVOT ( SUM(party_count) AS party_cnt FOR party
IN ('dem' AS dem, 'rep' AS rep))Tmp
) with data;
```

The input table, contingency1 using PIVOT, can be like this:

gender	dem	rep
male	8	5
female	6	9

### Alternate Query to construct table contingency1 using TD\_Pivoting

```
DROP TABLE contingency1;
CREATE MULTISET TABLE contingency1 AS (
SELECT * FROM TD_Pivoting (
ON (select gender AS gender, party, count(party) AS party_count FROM
```

```

mytesttable GROUP BY gender, party) AS InputTable PARTITION BY gender
  USING
    PartitionColumns ('gender')
    TargetColumns ('party_count')
    PivotColumn('party')
    PivotKeys('dem','rep')
    Aggregation('party_count:SUM')
    OutputColumnNames('dem_party_cnt','rep_party_cnt')
) AS dt
) with data;

```

The input table, contingency1 using TD\_Pivoting, can be like this:

gender	dem	rep
male	8	5
female	6	9

### TD\_Chisq SQL Call

```

SELECT * FROM TD_Chisq (
  ON contingency1 AS CONTINGENCY
  OUT TABLE EXPCOUNTS (exptable1)
  USING
    Alpha (0.05)
) AS dt;

```

### TD\_Chisq Output Table

chi_square	cramers_v	df	alpha	p_value	criticalvalue	conclusion
1.292307692	0.214834462	1	0.050000000	0.255623108	3.841458821	Fail to reject Null hypothesis

### exptable1 SQL Call

```

SELECT * FROM exptable1;

```

### exptable1 Output Table

gender	dem	rep
male	6.500000000	6.500000000
female	7.500000000	7.500000000

## TD\_FTest

The F test is a parametric statistical test that measures the variability between two or more normally distributed populations. It is commonly used in data analysis and machine learning to determine the similarity or dissimilarity between data points or features in a dataset. The F statistic is calculated by comparing the ratio of two variances and assumes that the populations being compared are normally distributed.

The F test is an important concept in quality control and process improvement, where it is used to compare the variability of different production runs or batches. It is also useful in regression analysis to test the overall significance of a model or to compare the variances of the residuals in different models. The choice of distance metric for calculating the F statistic depends on the nature of the data and the problem being solved.

Overall, the F test is a tool for analyzing and comparing different populations in a dataset, and it is a fundamental concept in the field of statistics.

### Function Information

- [TD\\_FTest Syntax](#)
- [Required Syntax Elements for TD\\_FTest](#)
- [Optional Syntax Elements for TD\\_FTest](#)
- [TD\\_FTest Input](#)
- [TD\\_FTest Output](#)
- [TD\\_FTest Usage Notes](#)
- [Examples: How to Use TD\\_FTest](#)

## TD\_FTest Syntax

```
TD_FTest (
    [ ON { table | view | (query) } AS InputTable ]
USING
{
    [ FirstSampleColumn('column_name_1') ]
    [ SecondSampleColumn('column_name_2') ]
|
    SampleNameColumn('sample_name_column')
    SampleValueColumn('sample_value_column')
    [ FirstSampleName('sample_name_1') ]
    [ SecondSampleName('sample_name_2') ]
}
[ AlternativeHypothesis('lower-tailed'|'upper-tailed'|'two-tailed') ]
[ FirstSampleVariance(var1) ]
[ SecondSampleVariance(var2) ]
```

```
[ df1(value1) ]
[ df2(value2) ]
[ Alpha(value) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_FTest

**SampleNameColumn**

Required for sample value input. Specifies the column that contains the names of the groups.

**Note:**

Cannot be specified if FirstSampleColumn or SecondSampleColumn are specified.

**SampleValueColumn**

Required for sample value input. Specifies the column that contains the values for each group.

**Note:**

Cannot be specified if FirstSampleColumn or SecondSampleColumn are specified.

## Optional Syntax Elements for TD\_FTest

**ON clause**

Specifies the table name, view name, or query as a InputTable.

**FirstSampleColumn**

If this is specified, do not specify FirstSampleVariance. First Column to calculate variance.

**SecondSampleColumn**

If this is specified, do not specify SecondSampleVariance. Second Column to calculate variance.

**FirstSampleName**

If this is specified, do not specify FirstSampleVariance. First sample name, must exist in SampleNameColumn.

**SecondSampleName**

If this is specified, do not specify SecondSampleVariance. Second sample name, must exist in SampleNameColumn.

**AlternativeHypothesis**

Specifies the alternative hypothesis to use:

Option	Description
'lower-tailed'	Alternate hypothesis ( $H_1$ ): $\mu < \mu_0$
'upper-tailed'	Alternate hypothesis ( $H_1$ ): $\mu > \mu_0$
'two-tailed'	Rejection region is on two sides of sampling distribution of test statistic. Two-tailed test considers both lower and upper tails of distribution of test statistic. Alternate hypothesis ( $H_1$ ): $\mu \neq \mu_0$

Default: two-tailed

**FirstSampleVariance**

If this is specified, FirstSampleColumn is not needed. First distribution variance.

**SecondSampleVariance**

If this is specified, SecondSampleColumn is not needed. Second distribution variance.

**df1**

Need not be specified if FirstSampleColumn is specified. Numerator degrees of freedom.

**df2**

Need not be specified if SecondSampleColumn is specified. Denominator degrees of freedom.

**Alpha**

Alpha ( $\alpha$ ) value specified in input query.

Default: 0.05

## TD\_FTest Input

The input table is optional when FirstSampleVariance, SecondSampleVariance, df1 and df2 are specified.

### Option 1 - Multiple column input using FirstSampleColumn or SecondSampleColumn argument

Input	Description	Comments
InputTable	A table containing numeric datatypes in columns.	At least two columns need to be present in this table.

#### InputTable Schema

Column	Data Type	Description
column_name_<i>	DOUBLE	Numerical value used to calculate FTest.

### Option 2 - Multiple column input using FirstSampleColumn or SecondSampleColumn argument

Input	Description	Comments
InputTable	A table containing numeric datatypes in columns.	At least two columns need to be present in this table.

#### InputTable Schema

Column	Data Type	Description
group_name_column	VARCHAR	Instance group name.
group_value_column	DOUBLE	Instance numerical value used to calculate FTest.

## TD\_FTest Output

### Output Table Schema

The function produces an output table as follows.

Column	Data Type	Description
FirstSampleVariance	DOUBLE PRECISION	Variance of first sample population.
SecondSampleVariance	DOUBLE PRECISION	Variance of second sample population.
VarianceRatio	DOUBLE PRECISION	FirstSampleVariance/SecondSampleVariance
DF1	INTEGER	Degrees of freedom of first sample.

Column	Data Type	Description
DF2	INTEGER	Degrees of freedom of second sample.
CriticalValue	DOUBLE PRECISION	Critical value calculated using Alpha for test.
Alpha	DOUBLE PRECISION	<i>alpha</i> (see <a href="#">Optional Syntax Elements for TD_FTest</a> ).
p_value	DOUBLE PRECISION	Probability associated with <i>F</i> -test statistic.
Conclusion	VARCHAR	<i>F</i> -test result, either 'reject null hypothesis' or 'fail to reject null hypothesis'.

## TD\_FTest Usage Notes

In statistical analysis, the F-test is a commonly used hypothesis test to determine whether two population variances are equal. It is based on the F distribution, which is a probability distribution that arises in the analysis of variance (ANOVA) models.

The F-test is typically used in situations where we want to compare the variability of two sets of data or test whether a particular variable has a significant effect on the outcome of a study. It works by calculating the ratio of two variances and comparing it to a critical value obtained from the F distribution.

### Assumptions

- Populations from which samples are drawn are normally distributed.
- Populations are independent of each other.
- Data is numeric.

### Test Type

- One-tailed (lower and upper-tailed) or two-tailed (your choice)
- Two-sample
- Unpaired

### Computational Method

The F-test is used to test the Null hypothesis  $\sigma^2 = \sigma_0^2$  in various applications. For example, you might need to test the variability in the measurement of the thickness of a manufactured part in a factory. If the thickness is not equal to a certain thickness ( $\sigma_0^2$ ) then you can conclude that the manufacturing process is uncontrolled. The types of hypothesis are as follows:

$$H_0: \sigma^2 = \sigma_0^2$$

versus

$$H_1: \sigma^2 > \sigma_0^2 \text{ (upper-tailed)}$$

or

$$H_1: \sigma^2 < \sigma_0^2 \text{ (lower-tailed)}$$

or

$$H_1: \sigma^2 \neq \sigma_0^2 \text{ (two-tailed)}$$

Let  $x_1, x_2, \dots, x_n$  be a random sample. To test the hypotheses, the test statistic is calculated as:

$$\chi^2 = \frac{(n-1)s^2}{\sigma_0^2}$$

$$\text{where } s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

The statistic  $\chi^2$  follows an F distribution with  $n-1$  degrees of freedom.

For the one-sided upper-tailed test  $\sigma^2 > \sigma_0^2$ , the Null hypothesis  $H_0$  is rejected if  $\frac{(n-1)s^2}{\sigma_0^2} > \chi_{n-1,\alpha}^2$ .

For the one-sided lower-tailed test  $\sigma^2 < \sigma_0^2$ , the Null hypothesis  $H_0$  is rejected

$$\text{if } \frac{(n-1)s^2}{\sigma_0^2} < \chi_{n-1,1-\alpha}^2$$

For the two-sided alternative  $\sigma^2 \neq \sigma_0^2$ , the Null hypothesis  $H_0$  is rejected if

$$\frac{(n-1)s^2}{\sigma_0^2} \geq \chi_{n-1,\alpha/2}^2$$

or

$$\frac{(n-1)s^2}{\sigma_0^2} \leq \chi_{n-1,1-\alpha/2}^2$$

Also, the F-test is used to test if the variances of two populations are equal. The F-test can have the following tests:

- One-tailed test: The test is used to determine if the variance of one population is either greater than (upper-tailed) or less than (lower-tailed) the variance of another population.
- Two-tailed test: The test is used to determine significant differences in variances of the two populations and tests the Null hypothesis ( $H_0$ ) against the alternative hypothesis ( $H_1$ ) to find out if the variances are not equal.

Let  $x_1, x_2, \dots, x_{n1} \sim N(\mu_1, \sigma^2)$  and  $y_1, y_2, \dots, y_{n2} \sim N(\mu_2, \sigma^2)$  be random samples from two independent populations. The corresponding sample means and variances are as follows:

- Sample Means Formula:  $\bar{x} = \frac{1}{n_1} \sum_{i=1}^{n_1} x_i$
- Sample Variance Formula:  $\bar{y} = \frac{1}{n_2} \sum_{i=1}^{n_2} y_i$
- Sample Variance Formula for  $S_1^2$  and  $S_2^2$ :  $s_1^2 = \frac{1}{n_1-1} \sum_{i=1}^{n_1} (x_i - \bar{x})^2$   
and  $s_2^2 = \frac{1}{n_2-1} \sum_{i=1}^{n_2} (y_i - \bar{y})^2$

In the following calculation, assume that sample 1 has a larger variance than sample 2. If sample 2 has a larger variance than sample 1, switch the samples and apply the same formula.

$$H_0: \sigma_1^2 = \sigma_2^2$$

versus

$$H_1: \sigma_1^2 > \sigma_2^2$$

or

$$\sigma_1^2 < \sigma_2^2$$

The test statistic for the one-sided upper tailed test ( $\sigma_1^2 > \sigma_2^2$ ) is calculated as:

$$\frac{s_1^2}{s_2^2} \sim F_{n_1-1, n_2-1}$$

where:  $n_1-1$  and  $n_2-1$  are degrees of freedom corresponding to sample 1 and sample 2.

$$\frac{s_1^2}{s_2^2} \geq F_{n_1-1, n_2-1, \alpha}$$

The Null hypothesis  $H_0$  is rejected if

The test statistic for the one-sided lower-tailed test ( $\sigma_1^2 < \sigma_2^2$ ) is calculated as:

$$\frac{s_2^2}{s_1^2} \sim F_{n_2-1, n_1-1}$$

$$\frac{s_2^2}{s_1^2} \geq F_{n_2-1, n_1-1, \alpha}$$

The Null hypothesis  $H_0$  is rejected if

For the two-sided hypothesis test:

$$H_0: \sigma_1^2 = \sigma_2^2$$

versus

$$H_1: \sigma_1^2 \neq \sigma_2^2$$

The Null hypothesis  $H_0$  is rejected if:

$$\frac{s_1^2}{s_2^2} \geq F_{n_1-1, n_2-1, \alpha/2}$$

and

$$\frac{s_2^2}{s_1^2} \geq F_{n_2-1, n_1-1, \alpha/2}$$

The two-tailed test is based on the upper tail of the F-distribution.

## Examples: How to Use TD\_FTest

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Using FirstSampleVariance and SecondSampleVariance Arguments](#)
- [Using FirstSampleName and SecondSampleName Arguments](#)
- [Using FirstSampleName and SecondSampleVariance Arguments](#)
- [Using SecondSampleName and FirstSampleVariance Arguments](#)

## Using FirstSampleVariance and SecondSampleVariance Arguments

### TD\_FTest Input

With two sample variances instead of two sample columns, you do not need InputTable.

### TD\_FTest SQL Call

```
SELECT * FROM td_ftest (
  USING
```

```
FirstSampleVariance (1385.61)
SecondSampleVariance (521.22)
DF1 (9)
DF2 (19)
) AS dt;
```

## TD\_FTest Output

df2	firstsamplevariance	secondsamplevariance	varianceratio	p_value	Conclusion	df1
	CriticalValue	Alpha				
1.38561000000000E 003	5.21220000000000E 002	2.65839760561759E 000			Fail to reject Null hypothesis	9
19	2.88005204672380E 000	5.00000000000000E-002	6.96764431913391E-002			

## Using FirstSampleName and SecondSampleName Arguments

### TD\_FTest Input: insect2cols

```
groupName groupValue
-----
groupE    34
groupF    15
groupE    31
groupF    26
groupE    21
groupF    26
groupE    11
groupF    10
...
groupD    12
groupB    21
groupD    12
groupB    19
groupD    5
groupB    21
...
groupC    13
groupC    11
groupC    10
groupC    21
groupC    11
groupA    20
groupA    17
groupA    20
groupA    23
groupA    14
...
...
```

### TD\_FTest SQL Call

```
SELECT * FROM TD_FTest(
  ON insect2cols AS InputTable
  USING
    SampleValueColumn('groupValue')
    SampleNameColumn('groupName')
    FirstSampleName('groupE')
    SecondSampleName('groupC')
) AS dt;
```

## TD\_FTest Output

```
firstsamplevariance secondsamplevariance varianceratio df1 df2 CriticalValue Alpha p_value Conclusion
----- ----- ----- ----- ----- ----- ----- -----
305.2888888889 19.7888888889 15.4272880404 9 9 4.0259941583 0.05 0.0003754266 Reject
Null hypothesis
```

## Using FirstSampleName and SecondSampleVariance Arguments

### TD\_FTest Input: insect2cols

```
groupName groupValue
-----
groupE 34
groupF 15
groupE 31
groupF 26
groupE 21
groupF 26
groupE 11
groupF 10
...
groupD 12
groupB 21
groupD 12
groupB 19
groupD 5
groupB 21
...
groupC 13
groupC 11
groupC 10
groupC 21
groupC 11
groupA 20
groupA 17
groupA 20
groupA 23
groupA 14
...
...
```

### TD\_FTest SQL Call

```
SELECT * FROM TD_FTest(
    ON insect2cols AS InputTable
    USING
        SampleNameColumn('groupName')
        SampleValueColumn('groupValue')
        FirstSampleName('groupE')
        SecondSampleVariance(100.0)
        df2(25)
) AS dt;
```

## TD\_FTest Output

```
firstsamplevariance secondsamplevariance varianceratio df1 df2 CriticalValue Alpha p_value Conclusion
----- ----- ----- ----- ----- ----- ----- -----
305.2888888889 100 3.0528888889 9 25 2.6766418069 0.05 0.0263040598 Reject
Null hypothesis
```

## Using SecondSampleName and FirstSampleVariance Arguments

### TD\_FTest Input: insect2cols

```
groupName groupValue
-----
groupE    34
groupF    15
groupE    31
groupF    26
groupE    21
groupF    26
groupE    11
groupF    10
...
groupD    12
groupB    21
groupD    12
groupB    19
groupD    5
groupB    21
...
groupC    13
groupC    11
groupC    10
groupC    21
groupC    11
groupA    20
groupA    17
groupA    20
groupA    23
groupA    14
...
...
```

### TD\_FTest SQL Call

```
SELECT * FROM TD_FTest(
  ON insect2cols AS InputTable
  USING
    SampleNameColumn('groupName')
    SampleValueColumn('groupValue')
    FirstSampleVariance(85.0)
    df1(19)
    SecondSampleName('groupC')
) AS dt;
```

### TD\_FTest Output

firstsamplervariance	secondsamplervariance	varianceratio	df1	df2	CriticalValue	Alpha	p_value	Conclusion
85	19.788888889	4.2953396968	19	9	3.6833380832	0.05	0.0300307396	Reject
Null hypothesis								

## TD\_ZTest

TD\_ZTest performs a Z-test. A Z-test is a statistical hypothesis test that you can use to determine whether two population means are different when the population standard deviation or variance is known. The test is based on the Z-statistic, which is the number of standard deviations that the sample mean is from the population mean.

TD\_ZTest tests the equality of two means under the assumption that the population variances are known (rarely true). For large samples, sample variances approximate population variances, so TD\_ZTest uses sample variances instead of population variances in the test statistic.

## **Assumptions**

- Sample distribution is normal.
- Data is numeric, not categorical.

## **Test Type**

- One-tailed or two-tailed (your choice)
- One-sample or two-sample (your choice)

Use one-sample to test whether the mean of a population is greater than, less than, or not equal to a specific value. TD\_ZTest compares the critical values of the normal distribution at levels of significance (alpha = 0.01, 0.05, 0.10) to the Z-test statistic.

- Unpaired

## **Computational Method**

A test of the hypothesis involves the following framework:

1. A null hypothesis  $H_0$  and an alternative hypothesis  $H_1$
2. A random sample  $x_1, x_2, \dots, x_n$  in the case of a one sample test
3. Two random samples  $x_1, x_2, \dots, x_n$  and  $y_1, y_2, \dots, y_n$  in the case of a two sample test
4. A test statistic  $Z_{\text{stat}}$
5. A level of significance  $\alpha \in \{0.10, 0.05, 0.01\}$
6. Compare the sample based  $Z_{\text{stat}}$  with the percentage point of the normal distribution  $|z|$  or  $|z_{\alpha/2}|$
7. Compute the p-value
8. Conclusion

The Z-test is commonly used in situations where the sample size is large (typically greater than 30) and the population standard deviation is known. The steps involved in performing a Z-test include defining the null and alternative hypotheses, calculating the test statistic, determining the critical value or p-value, and making a decision about whether to reject or fail to reject the null hypothesis based on the level of significance chosen for the test.

Some uses of TD\_ZTest include:

- Medical research: A pharmaceutical company wants to test the effectiveness of a new drug on blood pressure. The company can use TD\_ZTest to determine whether the mean blood pressure of patients who took the drug is significantly different from the mean blood pressure of patients who received a placebo.

- Marketing research: A company wants to determine if there is a significant difference in the mean sales of two different products. The company can use TD\_ZTest to determine whether the difference in means is statistically significant.
- Quality control: A manufacturer wants to test whether a new production process results in a different mean defect rate than the previous process. The manufacturer can use TD\_ZTest to compare the mean defect rate of the two processes.
- Education research: A researcher wants to determine whether there is a significant difference in the mean test scores of two groups of students who received different teaching methods. The researcher can use TD\_ZTest to compare the mean test scores of the two groups.
- Finance research: A stock analyst wants to test whether the mean return of a particular stock is significantly different from the mean return of the market as a whole. The analyst could use a Z-test to compare the mean returns of the stock and the market.

TD\_ZTest provides valuable insights for decision-making in a wide range of contexts. TD\_ZTest is a useful tool in statistical analysis for determining whether there is a significant difference between two population means, and it can provide valuable insights for researchers and analysts.

### Z-test Formula

$$Z = (\bar{x}_1 - \bar{x}_2) / (\sigma / \sqrt{n})$$

where:

- $\bar{x}_1$  and  $\bar{x}_2$  are the means of the two samples you want to compare
- $\sigma$  is the standard deviation of the population
- $n$  is the sample size of each group

### TD\_ZTest Use with Scalar Values

You can use TD\_ZTest to compare the heights of two different groups of people. You take a random sample of 25 people from each group, and you find that the mean height of Group 1 is 68 inches and the mean height of Group 2 is 72 inches. You also know that the standard deviation of the population is 3 inches.

Using the formula:

$$\begin{aligned} Z &= (\bar{x}_1 - \bar{x}_2) / (\sigma / \sqrt{n}) \\ Z &= (68 - 72) / (3 / \sqrt{25}) \\ Z &= -4 / 0.6 \\ Z &= -6.67 \end{aligned}$$

This Z-score indicates that the difference between the two sample means is large and statistically significant. To determine the level of significance and whether to reject or fail to reject the null hypothesis, we would compare this Z-score to a critical value from a Z-table or calculate the p-value associated with this Z-score using statistical software.

### TD\_ZTest Use with Vector Values

You can use TD\_ZTest to test whether the means of two or more vectors are significantly different from each other. In this case, it involves computing the mean and standard deviation of the combined vector.

Suppose you have two vectors of values representing the test scores of two different classes of students. The first vector contains the scores of Class A, and the second vector contains the scores of Class B.

Here are the test scores of Class A: [75, 80, 85, 90, 95]

Here are the test scores of Class B: [60, 65, 70, 75, 80]

To test whether there is a significant difference between the mean test scores of these two classes, you can use the TD\_ZTest for two vectors.

Compute the mean and standard deviation of the combined vector:

Mean:  $(75 + 80 + 85 + 90 + 95 + 60 + 65 + 70 + 75 + 80) / 10 = 77.5$

Standard deviation: 10.7 (calculated using a spreadsheet software or calculator)

Compute the Z-score:

$$Z = (\bar{x}_1 - \bar{x}_2) / (\sigma / \sqrt{n})$$

$$Z = (82.5 - 70) / (10.7 / \sqrt{5})$$

$$Z = 2.68$$

Use a Z-table or statistical software to determine the p-value associated with this Z-score.

## Function Information

- [TD\\_ZTest Syntax](#)
- [Required Syntax Elements for TD\\_ZTest](#)
- [Optional Syntax Elements for TD\\_ZTest](#)
- [TD\\_ZTest Input](#)
- [TD\\_ZTest Output](#)
- [TD\\_ZTest Usage Notes](#)
- [Examples: How to Use TD\\_ZTest](#)

## TD\_ZTest Syntax

```
TD_ZTest (
    ON { table | view | (query) }
    USING
    {
        FirstSampleColumn('FirstSampleColumn')
        [ SecondSampleColumn('SecondSampleColumn') ]
    |
        SampleNameColumn('sample_name_column')
        SampleValueColumn('sample_value_column')
        FirstSampleName('sample_name_1')
        [ SecondSampleName('sample_name_2') ]
    }
)
```

```
[ FirstSampleVariance(FirstSampleVariance) ]
[ SecondSampleVariance(SecondSampleVariance) ]
[ AlternativeHypothesis({'upper-tailed'|'lower-tailed'|'two-tailed'}) ]
[ MeanUnderH0(MeanUnderH0) ]
[ Alpha(alpha) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_ZTest

### **ON clause**

Specifies the table name, view name or query.

### **FirstSampleColumn**

Specifies the input column name that contains the data for the first sample population.

### **SampleNameColumn**

Required for sample-value input format. Cannot be specified if FirstSampleColumn is specified. Specifies the column that contains the groups names.

### **SampleValueColumn**

Required for sample-value input format. Cannot be specified if FirstSampleColumn is specified. Specifies the column that contains the values for each group.

### **FirstSampleName**

Required for sample-value input format. Specifies the first sample name, must exist in SampleColumn.

## Optional Syntax Elements for TD\_ZTest

### **SecondSampleColumn**

Specifies the input column name that contains the data for the second sample population.

**SecondSampleName**

Specifies the second sample name, must exist in SampleColumn.

**FirstSampleVariance**

[Required if first sample size is less than 30, optional otherwise.] Specifies the variance of the first sample population. *variance\_1* is a numeric value in the range (0, 1.79769e+308).

Default behavior: If sample size is greater than 30, the function approximates the variance.

**SecondSampleVariance**

[Required if you specify SecondSampleColumn and second sample size is less than 30, optional otherwise.] Specifies the variance of the second sample population. *variance\_2* is a numeric value in the range (0, 1.79769e+308).

Default behavior: If sample size is greater than 30, the function approximates the variance.

**AlternativeHypothesis**

Specifies the alternative hypothesis:

Option	Description
'upper-tailed'	Alternate hypothesis ( $H_1$ ): $\mu > \mu_0$
'lower-tailed'	Alternate hypothesis ( $H_1$ ): $\mu < \mu_0$
'two-tailed'	Rejection region is on two sides of sampling distribution of test statistic. Two-tailed test considers both lower and upper tails of distribution of test statistic. Alternate hypothesis ( $H_1$ ): $\mu \neq \mu_0$

Default: 'two-tailed'

**MeanUnderH0**

Specifies the mean under the null hypothesis ( $H_0$ ). *mean\_under\_H0* is a numeric value in the range (-1.79769e+308, 1.79769e+308).

Default: 0

**Alpha**

The null hypothesis is rejected if the P-value is smaller than the specified Alpha value (where Alpha is the probability of rejecting the null hypothesis when it is true). *alpha* must be a numeric value in the range [0, 1].

The null hypothesis is rejected if *p\_value* is less than *alpha*. (For a description of *p\_value*, see [TD\\_ZTest Output](#).) If the null hypothesis is rejected, the rejection *confidence level* is  $1 - \alpha$ .

Default: 0.05

## TD\_ZTest Input

### Option 1 - Multiple Column Input Using FirstSampleColumn or SecondSampleColumn Argument

Input	Description
InputTable	A table that contains columns of samples.

### Input Table Schema

Column	Data Type	Description
FirstSampleColumn	DOUBLE	Numerical value used to calculate ZTest.
SecondSampleColumn	DOUBLE	Numerical value used to calculate ZTest.

### Option 2 - Multiple Column Input Using FirstSampleColumn or SecondSampleColumn Argument

Input	Description
InputTable	Table with numeric datatypes in columns. This table needs at least 2 columns.

### Input Table Schema

Column	Data Type	Description
sample_name_column	VARCHAR	Instance group name.
sample_value_column	DOUBLE	Instance numerical value used to calculate FTest.

## TD\_ZTest Output

### Output Table Schema

Column	Data Type	Description
sample_column_1	VARCHAR	Data for first sample population.
sample_column_2	VARCHAR	[Column appears only if you specify SecondSampleColumn.] Data for second sample population.
N1	INTEGER	Size of first sample.
N2	INTEGER	[Column appears only if you specify SecondSampleColumn.] Size of second sample.
mean1	DOUBLE PRECISION	Mean of first sample.

Column	Data Type	Description
mean2	DOUBLE PRECISION	[Column appears only if you specify SecondSampleColumn.] Mean of second sample.
AlternativeHypothesis	VARCHAR	Hypothesis accepted if null hypothesis is rejected ( $H_1$ ).
z_score	DOUBLE PRECISION	Test statistic z score.
Alpha	DOUBLE PRECISION	Alpha value (see <a href="#">Required Syntax Elements for TD_ZTest</a> ).
CriticalValue	DOUBLE PRECISION	Critical value calculated using Alpha for test ( $z_\alpha$ ).
p_value	DOUBLE PRECISION	Probability associated with Z-test statistic: <ul style="list-style-type: none"> <li>One-tailed, lower-tailed test: This is the lower-tail probability under normal distribution.</li> <li>One-tailed, upper-tailed test: This is the upper-tail probability under normal distribution.</li> <li>Two-tailed test: If p_value is less than <math>\alpha</math>, reject null hypothesis.</li> </ul>
Conclusion	VARCHAR	Z-test result, either 'reject null hypothesis' or 'fail to reject null hypothesis'. If Conclusion is 'reject null hypothesis', rejection confidence level is 1-alpha.

## TD\_ZTest Usage Notes

### One Sample Z-Tests

Let  $x_1, x_2, \dots, x_n$  be a random sample drawn from a population with mean  $\mu$  and variance  $\sigma^2$ . Also, assume that the data follows a normal distribution  $N(\mu, \sigma^2)$ .

$H_0: \mu \leq \mu_0$

versus

$H_1: \mu > \mu_0$

or

$H_0: \mu \geq \mu_0$

versus

$H_1: \mu < \mu_0$

$H_0: \mu = \mu_0$

versus

$$H_1: \mu \neq \mu_0$$

The test statistic for testing the previous hypotheses is the Z-stat. The validity of the Z-stat is predicated on the assumption that the population variance  $\sigma^2$  is known.

The assumption of known variance is not practical because if the variance is known, then the mean  $\mu$  is known. So, if the mean  $\mu$  is known, the test is not required.

However, for large sample sizes (which is common in Big data applications), the sample variance  $s^2$  is approximately equal to the unknown variance  $\sigma^2$ . Therefore, a scenario that involves a large sample size validates the application of the Z-statistic.

The z-statistic is calculated as:

$$Z_{stat} = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

where the unknown standard deviation  $\sigma$  is replaced by the sample standard deviation

$$s = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}}$$

as  $n \rightarrow \infty$  (sample size is very large). Therefore, the z-statistic is rewritten as:

$$Z_{stat} = \frac{\bar{x} - \mu_0}{s / \sqrt{n}}$$

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

where

In case I of the upper tailed hypothesis test, the Null hypothesis is rejected if  $Z_{stat} > z_\alpha$  where  $\alpha \in \{0.10, 0.05, 0.01\}$ . In case II of the lower tailed hypothesis test, the Null hypothesis is rejected if  $Z_{stat} < z_\alpha$  where  $\alpha \in \{0.10, 0.05, 0.01\}$ . In case III of the two-tailed test, the Null hypothesis is rejected if  $Z_{stat} > z_{\alpha/2}$  and  $Z_{stat} < z_{\alpha/2}$ ,  $\alpha \in \{0.10, 0.05, 0.01\}$ .

## Two Sample Z tests

The two sample z-test is used for testing equality of means of two populations. Let  $x_1, x_2, \dots, x_{n1} \sim N(\mu_1, \sigma_1^2)$  and  $y_1, y_2, \dots, y_{n2} \sim N(\mu_2, \sigma_2^2)$  be random samples from two independent populations. The Null hypothesis  $H_0$  and the alternative hypothesis  $H_1$  respectively for a one-sided lower-tailed test is given as:

$$H_0: \mu_1 \geq \mu_2$$

versus

$$H_1: \mu_1 < \mu_2$$

$$Z_{stat} = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

The Null hypothesis is rejected if  $Z_{stat} < -z_\alpha$  where  $\alpha \in \{0.10, 0.05, 0.01\}$ . Also, note that  $-z_\alpha$  is a percentile of the normal distribution with area to its left.

A one-sided upper-tailed test is calculated as:

$$H_0: \mu_1 \leq \mu_2$$

versus

$$H_1: \mu_1 > \mu_2$$

$$Z_{stat} = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

The Null hypothesis is rejected if  $Z_{stat} > z_\alpha$  with  $\alpha \in \{0.10, 0.05, 0.01\}$ . Also, note that  $z_\alpha$  is a percentile of the normal distribution with  $(1-\alpha) \times 100$  area to its left. So,  $-z_\alpha$  puts  $100x\alpha$  area to its left.

$$H_0: \mu_1 = \mu_2$$

versus

$$H_1: \mu_1 \neq \mu_2$$

$$Z_{stat} = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

The Null hypothesis is rejected if  $Z_{stat} > z_{1-\alpha/2}$  or  $Z_{stat} < -z_{\alpha/2}$  with  $\alpha \in \{0.10, 0.05, 0.01\}$ . Also, note that  $z_{1-\alpha/2}$  is a percentile of the normal distribution with  $(1-\alpha/2) \times 100$  area to its left. So,  $-z_\alpha$  puts  $100x\alpha$  area to its left. Note  $Z_{stat} \sim N(0,1)$ .

## Examples: How to Use TD\_ZTest

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Using Two-tailed Test](#)
- [Example: Using Two Samples](#)
- [Example: Using One Sample](#)

## Using Two-tailed Test

### **TD\_ZTest Input: example\_table**

col1	col2
---	---
93	12
?	12
22	4
?	87
1	10
?	43
92	31
?	23
2	3
?	52
21	65
?	49
?	17
?	17
?	14
?	24
53	20
85	9
50	11
86	1

### **TD\_ZTest SQL Call Using Column Names**

```
SELECT * FROM TD_ZTest (
    ON example_table AS InputTable
    USING
        FirstSampleColumn ('col1')
        SecondSampleColumn ('col2')
        FirstSampleVariance (0.5)
        SecondSampleVariance (0.7)
        AlternativeHypothesis ('two-tailed')
        MeanUnderH0 (-20)
```

```
Alpha (0.05)
) AS dt;
```

## TD\_ZTest SQL Call Using Column Number Addresses

```
SELECT * FROM TD_ZTest (
    ON example_table AS InputTable
    USING
        FirstSampleColumn ('[0]')
        SecondSampleColumn ('[1]')
        FirstSampleVariance (0.5)
        SecondSampleVariance (0.7)
        AlternativeHypothesis ('two-tailed')
        MeanUnderH0 (-20)
        Alpha (0.05)
    ) AS dt;
```

## TD\_ZTest Output

firstsamplecolumn	secondsamplecolumn	N1	N2	mean1	mean2	AlternativeHypothesis	z_score	Alpha	CriticalValue
p_value	Conclusion								
col1	col2	10	20	50.5	25.2	TWO-TAILED	155.38	0.05	1.96
Reject Null hypothesis									0.0

```
.sidetitles view
```

```
firstsamplecolumn col1
secondsamplecolumn col2
N1 10
N2 20
mean1 5.05000000000000E 001
mean2 2.52000000000000E 001
AlternativeHypothesis TWO-TAILED
z_score 1.55377718139113E 002
Alpha 5.00000000000000E-002
CriticalValue 1.95996398454005E 000
p_value 0.00000000000000E 000
Conclusion Reject Null hypothesis
```

## Example: Using Two Samples

### TD\_ZTest Input: boston2cols

The following example assumes the boston2cols input table exists:

groupName	groupValue
RM	-0.2276075065
RM	-0.4401326616
RM	-1.1572243415
RM	2.4224510607
RM	0.1294925842
...	...
NOX	-0.1467014464
NOX	-0.1467014464
NOX	1.1684487614
NOX	0.421782837
NOX	1.1684487614
...	...

### TD\_ZTest SQL Call

```
SELECT * FROM TD_ZTest(
    ON boston2cols AS InputTable
    USING
        FirstSampleName('NOX')
        SecondSampleName('RM')
        SampleNameColumn('groupName')
        SampleValueColumn('groupValue')
) AS dt;
```

### TD\_ZTest Output

firstsamplecolumn	secondsamplecolumn	N1	N2	mean1	mean2	AlternativeHypothesis	z_score	Alpha	CriticalValue	p_value	Conclusion
NOX	RM	360	360	-0.0008249076	0.0008249076	TWO-TAILED	-0.0474590359	0.05	1.9599639845	0.9621473781	Fail to reject Null hypothesis

### Example: Using One Sample

#### TD\_ZTest Input: boston2cols

The following example assumes the boston2cols input table exists:

groupName	groupValue
RM	-0.2276075065

```
----- -----  

RM      -0.2276075065  

RM      -0.4401326616  

RM      -1.1572243415  

RM      2.4224510607  

RM      0.1294925842  

...     ...  

NOX     -0.1467014464  

NOX     -0.1467014464  

NOX     1.1684487614  

NOX     0.421782837  

NOX     1.1684487614  

...     ...
```

## TD\_ZTest SQL Call

```
SELECT * FROM TD_ZTest(  

    ON boston2cols AS InputTable  

    USING  

        SampleNameColumn('groupName')  

        SampleValueColumn('groupValue')  

        FirstSampleName('NOX')  

) AS dt;
```

## TD\_ZTest Output

firstsamplecolumn	N1	mean1	AlternativeHypothesis	z_score	Alpha	CriticalValue	p_value	Conclusion
NOX	360	-0.0027042831	TWO-TAILED	-0.0515652345	0.05	1.9599639845	0.9588751214	Fail to reject
Null hypothesis								

# Association Analysis Functions

Association rule learning is a rule-based machine learning method for discovering relations between variables in large databases.

- [TD\\_CFilter](#)

## TD\_CFilter

The CFilter function calculates several statistical measures of how likely each pair of items is to be purchased together. A typical input table for the CFilter function is a set of sales transactions, with a column of purchased items and a column of something by which to group the purchased items.

### Note:

- This function requires the UTF8 client character set for UNICODE data.
- This function does not support Pass Through Characters (PTCs).  
For information about PTCs, see *International Character Set Support*, B035-1125.
- This function does not support KanjiSJIS or Graphic data types.

- 
- [TD\\_CFilter Syntax](#)
  - [Required Syntax Elements for TD\\_CFilter](#)
  - [Optional Syntax Elements for TD\\_CFilter](#)
  - [TD\\_CFilter Input](#)
  - [TD\\_CFilter Output](#)
  - [TD\\_CFilter Usage Notes](#)
  - [Example: How to Use TD\\_CFilter](#)

## TD\_CFilter Syntax

```
TD_CFilter (
    ON { table | view | (query) } AS InputTable
    USING
        TargetColumn('target_column')
        TransactionIDColumns('transactionid_Column' [,...])
        [ PartitionColumns('partition_column' [,...]) ]
        [ MaxDistinctItems(max_distinct_items) ]
)
```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for TD\_CFilter

The following elements are required when using TD\_CFilter.

### **ON clause**

Accept the InputTable clause.

### **TargetColumn**

Specify the name of the InputTable column that contain the data to filter.

Maximum 1 column is allowed.

### **TransactionIDColumns**

Specify the names of InputTable column that contain the transactionID which groups the items that are purchased together.

Maximum 2047 columns are allowed.

## Optional Syntax Elements for TD\_CFilter

The following elements are optional when using TD\_CFilter.

### **PartitionColumns**

Specify the names of the input columns to copy to the output table. The function partitions the input data and the output table on these columns.

Maximum 10 columns are allowed.

**Note:**

PartitionColumns makes TD\_CFilter output nondeterministic unless each partition\_column is unique in the group defined by TransactionIDColumns.

### **MaxDistinctItems**

Specify the maximum size of the item set.

Default: 100

## TD\_CFilter Input

### Input Table Schema

Column Name	Data Type	Description
target_column	CHAR/VARCHAR	Column that contains the data to filter.
transactionid_column	ANY (allowed by Partition By Clause)	Column that contains the transactionid column.
partition_column	ANY (allowed by Partition By Clause)	Column to copy to output table. Used to partition input data and output table.

## TD\_CFilter Output

Column Name	Data Type	Description
partition_column	ANY	Column to copy to output table. Used to partition input data and output table.
TD_item1	VARCHAR	Name of item1.
TD_item2	VARCHAR	Name of item2.
cntb	INTEGER	Count of co-occurrence of both items in partition.
cnt1	INTEGER	Count of occurrence of item1 in partition.
cnt2	INTEGER	Count of occurrence of item2 in partition.
score	REAL	Product of two conditional probabilities: $P(\{ \text{item2}   \text{item1} \}) * P(\{ \text{item1}   \text{item2} \})$ Preceding product equals following quotient: $(\text{cntb} * \text{cntb}) / (\text{cnt1} * \text{cnt2})$
support	REAL	Percentage of transactions in partition in which the two items co-occur, calculated with this formula: $\text{cntb} / \text{tran\_cnt}$ where $\text{tran\_cnt}$ is the number of transactions in the partition.
confidence	REAL	Percentage of transactions in partition in which item1 occurs, in which item2 also occurs, calculated with this formula: $\text{cntb} / \text{cnt1}$
lift	REAL	Ratio of observed support value to expected support value if item1 and item2 were independent; that is: $\text{support(item1 and item2)} / [\text{support(item1)} * \text{support(item2)}]$

Column Name	Data Type	Description
		<p>Value is calculated with this formula:  <math>(cntb/tran\_cnt) / [(cnt1/tran\_cnt) * (cnt2/tran\_cnt)]</math></p> <p>If lift &gt; 1, the occurrence of item1 or item2 has a positive effect on the occurrence of the other items.</p> <p>If lift = 1, the occurrence of item1 or item2 has a no effect on the occurrence of the other items.</p> <p>If lift &lt; 1, the occurrence of item1 or item2 has a negative effect on the occurrence of the other items.</p>
z_score	REAL	<p>Significance of co-occurrence, assuming that cntb follows a normal distribution, calculated with this formula:  <math>(cntb - mean(cntb)) / sd(cntb)</math></p> <p>If all cntb values are equal, then <math>sd(cntb)</math> is 0, and function does not calculate zscore.</p>

## TD\_CFilter Usage Notes

- The InputTable in TD\_CFilter query can have no partition at all or have the PARTITION BY ANY clause.
- TargetColumn argument is mandatory for the TD\_CFilter function. Otherwise, an error is reported.
- TransactionIDColumn argument is mandatory for the TD\_CFilter function. Otherwise, an error is reported.
- Number of Target column supported is 1. If more than 1 target\_column is provided, an error is reported.
- Number of Partition column supported is 10. If more than 10 partition\_column are provided, an error is reported.

## Example: How to Use TD\_CFilter

### Input Table

The following table contains grocery\_transaction data.

```

DROP TABLE grocery_transaction;
CREATE TABLE grocery_transaction (tranid int, period varchar(20), storeid
int, region varchar(20), item varchar(20), sku int, category varchar(20))
Primary index(tranid);
insert into
grocery_transaction values(999,'20100715',1,'west','milk',1,'dairy');
insert into
grocery_transaction values(999,'20100715',1,'west','butter',2,'dairy');
insert into
grocery_transaction values(999,'20100715',1,'west','eggs',3,'dairy');

```

```

insert into
grocery_transaction values(999,'19990715',1,'west','flour',4,'baking');
insert into
grocery_transaction values(999,'19990715',1,'west','spinach',4,'produce');
insert into
grocery_transaction values(1000,'20100715',1,'west','milk',1,'dairy');
insert into
grocery_transaction values(1000,'20100715',1,'west','eggs',3,'dairy');
insert into
grocery_transaction values(1000,'19990715',1,'west','flour',4,'baking');
insert into
grocery_transaction values(1000,'19990715',1,'west','spinach',2,'produce');
insert into
grocery_transaction values(1001,'20100715',1,'west','milk',1,'dairy');
insert into
grocery_transaction values(1001,'20100715',1,'west','butter',2,'dairy');
insert into
grocery_transaction values(1001,'20100715',1,'west','eggs',3,'dairy');
insert into
grocery_transaction values(1002,'20100715',1,'west','milk',1,'dairy');
insert into
grocery_transaction values(1002,'20100715',1,'west','butter',2,'dairy');
insert into
grocery_transaction values(1002,'20100715',1,'west','spinach',3,'produce');
insert into
grocery_transaction values(1500,'20100715',3,'west','butter',2,'dairy');
insert into
grocery_transaction values(1500,'20100715',3,'west','eggs',3,'dairy');
insert into
grocery_transaction values(1500,'20100715',3,'west','flour',4,'baking');

```

## SELECT Statement

```
SELECT * from grocery_transaction order by 1;
```

Result:

tranid	period	storeid	region	item	sku	category
999	20100715	1	west	eggs	3	dairy
999	19990715	1	west	spinach	4	produce
999	19990715	1	west	flour	4	baking
999	20100715	1	west	butter	2	dairy

tranid	period	storeid	region	item	sku	category
999	20100715	1	west	milk	1	dairy
1000	19990715	1	west	flour	4	baking
1000	19990715	1	west	spinach	2	produce
1000	20100715	1	west	eggs	3	dairy
1000	20100715	1	west	milk	1	dairy
1001	20100715	1	west	eggs	3	dairy
1001	20100715	1	west	butter	2	dairy
1001	20100715	1	west	milk	1	dairy
1002	20100715	1	west	spinach	3	produce
1002	20100715	1	west	butter	2	dairy
1002	20100715	1	west	milk	1	dairy
1500	20100715	3	west	flour	4	baking
1500	20100715	3	west	eggs	3	dairy
1500	20100715	3	west	butter	2	dairy

### Query 1 - Use Case without PartitionColumns

```
SELECT * FROM TD_CFilter (
    ON grocery_transaction AS InputTable
    USING
        TargetColumn ('item')
        TransactionIDColumns ('tranid')
        MaxDistinctItems(100)
    ) AS dt Order By 1,2;
```

Result:

TD_item1	TD_item2	cntb	cnt1	cnt2	score	support	confidence	lift	z_score
butter	eggs	3	4	4	5.625000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	9.375000000000000E-01	1.000000000000000E000
butter	flour	2	4	3	3.33333333333330E-01	4.000000000000000E-01	5.000000000000000E-01	8.33333333333330E-01	-1.000000000000000E000
butter	milk	3	4	4	5.625000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	9.375000000000000E-01	1.000000000000000E000
butter	spinach	2	4	3	3.33333333333330E-01	4.000000000000000E-01	5.000000000000000E-01	8.33333333333330E-01	-1.000000000000000E000
eggs	butter	3	4	4	5.625000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	9.375000000000000E-01	1.000000000000000E000
eggs	flour	3	4	3	7.500000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	1.250000000000000E000	1.000000000000000E000
eggs	milk	3	4	4	5.625000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	9.375000000000000E-01	1.000000000000000E000
eggs	spinach	2	4	3	3.33333333333330E-01	4.000000000000000E-01	5.000000000000000E-01	8.33333333333330E-01	-1.000000000000000E000
flour	butter	2	3	4	3.33333333333330E-01	4.000000000000000E-01	6.66666666666670E-01	8.33333333333330E-01	-1.000000000000000E000
flour	eggs	3	3	4	7.500000000000000E-01	6.000000000000000E-01	1.000000000000000E000	1.250000000000000E000	1.000000000000000E000
flour	milk	2	3	4	3.33333333333330E-01	4.000000000000000E-01	6.66666666666670E-01	8.33333333333330E-01	-1.000000000000000E000
flour	spinach	2	3	3	4.44444444444440E-01	4.000000000000000E-01	6.66666666666670E-01	1.1111111111111E000	-1.000000000000000E000
milk	butter	3	4	4	5.625000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	9.375000000000000E-01	1.000000000000000E000
milk	eggs	3	4	4	5.625000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	9.375000000000000E-01	1.000000000000000E000
milk	flour	2	4	3	3.33333333333330E-01	4.000000000000000E-01	5.000000000000000E-01	8.33333333333330E-01	-1.000000000000000E000
milk	spinach	3	4	3	7.500000000000000E-01	6.000000000000000E-01	7.500000000000000E-01	1.250000000000000E000	1.000000000000000E000
spinach	butter	2	3	4	3.33333333333330E-01	4.000000000000000E-01	6.66666666666670E-01	8.33333333333330E-01	-1.000000000000000E000

<b>TD_item1</b>	<b>TD_item2</b>	<b>cntb</b>	<b>cnt1</b>	<b>cnt2</b>	<b>score</b>	<b>support</b>	<b>confidence</b>	<b>lift</b>	<b>z_score</b>
spinach	eggs	2	3	4	3.3333333333330E-01	4.000000000000000E-01	6.66666666666670E-01	8.3333333333330E-01	-1.00000000000000E 000
spinach	flour	2	3	3	4.44444444444440E-01	4.00000000000000E-01	6.66666666666670E-01	1.1111111111111E 000	-1.00000000000000E 000
spinach	milk	3	3	4	7.50000000000000E-01	6.00000000000000E-01	1.00000000000000E 000	1.25000000000000E 000	1.00000000000000E 000

**Query 2 - Use Case with PartitionColumns**

```
SELECT * FROM TD_CFilter (
    ON grocery_transaction AS InputTable
    USING
        TargetColumn ('item')
        TransactionIDColumns ('tranid')
        PartitionColumns ('storeid')
        MaxDistinctItems(100)
) AS dt Order By 1,2,3;
```

Result:

storeid	TD_item1	TD_item2	cnty	cnt1	cnt2	score	support	confidence	lift	z_score
1	butter	eggs	2	3	3	4.44444444444440E-01	5.00000000000000E-01	6.66666666666670E-01	8.8888888888890E-01	-3.3333333333340E-01
1	butter	flour	1	3	2	1.66666666666670E-01	2.50000000000000E-01	3.3333333333330E-01	6.66666666666670E-01	-2.0000000000000E 000
1	butter	milk	3	3	4	7.50000000000000E-01	7.50000000000000E-01	1.00000000000000E 000	1.00000000000000E 000	1.333333333333E 000
1	butter	spinach	2	3	3	4.44444444444440E-01	5.00000000000000E-01	6.66666666666670E-01	8.8888888888890E-01	-3.3333333333340E-01
1	eggs	butter	2	3	3	4.44444444444440E-01	5.00000000000000E-01	6.66666666666670E-01	8.8888888888890E-01	-3.3333333333340E-01
1	eggs	flour	2	3	2	6.66666666666670E-01	5.00000000000000E-01	6.66666666666670E-01	1.333333333333E 000	-3.3333333333340E-01
1	eggs	milk	3	3	4	7.50000000000000E-01	7.50000000000000E-01	1.00000000000000E 000	1.00000000000000E 000	1.333333333333E 000
1	eggs	spinach	2	3	3	4.44444444444440E-01	5.00000000000000E-01	6.66666666666670E-01	8.8888888888890E-01	-3.3333333333340E-01
1	flour	butter	1	2	3	1.66666666666670E-01	2.50000000000000E-01	5.00000000000000E-01	6.66666666666670E-01	-2.0000000000000E 000
1	flour	eggs	2	2	3	6.66666666666670E-01	5.00000000000000E-01	1.00000000000000E 000	1.333333333333E 000	-3.3333333333340E-01
1	flour	milk	2	2	4	5.00000000000000E-01	5.00000000000000E-01	1.00000000000000E 000	1.00000000000000E 000	-3.3333333333340E-01
1	flour	spinach	2	2	3	6.66666666666670E-01	5.00000000000000E-01	1.00000000000000E 000	1.333333333333E 000	-3.3333333333340E-01
1	milk	butter	3	4	3	7.50000000000000E-01	7.50000000000000E-01	7.50000000000000E-01	1.00000000000000E 000	1.333333333333E 000
1	milk	eggs	3	4	3	7.50000000000000E-01	7.50000000000000E-01	7.50000000000000E-01	1.00000000000000E 000	1.333333333333E 000
1	milk	flour	2	4	2	5.00000000000000E-01	5.00000000000000E-01	5.00000000000000E-01	1.00000000000000E 000	-3.3333333333340E-01
1	milk	spinach	3	4	3	7.50000000000000E-01	7.50000000000000E-01	7.50000000000000E-01	1.00000000000000E 000	1.333333333333E 000
1	spinach	butter	2	3	3	4.44444444444440E-01	5.00000000000000E-01	6.66666666666670E-01	8.8888888888890E-01	-3.3333333333340E-01

storeid	TD_item1	TD_item2	cnty	cnt1	cnt2	score	support	confidence	lift	z_score
1	spinach	eggs	2	3	3	4. 444444444444440E-01	5. 000000000000000E-01	6. 66666666666670E-01	8. 88888888888890E-01	-3. 33333333333340E-01
1	spinach	flour	2	3	2	6. 66666666666670E-01	5. 000000000000000E-01	6. 66666666666670E-01	1. 3333333333333E 000	-3. 33333333333340E-01
1	spinach	milk	3	3	4	7. 500000000000000E-01	7. 500000000000000E-01	1. 000000000000000E 000	1. 000000000000000E 000	1. 3333333333333E 000
3	butter	eggs	1	1	1	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	?
3	butter	flour	1	1	1	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	?
3	eggs	butter	1	1	1	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	?
3	eggs	flour	1	1	1	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	?
3	flour	butter	1	1	1	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	?
3	flour	eggs	1	1	1	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	1. 000000000000000E 000	?

# Path and Pattern Analysis Functions

- [Terminology](#)
- [Attribution](#)
- [nPath](#)
- [Sessionize](#)

## Terminology

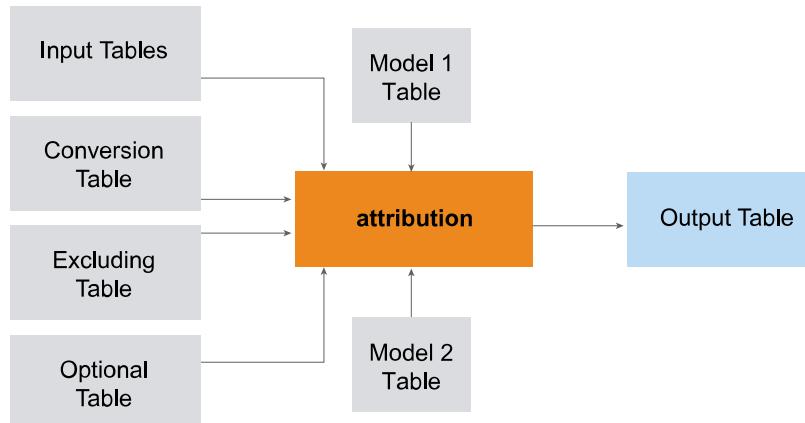
This document uses the following terms.

Term	Description
Path	An ordered, start-to-finish series of actions, for example, page views, for which sequences and sub-sequences can be created.
Sequence	A sequence is the path prefixed with a carat (^), which indicates the start of a path. For example, if a user visited page a, page b, and page c, in that order, the session sequence is ^,a,b,c.
Subsequence	For a given sequence of actions, a sub-sequence is one possible subset of the steps that begins with the initial action. For example, the path a,b, creates three subsequences: ^,a; ^,a,b; and ^,a,b,c.

## Attribution

The Attribution function is used in web page analysis, where companies assign weights to pages before certain events, such as buying a product.

The function takes data and parameters from multiple tables and outputs attributions.



Analytics Database Attribution function corresponds to the multiple-input version. Unlike Attribution\_MLE, Attribution does not support Unicode.

Attribution refers to the process of assigning credit or responsibility to a specific event or entity that contributes to an outcome of interest. For example, in finance, attribution analysis is used to understand the performance of investment portfolios and to identify the sources of returns. In cybersecurity, attribution is the process of identifying the source of a cyber attack or incident. In machine learning and artificial intelligence, attribution refers to the process of understanding which features or inputs in a model are most responsible for its output. This is useful for interpreting and debugging models. Overall, attribution is a fundamental concept in many different fields that helps organizations understand the factors that contribute to outcomes and make better decisions.

Specifically, the Attribution function is used for web page analysis, which refers to the process of assigning value or credit to different pages on a website for specific actions taken by visitors, such as making a purchase or filling out a form. The goal of attribution is to identify the most effective pages or content on a website that contribute to achieving business goals. By assigning weights or credit to different pages, organizations can optimize their website by improving or eliminating underperforming pages and investing more resources into the most effective ones. Attribution can be done using various methods, including rule-based attribution and data-driven attribution.

For example, an online store wants to determine the effectiveness of their marketing channels in driving sales. An attribution algorithm is assigned to credit each marketing channel based on its contribution to sales. To do this, the algorithm might analyze customer journeys and the various touchpoints they had before making a purchase. Touchpoints could include seeing an ad on social media, clicking on a link in an email, or doing a search.

The algorithm might use a probabilistic model, such as a Markov chain, to estimate the probability that each touchpoint led to a sale. It might find that customers who clicked on an email link were 3 times more likely to make a purchase than those who saw a social media ad.

Based on these probabilities, the algorithm would assign weights or credits to each touchpoint. In this example, the email link touchpoint would receive a higher weight than the social media ad touchpoint, because it had a higher estimated probability of contributing to a sale.

Once the algorithm has assigned credits to each touchpoint, the store can use this information to optimize their marketing efforts. They might decide to invest more in email marketing since it has been shown to be more effective in driving sales than social media advertising.

The attribution algorithm provides organizations with data-driven insights into the effectiveness of their marketing channels, allowing them to make informed decisions about how to allocate their marketing budget for maximum impact.

## Function Information

- [Attribution Syntax](#)
- [Attribution Syntax Elements](#)
- [Attribution Input](#)
- [Attribution Output](#)

- [Example: Model Assigns Attribution Weights to Events and Channels](#)

## Attribution Syntax

```
ATTRIBUTION (
    ON { table | view | (query) } [ AS InputTable1 ]
        PARTITION BY user_id
        ORDER BY times_column
    [ ON { table | view | (query) } [ AS InputTable2 ]
        PARTITION BY user_id
        ORDER BY time_column [,....] ]
    ON conversion_event_table AS ConversionEventTable DIMENSION
    [ ON excluding_event_table AS ExcludedEventTable DIMENSION ]
    [ ON optional_event_table AS OptionalEventTable DIMENSION ]
    ON model1_table AS FirstModelTable DIMENSION
    [ ON model2_table AS SecondModelTable DIMENSION ]
    USING
        EventColumn ('event_column')
        TimeColumn ('time_column')
        WindowSize ({'rows:K' | 'seconds:K' | 'rows:K&seconds:K2'})
) ORDER BY user_id,time_stamp;
```

---

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
  - As part of a CREATE TABLE statement
  - As part of a CREATE VIEW statement
- 

## Attribution Syntax Elements

### **EventColumn**

Specify the name of the input column that contains the clickstream events.

### **TimeColumn**

Specify the name of the input column that contains the timestamps of the clickstream events.

### **WindowSize**

Specify how to determine the maximum window size for the attribution calculation:

Option	Description
rows : $K$	Assign attributions to at most $K$ events before conversion event, excluding events of types specified in ExcludedEventTable.
seconds : $K$	Assign attributions only to rows not more than $K$ seconds before conversion event.
rows : $K$ & seconds : $K2$	Apply both constraints and comply with stricter one.

## Attribution Input

### Required

Table	Description
Input tables (maximum of five)	Contains clickstream data for computing attributions.
ConversionEventTable	Contains conversion events.
FirstModelTable	Defines type and distributions of first model.

### Optional

Table	Description
ExcludedEventTable	Contains events to exclude from attribution.
OptionalEventTable	Contains optional events.
SecondModelTable	Defines type and distributions of second model.

### Input Table Schema

Column	Data Type	Description
userid_column	INTEGER or VARCHAR	User identifier.
event_column	INTEGER or VARCHAR	Event from clickstream.
time_column	INTEGER, SMALLINT, BIGINT, TIMESTAMP, or TIME	Event timestamp.

### ConversionEventTable Schema

Column	Data Type	Description
conversion_event	VARCHAR	Conversion event value (string or integer).

## FirstModelTable and SecondModelTable Schema

Column	Data Type	Description
id	INTEGER	Row identifier. Rows are numbered 0, 1, 2, and so on.
model	VARCHAR	Row 0: Model type. Row 1, ..., n: Distribution model definition. SIMPLE model: Model table has single row that specifies model type and parameters. Other model types: n is number of rows or events included in model. For model type and specification definitions, see <a href="#">Model Specification</a> .

## ExcludedEventTable Schema

Column	Data Type	Description
excluding_event	VARCHAR	Excluded event (string or integer). Cannot be a conversion event.

## OptionalEventTable Schema

Column	Data Type	Description
optional_event	VARCHAR	Optional event (string or integer). Cannot be a conversion or excluded event. Function attributes conversion event to optional event only if it cannot attribute it to regular event.

## Model Specification

### Model Types and Specification Definitions

Row 0: Model Type	Row 1, ..., n: Distribution Model Specification	Additional Information
SIMPLE	MODEL: PARAMETERS	Distribution model for all events. For MODEL and PARAMETER definitions, see following table.
EVENT_REGULAR	EVENT:WEIGHT: MODEL: PARAMETERS	Distribution model for a regular event. EVENT cannot be a conversion, excluded, or optional event. For MODEL and PARAMETER definitions, see following table. Sum of WEIGHT values must be 1.0. For example, suppose that model table has these specifications: email:0.19:LAST_CLICK:NA impression:0.81:UNIFORM:NA Within WindowSize of a conversion event, 19% of conversion event is attributed to last email event and 81% is attributed uniformly to all impression events.

Row 0: Model Type	Row 1, ..., n: Distribution Model Specification	Additional Information
EVENT_OPTIONAL	EVENT:WEIGHT: MODEL: PARAMETERS	<p>Distribution model for an optional event.</p> <p>EVENT must be in optional event table.</p> <p>For MODEL and PARAMETER definitions, see following table.</p> <p>Sum of WEIGHT values must be 1.0.</p>
SEGMENT_ROWS	K <sub>j</sub> :WEIGHT: MODEL: PARAMETERS	<p>Distribution model by row. Sum of K<sub>j</sub> values must be value K specified by 'rows:K' in WindowSize syntax element.</p> <p>Function considers rows from most to least recent. For example, suppose that function call has these syntax elements:</p> <p>WindowSize ('rows:10')  Model1 ('SEGMENT_ROWS',  '3:0.5:UNIFORM:NA',  '4:0.3:LAST_CLICK:NA',  '3:0.2:FIRST_CLICK:NA')</p> <p>Attribution for a conversion event is divided among attributable events in 10 rows immediately preceding conversion event. If conversion event is in row 11, first model specification applies to rows 10, 9, and 8; second applies to rows 7, 6, 5, and 4; and third applies to rows 3, 2, and 1.</p> <p>Half attribution (5/10) is uniformly divided among rows 10, 9, and 8; 3/10 to last click in rows 7, 6, 5, and 4 (that is, in row 7), and 2/10 to first click in rows 3, 2, and 1 (that is, in row 1).</p>
SEGMENT_SECONDS	K <sub>j</sub> :WEIGHT: MODEL: PARAMETERS	<p>Distribution model by time in seconds. Sum of K<sub>j</sub>values must be value K specified by 'seconds:K' in WindowSize syntax element.</p> <p>Function considers rows from most to least recent. For example, suppose that function call has these syntax elements:</p> <p>WindowSize ('seconds:20')  Model1 ('SEGMENT_SECONDS',  '6:0.5:UNIFORM:NA',  '8:0.3:LAST_CLICK:NA',  '6:0.2:FIRST_CLICK:NA')</p> <p>Attribution for a conversion event is divided among attributable events in 20 seconds immediately preceding conversion event. If conversion event is at second 21, first model specification applies to seconds 20-15 (counting backward); second applies to seconds 14-7; and third applies to seconds 6-1.</p> <p>Half attribution (5/10) is uniformly divided among seconds 20-15; 3/10 to last click in seconds 14-7, and 2/10 to first click in seconds 6-1.</p>

## MODEL Values and Corresponding PARAMETER Values

MODEL values are case-sensitive. Attributable events are those whose types are not specified in excluding events table.

MODEL	PARAMETERS	Description
'LAST_CLICK'	'NA'	Conversion event is attributed entirely to most recent attributable event.

MODEL	PARAMETERS	Description
'FIRST_CLICK'	'NA'	Conversion event is attributed entirely to first attributable event.
'UNIFORM'	'NA'	Conversion event is attributed uniformly to preceding attributable events.
'EXPONENTIAL'	'alpha,type' where <i>alpha</i> is a decay factor in range (0, 1) and <i>type</i> is ROW, MILLISECOND, SECOND, MINUTE, HOUR, DAY, MONTH, or YEAR. When <i>alpha</i> is in range (0, 1), sum of series $w_i = (1-\alpha)^i \alpha^i$ is 1. Function uses $w_i$ as exponential weights.	Conversion event is attributed exponentially to preceding attributable events (the more recent the event, the higher the attribution).
'WEIGHTED'	You can specify any number of weights. If there are more attributable events than weights, extra (least recent) events are assigned zero weight. If there are more weights than attributable events, then function renormalizes weights.	Conversion event is attributed to preceding attributable events with weights specified by PARAMETERS. SEGMENT_SECONDS (when you specify 'rows:K&seconds:K' in WindowSize syntax element)

## Allowed FirstModelTable/SecondModelTable Combinations

FirstModelTable Type	SecondModelTable Type
SIMPLE	Not allowed
EVENT_REGULAR	
EVENT_REGULAR	EVENT_OPTIONAL (when you specify optional events table)
SEGMENT_ROWS	SEGMENT_SECONDS (when you specify 'rows:K&seconds:K' in WindowSize syntax element)
SEGMENT_ROWS	
SEGMENT_SECONDS	Not allowed

## Attribution Output

### Attribution Output Table Schema

Column	Data Type	Description
user_id	INTEGER or VARCHAR	User identifier from input table.
event	VARCHAR	Clickstream event from input table.
time_stamp	TIMESTAMP	Event timestamp from input table.

Column	Data Type	Description
attribution	DOUBLE PRECISION	Fraction of attribution for conversion event that is attributed to this event.
time_to_conversion	INTEGER	Elapsed time between attributable event and conversion event.

## Example: Model Assigns Attribution Weights to Events and Channels

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### Event Type Channels

This example uses models to assign attribution weights to these events and channels.

Event Type	Channels
conversion	SocialNetwork, PaidSearch
excluding	Email
optional	Direct, Referral, OrganicSearch

### Attribution Input

Attribution InputTable1: attribution\_sample\_table1

user_id	event	time_stamp
1	impression	2022-09-27 23:00:01
1	impression	2022-09-27 23:00:05
1	Email	2022-09-27 23:00:15
2	impression	2022-09-27 23:00:31
2	impression	2022-09-27 23:00:51

Attribution InputTable2: attribution\_sample\_table2

user_id	event	time_stamp
1	impression	2022-09-27 23:00:19
1	SocialNetwork	2022-09-27 23:00:20

<b>user_id</b>	<b>event</b>	<b>time_stamp</b>
1	Direct	2022-09-27 23:00:21
1	Referral	2022-09-27 23:00:22
1	PaidSearch	2022-09-27 23:00:23
2	impression	2022-09-27 23:00:29
2	impression	2022-09-27 23:00:31
2	impression	2022-09-27 23:00:33
2	impression	2022-09-27 23:00:36
2	impression	2022-09-27 23:00:38

**Attribution ConversionEventTable: conversion\_event\_table**

<b>conversion_events</b>
PaidSearch
SocialNetwork

**Attribution ExcludedEventTable: excluding\_event\_table**

<b>excluding_events</b>
Email

**Attribution OptionalEventTable: optional\_event\_table**

<b>optional_events</b>
Direct
OrganicSearch
Referral

The following two model tables apply the distribution models by rows and by seconds, respectively.

**Attribution FirstModelTable: model1\_table**

<b>id</b>	<b>model</b>
0	SEGMENT_ROWS
1	3:0.5:EXPONENTIAL:0.5,SECOND
2	4:0.3:WEIGHTED:0.4,0.3,0.2,0.1
3	3:0.2:FIRST_CLICK:NA

**Attribution SecondModelTable: model2\_table**

<b>id</b>	<b>model</b>
0	SEGMENT_SECONDS
1	6:0.5:UNIFORM:NA
2	8:0.3:LAST_CLICK:NA
3	6:0.2:FIRST_CLICK:NA

**Attribution SQL Call**

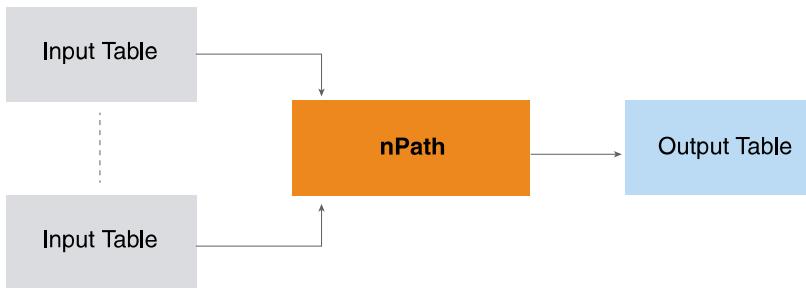
```
SELECT * FROM ATTRIBUTION (
    ON attribution_sample_table1 AS InputTable1
        PARTITION BY user_id ORDER BY time_stamp
    ON attribution_sample_table2 AS InputTable2
        PARTITION BY user_id ORDER BY time_stamp
    ON conversion_event_table AS ConversionEventTable DIMENSION
    ON excluding_event_table AS ExcludedEventTable DIMENSION
    ON optional_event_table AS OptionalEventTable DIMENSION
    ON model1_table AS FirstModelTable DIMENSION
    ON model2_table AS SecondModelTable DIMENSION
    USING
        EventColumn ('event')
        TimeColumn ('time_stamp')
        WindowSize ('rows:10&seconds:20')
) AS dt ORDER BY user_id, time_stamp;
```

**Attribution Output**

<b>user_id</b>	<b>event</b>	<b>time_stamp</b>	<b>attribution</b>	<b>time_to_conversion</b>
1	impression	2022-09-27 23:00:01	0.285714	-19
1	impression	2022-09-27 23:00:05	0	?
1	impression	2022-09-27 23:00:19	0.714286	-1
1	SocialNetwork	2022-09-27 23:00:20	?	?
1	Direct	2022-09-27 23:00:21	0.5	-2
1	Referral	2022-09-27 23:00:22	0.5	-1
1	PaidSearch	2022-09-27 23:00:23	?	?

## nPath

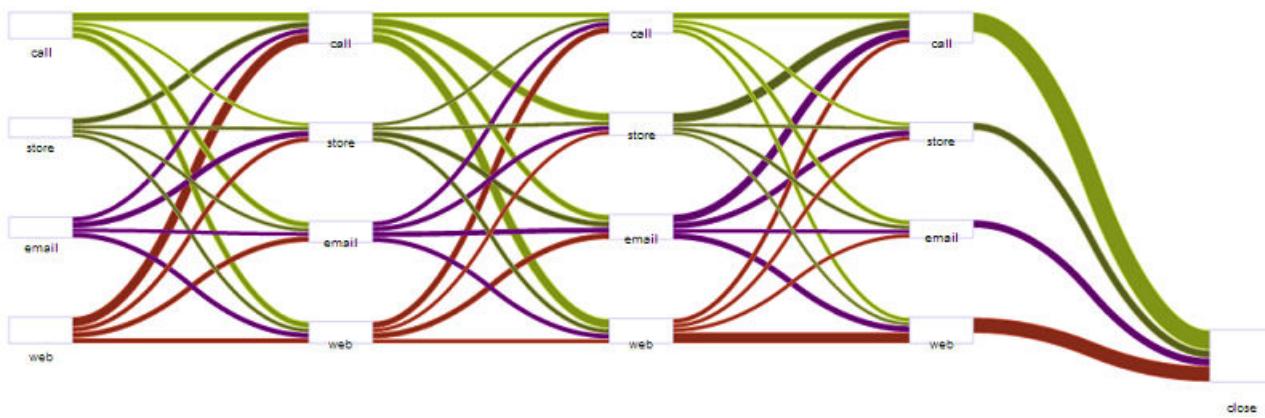
The nPath function scans a set of rows, looking for patterns that you specify. For each set of input rows that matches the pattern, nPath produces a single output row. The function provides a flexible pattern-matching capability that lets you specify complex patterns in the input data and define the values that are output for each matched input set.



nPath is useful when your goal is to identify the paths that lead to an outcome. For example, you can use nPath to analyze:

- Web site click data, to identify paths that lead to sales over a specified amount
- Sensor data from industrial processes, to identify paths to poor product quality
- Healthcare records of individual patients, to identify paths that indicate that patients are at risk of developing conditions such as heart disease or diabetes
- Financial data for individuals, to identify paths that provide information about credit or fraud risks

**Sankey Diagram of Analytics Database nPath Output**



An nPath call specifies:

- [Mode \(overlapping or nonoverlapping\)](#)
- [Pattern to match](#)
- [Symbols to use](#)
- [Optional] [Filters to apply](#)

- [Results to output](#)
- 

**Note:**

- This function requires the UTF8 client character set for UNICODE data.
  - This function does not support UNICODE input/output data for ACCUMULATE function of Result syntax element.
  - This function does not support UNICODE characters for Pattern syntax element and symbol in Symbols syntax element.
  - This function does not support Pass Through Characters (PTCs).  
For information about PTCs, see *International Character Set Support*, B035-1125.
  - This function does not support KanjiSJIS or Graphic data types.
  - When used with this function, the ORDER BY clause supports only ASCII collation.
  - When used with this function, the PARTITION BY clause assumes column names are in Normalization Form C (NFC).
- 

**Function Information**

- [nPath Syntax](#)
- [nPath Syntax Elements](#)
- [nPath Input](#)
- [nPath Output](#)
- [nPath Symbols](#)
- [nPath Patterns](#)
- [nPath Filters](#)
- [nPath Results](#)
- [nPath Examples](#)

**nPath Syntax**

```

nPath (
    ON { table | view | (query) }
        PARTITION BY partition_column
        ORDER BY order_column [ ASC | DESC ][...]
    [ ON { table | view | (query) }
        [ PARTITION BY partition_column | DIMENSION ]
        ORDER BY order_column [ ASC | DESC ]
    ][...]
    USING
    Mode ({ OVERLAPPING | NONOVERLAPPING })
    Pattern ('pattern')

```

```

Symbols ({ col_expr = symbol_predicate AS symbol }[, ...])
[ Filter (filter_expression[, ...]) ]
Result ({ aggregate_function (expression OF [ANY] symbol [, ...]) AS alias_1 }[, ...])
)

```

**Note:**

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## nPath Syntax Elements

### Mode

Specify the pattern-matching mode:

Option	Description
OVERLAPPING	Find every occurrence of pattern in partition, regardless of whether it is part of a previously found match. One row can match multiple symbols in a given matched pattern.
NONOVERLAPPING	Start next pattern search at row that follows last pattern match.

### Pattern

Specify the pattern for which the function searches. You compose *pattern* with the symbols (which you define in the Symbols syntax element), operators, and parentheses.

When patterns have multiple operators, the function applies them in order of precedence, and applies operators of equal precedence from left to right. To force the function to evaluate a subpattern first, enclose it in parentheses. For more information, see [nPath Patterns](#).

### Symbols

Defines the symbols that appear in the values of the Pattern and Result syntax elements. The *col\_expr* is an expression whose value is a column name, *symbol* is any valid identifier, and *symbol\_predicate* is a SQL predicate (often a column name).

Each *col\_expr = symbol\_predicate* must satisfy the SQL syntax of the Analytics Database when nPath is invoked. Otherwise, it is a syntax error.

For example, this Symbols syntax element is for analyzing website visits:

```

Symbols (
    pagetype = 'homepage' AS H,
    pagetype <> 'homepage' AND pagetype <> 'checkout' AS PP,
    pagetype = 'checkout' AS CO
)

```

The *symbol* is case-insensitive; however, a *symbol* of one or two uppercase letters is easy to identify in patterns.

If *col\_expr* represents a column that appears in multiple input tables, you must qualify the ambiguous column name with its table name. For example:

```

Symbols (
    weblog.pagetype = 'homepage' AS H,
    weblog.pagetype = 'thankyou' AS T,
    ads.adname = 'xmaspromo' AS X,
    ads.adname = 'realtorpromo' AS R
)

```

For more information about symbols that appear in the Pattern syntax element value, see [nPath Symbols](#). For more information about symbols that appear in the Result syntax element value, see [nPath Results](#).

## Filter

[Optional] Specify filters to impose on the matched rows. The function combines the filter expressions using the AND operator.

This is the *filter\_expression* syntax:

```
symbol_expression comparison_operator symbol_expression
```

The two symbol expressions must be type-compatible. This is the *symbol\_expression* syntax:

```
{ FIRST | LAST }(column_with_expression OF [ANY](symbol[, . . .]))
```

The *column\_with\_expression* cannot contain the operator AND or OR, and all its columns must come from the same input. If the function has multiple inputs, *column\_with\_expression* and *symbol* must come from the same input.

The *comparison\_operator* is either <, >, <=, >=, =, or <>.

## Result

Defines the output columns. The *col\_expr* is an expression whose value is a column name; it specifies the values to retrieve from the matched rows. The function applies *aggregate\_function* to these values. For details, see [nPath Results](#).

The function evaluates this syntax element once for every matched pattern in the partition (that is, it outputs one row for each pattern match).

## nPath Input

The function requires at least one partitioned input table, and can have additional input tables that are either partitioned or DIMENSION tables.

---

### Note:

If an additional input table has CLOB columns, it cannot be a DIMENSION table. See [nPath Results](#).

---

### Note:

If the input to nPath is nondeterministic, the results are nondeterministic.

---

## Input Table Schema

Column	Data Type	Description
<i>partition_column</i>	INTEGER or VARCHAR	Column by which every partitioned input table is partitioned.
<i>order_column</i>	INTEGER or VARCHAR	Column by which every input table is ordered.
<i>input_column</i>	INTEGER or VARCHAR	Contains data to search for patterns.

## nPath Output

The Result syntax element determines the output—see [nPath Results](#).

## nPath Symbols

A *symbol* identifies a row in the Pattern and Result syntax elements. A symbol can be any valid identifier (that is, a sequence of characters and digits that begins with a character) but is typically one or two uppercase letters. Symbols are case-insensitive; that is, 'SU' is identical to 'su', and the system reports an error if you use both.

For example, suppose that you have this input table:

record	city	temp	rh	cloudcover	windspeed	winddirection	rained_next_day
1	?	81	30	0.0	5	NW	1
2	Tempe	76	40	0.2	15	NE	0
3	?	70	70	0.4	10	N	0

record	city	temp	rh	cloudcover	windspeed	winddirection	rained_next_day
4	Tusayan	75	50	0.4	5	NW	0

This table has examples of symbol definitions and the rows of the table that they match in NONOVERLAPPING mode:

Symbol Definition	Rows Matched						
temp >= 80 AS H	1						
winddirection = 'NW' AS NW	1, 4						
winddirection = 'NW' OR windspeed > 12 AS W	1, 2, 4						
cloudcover <> 0.0 AND rh > 35 AS C	2, 3, 4						
TRUE AS A	1, 2, 3, 4 This symbol definition matches all rows, for any input table.						
city like 'tu%' AS TU	The like operator depends on Teradata Session mode: <table border="1"> <thead> <tr> <th>Mode</th> <th>Match</th> </tr> </thead> <tbody> <tr> <td>BTET</td> <td>1, 3, 4</td> </tr> <tr> <td>ANSI</td> <td>None</td> </tr> </tbody> </table>	Mode	Match	BTET	1, 3, 4	ANSI	None
Mode	Match						
BTET	1, 3, 4						
ANSI	None						
city not like 'tu%' AS TU	2						
city not like 'Tu%' AS N	2						
city like 'Tu%n' as T	1, 3, 4 The % operator matches any number of characters.						
city like 'Tu___n' as T	1, 3 The underscore (_) operator matches any single character. The pattern 'Tu___n' has three underscores, so it matches 'Tucson' but not 'Tusayan'.						

Rows with NULL values do not match any symbol. That is, the function ignores rows with missing values.

## LAG and LEAD Expressions in Symbol Predicates

You can create symbol predicates that compare a row to a previous or subsequent row, using a LAG or LEAD operator. See the following topics:

- [LAG Expression Syntax](#)
- [LAG and LEAD Expression Rules](#)

- [LAG and LEAD Expressions Example: Alias for Input Query](#)
- [LAG and LEAD Expressions Example: No Alias for Input Query](#)

## LAG Expression Syntax

```
{ current_expr operator LAG (previous_expr, lag_rows [, default]) |
  LAG (previous_expr, lag_rows [, default]) operator current_expr }
```

where:

- *current\_expr* is the name of a column from the current row (or an expression operating on this column).
- *operator* is either `>`, `>=`, `<`, `<=`, `=`, or `<>`
- *previous\_expr* is the name of a column from a previous row (or an expression operating on this column).
- *lag\_rows* is the number of rows to count backward from the current row to reach the previous row. For example, if *lag\_rows* is 1, the previous row is the immediately preceding row.
- *default* is the value to use for *previous\_expr* when there is no previous row (that is, when the current row is the first row or there is no row that is *lag\_rows* before the current row).

## LAG and LEAD Expression Rules

- A symbol definition can have multiple LAG and LEAD expressions.
- A symbol definition that has a LAG or LEAD expression cannot have an OR operator.
- If a symbol definition has a LAG or LEAD expression and the input is not a table, you must create an alias of the input query, as in [LAG and LEAD Expressions Example: Alias for Input Query](#).

## LAG and LEAD Expressions Example: Alias for Input Query

### Input

`bank_web_clicks`

customer_id	session_id	page	datestamp
529	0	ACCOUNT SUMMARY	2004-03-17 16:35:00
529	0	FAQ	2004-03-17 16:38:00
529	0	ACCOUNT HISTORY	2004-03-17 16:42:00
529	0	FUNDS TRANSFER	2004-03-17 16:45:00
529	0	ONLINE STATEMENT ENROLLMENT	2004-03-17 16:49:00
529	0	PROFILE UPDATE	2004-03-17 16:50:00

customer_id	session_id	page	datestamp
529	0	ACCOUNT SUMMARY	2004-03-17 16:51:00
529	0	CUSTOMER SUPPORT	2004-03-17 16:53:00
529	0	VIEW DEPOSIT DETAILS	2004-03-17 16:57:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:16:00
529	1	ACCOUNT SUMMARY	2004-03-18 01:18:00
529	1	FAQ	2004-03-18 01:20:00
...	...	...	...

## SQL Call

```

SELECT * FROM nPath (
    ON (SELECT customer_id, session_id, datestamp, page FROM bank_web_clicks)
AS dt1
    PARTITION BY customer_id, session_id
    ORDER BY datestamp
    USING
    Mode (NONOVERLAPPING)
    Pattern ('(DUP|A)*')
    Symbols (
        TRUE AS A,
        page = LAG (page,1) AS DUP
    )
    Result (
        FIRST (customer_id OF any (A)) AS customer_id,
        FIRST (session_id OF A) AS session_id,
        FIRST (datestamp OF A) AS first_date,
        LAST (datestamp OF ANY(A,DUP)) AS last_date,
        ACCUMULATE (page OF A) AS page_path,
        ACCUMULATE (page of DUP) AS dup_path
    )
) AS dt2;

```

## Output

Columns 1-4

customer_id	session_id	first_date	last_date
529	0	2004-03-17 16:35:00	2004-03-17 16:57:00
529	1	2004-03-18 01:16:00	2004-03-18 01:28:00

customer_id	session_id	first_date	last_date
529	2	2004-03-18 09:22:00	2004-03-18 09:36:00
529	3	2004-03-18 22:41:00	2004-03-18 22:55:00
529	4	2004-03-19 08:33:00	2004-03-19 08:41:00
529	5	2004-03-19 10:06:00	2004-03-19 10:14:00
...	...	...	...

**Columns 5-6**

page_path	dup_path
[ACCOUNT SUMMARY, FAQ, ACCOUNT HISTORY, FUNDS TRANSFER, ONLINE STATEMENT ENROLLMENT, PROFILE UPDATE, ACCOUNT SUMMARY, CUSTOMER SUPPORT, VIEW DEPOSIT DETAILS]	[]
[ACCOUNT SUMMARY, FAQ, ACCOUNT SUMMARY, FUNDS TRANSFER, ACCOUNT HISTORY, VIEW DEPOSIT DETAILS, ACCOUNT SUMMARY, ACCOUNT HISTORY]	[ACCOUNT SUMMARY]
[ACCOUNT SUMMARY, ACCOUNT HISTORY, FUNDS TRANSFER, ACCOUNT SUMMARY, FAQ]	[ACCOUNT SUMMARY, ACCOUNT SUMMARY, FAQ]
[ACCOUNT SUMMARY, ACCOUNT HISTORY, ACCOUNT SUMMARY, ACCOUNT HISTORY, FAQ, ACCOUNT SUMMARY]	[ACCOUNT SUMMARY]
[ACCOUNT SUMMARY, FAQ, VIEW DEPOSIT DETAILS, FAQ]	[]
[ACCOUNT SUMMARY, FUNDS TRANSFER, VIEW DEPOSIT DETAILS, ACCOUNT HISTORY]	[VIEW DEPOSIT DETAILS]
...	...

**LAG and LEAD Expressions Example: No Alias for Input Query****Input**

aggregate_clicks						
userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	sneakers	home	2009-07-29 20:17:59	Company1	100
1039	2	books	home	2009-04-21 13:17:59	Company4	300
1039	3	television	home	2009-05-23 13:17:59	Company2	500
1039	4	envelopes	home	2009-07-16 11:17:59	Company3	10
1039	4	envelopes	home1	2009-07-16 11:18:16	Company3	10

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	4	envelopes	page1	2009-07-16 11:18:18	Company3	10
1039	5	bookcases	home	2009-08-19 22:17:59	Company5	150
1039	5	bookcases	home1	2009-08-19 22:18:02	Company5	150
1039	5	bookcases	page1	2009-08-19 22:18:05	Company5	150
1039	5	bookcases	page2	2009-08-22 04:20:05	Company5	150
1039	5	bookcases	checkout	2009-08-24 14:30:05	Company5	150
1039	5	bookcases	page2	2009-08-27 23:03:05	Company5	150
1040	1	tables	home	2009-07-29 20:17:59	Company5	250
1040	2	Appliances	home	2009-04-21 13:17:59	Company6	1500
1040	3	laptops	home	2009-05-23 13:17:59	Company7	800
1040	4	chairs	home	2009-07-16 11:17:59	Company3	400
1040	4	chairs	home1	2009-07-16 11:18:16	Company3	400
1040	4	chairs	page1	2009-07-16 11:18:18	Company3	400
1040	5	cellphones	home	2009-08-19 22:17:59	Company8	600
1040	5	cellphones	home1	2009-08-19 22:18:02	Company8	600
1040	5	cellphones	page1	2009-08-19 22:18:05	Company8	600
1040	5	cellphones	page2	2009-08-22 04:20:05	Company8	600
1040	5	cellphones	checkout	2009-08-24 14:30:05	Company8	600
1040	5	cellphones	page2	2009-08-27 23:03:05	Company8	600
...	...	...	...	...	...	...

## SQL Call

```

SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime ASC
  USING
  Mode (NONOVERLAPPING)
  Pattern ('H+.D*.X*.P1.P2+')
  Symbols (
    TRUE AS X,
    pagetype = 'home' AS H,
    pagetype <> 'home' AND pagetype <> 'checkout' AS D,
    pagetype = 'checkout' AS P1,
    pagetype = 'checkout' AND
    productprice > 100 AND
  )
)
```

```

productprice > LAG (productprice, 1, 100) AS P2
)
Result (
  FIRST (productname OF P1) AS first_product,
  MAX_CHOOSE (productprice, productname OF P2) AS max_product,
  FIRST (sessionid OF P2) AS sessionid
)
) AS dt ORDER BY sessionid;

```

## Output

first_product	max_product	sessionid
bookcases	cellphones	5

## nPath Patterns

The value of the Pattern syntax element specifies the sequence of rows for which the function searches. You compose the pattern definition, *pattern*, with symbols (which you define in the Symbols syntax element), operators, and parentheses. In the pattern definition, symbols represent rows. You can combine symbols with pattern operators to define simple or complex patterns of rows for which to search.

The following table lists and describes the basic pattern operators, in decreasing order of precedence. In the table, A and B are symbols that have been defined in the Symbols syntax element.

### Basic Pattern Operators

Operator	Description	Precedence
A	Matches one row that meets the definition of A.	1 (highest)
A.	Matches one row that meets the definition of A.	1
A?	Matches 0 or 1 rows that satisfy the definition of A.	1
A*	Matches 0 or more rows that satisfy the definition of A (greedy operator).	1
A+	Matches 1 or more rows that satisfy the definition of A (greedy operator).	1
A.B	Matches two rows, where the first row meets the definition of A and the second row meets the definition of B.	2
A B	Matches one row that meets the definition of either A or B.	3

The nPath function uses greedy pattern matching. That is, it finds the longest available match when matching patterns specified by nongreedy operators. For more information, see [nPath Greedy Pattern Matching](#).

These examples show the pattern operator precedence rules:

- A.B+ is the same as A.(B+)
- A|B\* is the same as A|(B\*)
- A.B|C is the same as (A.B)|C

Example:

```
A.(B|C)+.D?.X*.A
```

The preceding pattern definition matches any set of rows whose first row meets the definition of symbol A, followed by a nonempty sequence of rows, each of which meets the definition of either symbol B or C, optionally followed by one row that meets the definition of symbol D, followed by any number of rows that meet the definition of symbol X, and ending with a row that meets the definition of symbol A.

You can use parentheses to define precedence rules. Parentheses are recommended for clarity, even where not strictly required.

To indicate that a sequence of rows must start or end with a row that matches a certain symbol, use the start anchor (^) or end anchor (\$) operator.

#### Start Anchor and End Anchor Pattern Operators

Operator	Description
<code>^A</code>	Appears only at the beginning of a pattern. Indicates that a set of rows must start with a row that meets the definition of A.
<code>A\$</code>	Appears only at the end of a pattern. Indicates that a set of rows must end with a row that meets the definition of A.

Subpattern operators let you specify how often a subpattern must appear in a match. You can specify a minimum number, exact number, or range. In the following table, X represents any pattern definition composed of symbols and any of the previously described pattern operators.

#### Subpattern Operators

Operator	Description
<code>(X){a}</code>	Matches exactly a occurrences of the pattern X.
<code>(X){a,}</code>	Matches at least a occurrences of the pattern X.
<code>(X){a,b}</code>	Matches at least a and no more than b occurrences of the pattern X.

## nPath Greedy Pattern Matching

The nPath function uses greedy pattern matching, finding the longest available match despite any nongreedy operators in the pattern.

For example, consider the input table link2:

**nPath Greedy Pattern Matching Examples Input Table link2**

userid	job_title	startdate	enddate
21	Chief Exec Officer	1994-10-01	2005-02-28
21	Software Engineer	1996-10-01	2001-06-30
21	Software Engineer	1998-10-01	2001-06-30
21	Chief Exec Officer	2005-03-01	2007-03-31
21	Chief Exec Officer	2007-06-01	?

The following query returns the following table:

```
SELECT job_transition_path, count(*) AS path_count FROM nPath (
    ON link2 PARTITION BY userid ORDER BY startdate
    USING
    Mode (NONOVERLAPPING)
    Pattern ('CEO.ENGR.OTHER*')
    Symbols (
        job_title like '%Software Eng%' AS ENGR,
        TRUE AS OTHER,
        job_title like 'Chief Exec Officer' AS CEO
    )
    Result (accumulate(job_title OF ANY(ENGR,OTHER,CEO)) AS job_transition_path)
) AS dt GROUP BY 1 ORDER BY 2 DESC;
```

job_transition_path	path_count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, CEO matches the first row, ENGR matches the second row, and OTHER\* matches the remaining rows:

title	pattern ('CEO.ENGR.OTHER*')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

The following query returns the following table:

```
SELECT job_transition_path , count(*) AS path_count FROM nPath (
  ON link2 PARTITION BY userid ORDER BY startdate
  USING
  Mode (NONOVERLAPPING)
  Pattern ('CEO.ENGR.OTHER*.CEO')
  Symbols (
    job_title like '%Software Eng%' AS ENGR,
    TRUE AS OTHER,
    job_title like 'Chief Exec Officer' AS CEO
  )
  Result (accumulate(job_title OF ANY(ENGR,OTHER,CEO)) AS job_transition_path)
) AS dt GROUP BY 1 ORDER BY 2 DESC;
```

job_transition_path	path_count
[Chief Exec Officer, Software Engineer, Software Engineer, Chief Exec Officer, Chief Exec Officer]	1

In the pattern, CEO matches the first row, ENGR matches the second row, OTHER\* matches the next two rows, and CEO matches the last row:

title	pattern ('CEO.ENGR.OTHER*.CEO')
Chief Exec Officer	
Software Engineer	
Software Engineer	
Chief Exec Officer	
Chief Exec Officer	

## nPath Filters

The Filter syntax element specifies filters to impose on the matched rows. [nPath Filters Example](#) shows an example of an nPath call that uses a filter.

## nPath Filters Example

Using clickstream data from an online store, this example finds the sessions where the user visited the checkout page within 10 minutes of visiting the home page. Because there is no way to know in advance how many rows might appear between the home page and the checkout page, the example cannot use a LAG or LEAD expression. Therefore, it uses the Filter syntax element.

### Input

**clickstream**

userid	sessionid	clicktime	pagetype
1	1	10-10-2012 10:15	home
1	1	10-10-2012 10:16	view
1	1	10-10-2012 10:17	view
1	1	10-10-2012 10:20	checkout
1	1	10-10-2012 10:30	checkout
1	1	10-10-2012 10:35	view
1	1	10-10-2012 10:45	view
2	2	10-10-2012 13:15	home
2	2	10-10-2012 13:16	view
2	2	10-10-2012 13:43	checkout
2	2	10-10-2012 13:35	view
2	2	10-10-2012 13:45	view

### SQL Call

```

SELECT * FROM Npath (
    ON clickstream PARTITION BY userid ORDER BY clicktime
    USING
    Symbols (
        pagetype='home' AS home,
        pagetype <> 'home' AND pagetype <> 'checkout' AS clickview,
        pagetype='checkout' AS checkout
    )
    Pattern ('home.clickview*.checkout')
    Result (
        FIRST(userid of ANY(home, checkout, clickview)) AS userid,

```

```

        FIRST (sessionid of ANY(home, checkout, clickview)) AS sessioinid,
        COUNT (* of any(home, checkout, clickview)) AS cnt,
        FIRST (clicktime of ANY(home)) AS firsthome,
        LAST (clicktime of ANY(checkout)) AS lastcheckout
    )
    Filter (
        FIRST (clicktime + interval '10' minute OF ANY (home)) >
        FIRST (clicktime of any(checkout))
    )
    Mode (NONOVERLAPPING)
);

```

## Output

userid	sessionid	cnt	firsthome	lastcheckout
1	1	4	2012-10-10 10:15:00	2012-10-10 10:20:00

## nPath Results

The Result syntax element defines the output columns, specifying the values to retrieve from the matched rows and the aggregate function to apply to these values.

For each pattern, the nPath function can apply one or more aggregate functions to the matched rows and output the aggregated results. These are the supported aggregate functions: AVG, COUNT, MAX, MIN, and SUM, described in *Teradata Vantage™ - SQL Functions, Expressions, and Predicates*, B035-1145

In the following table, *col\_expr* is an expression whose value is a column name, *symbol* is defined by the Symbols syntax element, and *symbol\_list* has this syntax:

```
{ symbol | ANY (symbol[,...]) }
```

Function	Description
COUNT (         { *   [DISTINCT] col_         expr }         OF symbol_list )	Returns either the number of total number of matched rows (*) or the number (or distinct number) of <i>col_expr</i> values in the matched rows.
FIRST (         col_expr OF symbol_list )	Returns the <i>col_expr</i> value of the first matched row.
LAST (         col_expr OF symbol_list )	Returns the <i>col_expr</i> value of the last matched row.

Function	Description
<code>NTH ( col_expr, n OF symbol_list )</code>	Returns the <code>col_expr</code> value of the <code>n</code> th matched row, where <code>n</code> is a nonzero value of the data type SMALLINT, INTEGER, or BIGINT. The sign of <code>n</code> determines whether the <code>n</code> th matched row is <code>n</code> th from the first or last matched row. For example, if <code>n</code> is 1, the <code>n</code> th matched row is the first matched row, and if <code>n</code> is -1, the <code>n</code> th matched row is the last matched row. If <code>n</code> is greater than the number of matched rows, the <code>n</code> th function returns NULL.
<code>FIRST_NOTNULL ( col_expr OF symbol_list )</code>	Returns the first non-null <code>col_expr</code> value in the matched rows.
<code>LAST_NOTNULL ( col_expr OF symbol_list )</code>	Returns the last non-null <code>col_expr</code> value in the matched rows.
<code>MAX_CHOOSE ( quantifying_col_expr, descriptive_col_expr OF symbol_list )</code>	Returns the <code>descriptive_col_expr</code> value of the matched row with the highest-sorted <code>quantifying_col_expr</code> value. For example, <code>MAX_CHOOSE (product_price, product_name OF B)</code> returns the <code>product_name</code> of the most expensive product in the rows that map to B. The <code>descriptive_col_expr</code> can have any data type. The <code>quantifying_col_expr</code> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).
<code>MIN_CHOOSE ( quantifying_col_expr, descriptive_col_expr OF symbol_list )</code>	Returns the <code>descriptive_col_expr</code> value of the matched row with the lowest-sorted <code>quantifying_col_expr</code> value. For example, <code>MIN_CHOOSE (product_price, product_name OF B)</code> returns the <code>product_name</code> of the least expensive product in the rows that map to B. The <code>descriptive_col_expr</code> can have any data type. The <code>quantifying_col_expr</code> must have a sortable datatype (SMALLINT, INTEGER, BIGINT, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, VARCHAR, or CHARACTER).
<code>DUPCOUNT ( col_expr OF symbol_list )</code>	Returns the duplicate count for <code>col_expr</code> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <code>col_expr</code> in the immediately preceding matched row. When <code>col_expr</code> is also the ORDER BY <code>col_expr</code> , this function returns the equivalent of <code>ROW_NUMBER() - RANK()</code> .
<code>DUPCOUNTCUM ( col_expr OF symbol_list )</code>	Returns the cumulative duplicate count for <code>col_expr</code> in the matched rows. That is, for each matched row, the function returns the number of occurrences of the current value of <code>col_expr</code> in all preceding matched rows. When <code>col_expr</code> is also the ORDER BY <code>col_expr</code> , this function returns the equivalent of <code>ROW_NUMBER() - DENSE_RANK()</code> .
<code>ACCUMULATE [ (size, truncateFlag) ]</code>	Returns, for each matched row, the concatenated values in <code>col_expr</code> , separated by <code>delimiter</code> .

Function	Description
<pre>( [ DISTINCT   CDISTINCT ]   col_expr OF symbol_list   [ DELIMITER 'delimiter' ] ) [ AS result_col_name ]</pre>	<p><i>size</i> is the maximum size of the accumulated string. The default value of <i>size</i> is 64000 and the maximum is 2097088000. If <i>size</i> is greater than 64000, the accumulated string has data type CLOB. Otherwise, it has data type VARCHAR.</p> <p><i>truncateFlag</i> specifies whether to truncate the accumulated string if it is longer than <i>size</i>. <i>truncateFlag</i> is TRUE or FALSE (not enclosed in single quotation marks). Its default value is FALSE.</p> <p><i>delimiter</i> is a string of LATIN characters. Its default value is ',' (a comma followed by a space). Its maximum size is 100 bytes.</p> <p>DISTINCT limits the concatenated values to distinct values.</p> <p>CDISTINCT limits the concatenated values to consecutive distinct values.</p> <p>The accumulated string can have at most 2097088000 LATIN characters.</p> <p>ACCUMULATE does not support UNICODE characters in the input data.</p>

You can compute an aggregate over more than one symbol. For example, SUM (val OF ANY (A,B)) computes the sum of the values of the attribute *val* across all rows in the matched segment that map to A or B.

See [nPath Results Examples](#) for a list of examples.

## nPath Results Examples

- [nPath Results Example: FIRST, LAST\\_NOTNULL, MAX\\_CHOOSE, and MIN\\_CHOOSE](#)
- [nPath Results Example: FIRST and Three Forms of ACCUMULATE](#)
- [nPath Results Example: FIRST, Three Forms of ACCUMULATE, COUNT, and NTH](#)
- [nPath Results Example: Combine Values from One Row with Values from the Next Row](#)
- [nPath Results Example: ACCUMULATE String over 64000 Characters](#)
- [nPath Results Example: ACCUMULATE String over 64000 Characters, DISTINCT](#)
- [nPath Results Example: ACCUMULATE String over 64000 Characters, CDISTINCT](#)
- [nPath Results Example: Hindi Input, ACCUMULATE FIRST\\_NOTNULL](#)
- [nPath Results Example: Hindi Input, ACCUMULATE DISTINCT and CDISTINCT](#)

### nPath Results Example: FIRST, LAST\_NOTNULL, MAX\_CHOOSE, and MIN\_CHOOSE

#### Input

trans1

userid	gender	ts	productname	productamt
1	M	2012-01-01 00:00:00	shoes	100

userid	gender	ts	productname	productamt
1	M	2012-02-01 00:00:00	books	300
1	M	2012-03-01 00:00:00	television	500
1	M	2012-04-01 00:00:00	envelopes	10
2		2012-01-01 00:00:00	bookcases	150
2		2012-02-01 00:00:00	tables	250
2	F	2012-03-01 00:00:00	appliances	1500
3	F	2012-01-01 00:00:00	chairs	400
3	F	2012-02-01 00:00:00	cellphones	600
3	F	2012-03-01 00:00:00	dvds	50

## SQL Call

```
SELECT * FROM nPath (
    ON trans1 PARTITION BY userid ORDER BY ts
    USING
    Mode (NONOVERLAPPING)
    Pattern ('A+')
    Symbols (TRUE AS A)
    Result (
        FIRST (userid OF A) AS Userid,
        LAST_NOTNULL (gender OF A) AS Gender,
        MAX_CHOOSE (productamt, productname OF A) AS Max_prod,
        MIN_CHOOSE (productamt, productname OF A) AS Min_prod
    )
) ORDER BY 1;
```

## Output

userid	gender	max_prod	min_prod
1	M	television	envelopes
2	F	appliances	bookcases
3	F	cellphones	Dvds

## nPath Results Example: FIRST and Three Forms of ACCUMULATE

### Input

clicks						
userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	?	home	06:59:13	Company1	100
1039	1	?	home	07:00:10	Company2	300
1039	1	television	checkout	07:00:12	Company2	500
1039	1	television	checkout	07:00:18	Company2	10
1039	1	envelopes	checkout	07:01:00	Company3	10
1039	1	?	checkout	07:01:10	Company3	10

### SQL Call

```

SELECT * FROM nPath (
    ON clicks PARTITION BY sessionid ORDER BY clicktime
    USING
        Mode (NONOVERLAPPING)
        Symbols (
            pagetype='home' AS H,
            pagetype='checkout' AS C,
            pagetype <> 'home' AND pagetype <>'checkout' AS A
        )
        Pattern ('^H+.A*.C+$')
        Result (
            FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
            FIRST (clicktime OF H) AS firsthome,
            FIRST (clicktime OF C) AS firstcheckout,
            ACCUMULATE (productname OF ANY (H,A,C) DELIMITER '**')
            AS products_accumulate,
            ACCUMULATE (CDISTINCT productname OF ANY (H,A,C) DELIMITER '$$')
            AS cde_dup_products,
            ACCUMULATE (DISTINCT productname OF ANY (H,A,C)) AS de_dup_products
        )
    ) ORDER BY sessionid;

```

## Output

sessionid	firsthome	firstcheckout	products_accumulate	cde_dup_products	de_dup_products
1	06:59:13	07:00:12	[null>null*television*television*envelopes>null]	[null\$\$television\$ \$envelopes\$\$null]	[null, television, envelopes]

## nPath Results Example: FIRST, Three Forms of ACCUMULATE, COUNT, and NTH

### Input

The input table for this example is clicks, as in [nPath Results Example: FIRST and Three Forms of ACCUMULATE](#).

### SQL Call

```
SELECT * FROM nPath (
    ON clicks PARTITION BY sessionid ORDER BY clicktime
    USING
        Mode (NONOVERLAPPING)
        Symbols (
            pagetype='home' AS H,
            pagetype='checkout' AS C,
            pagetype <> 'home' AND pagetype <>'checkout' AS A
        )
        Pattern ('^H+.A*.C+$')
        Result (
            FIRST (sessionid OF ANY (H, A, C)) AS sessionid,
            FIRST (clicktime OF H) AS firsthome,
            FIRST (clicktime OF C) AS firstcheckout,
            ACCUMULATE (productname OF ANY (H,A,C)) AS products_accumulate,
            COUNT (DISTINCT productname OF ANY(H,A,C)) AS count_distinct_products,
            ACCUMULATE (CDISTINCT productname OF ANY
(H,A,C)) AS consecutive_distinct_products,
            ACCUMULATE (DISTINCT productname OF ANY (H,A,C)) AS distinct_products,
            NTH (productname, -1 OF ANY(H,A,C)) AS nth
        )
    ) ORDER BY sessionid;
```

## Output

sessionid	firsthome	firstcheckout	products_accumulate	count_distinct_products	consecutive_distinct_products	distinct_products	nth
1	06:59:13	07:00:12	[null, null, television, television, envelopes, null]	2	[null, television, envelopes, null]	[null, television, envelopes]	?

## nPath Results Example: Combine Values from One Row with Values from the Next Row

### Input

The input table is clickstream, as in [nPath Filters Example](#).

### SQL Call

```
SELECT * FROM nPath (
    ON clickstream PARTITION BY userid ORDER BY userid, sessionid, clicktime
    USING
        Mode (OVERLAPPING)
        Pattern ('A.B')
        Symbols (TRUE AS A, TRUE AS B)
    Result (
        FIRST (sessionid OF A) AS sessionid,
        FIRST (pagetype OF A) AS pageid,
        FIRST (pagetype OF B) AS next_pageid
    )
) ORDER BY sessionid;
```

### Output

sessionid	pageid	next_pageid
1	home	view
1	view	view
1	checkout	view
1	checkout	checkout
1	view	checkout
1	view	view
2	checkout	view
2	home	view
2	view	view
2	view	checkout

## nPath Results Example: ACCUMULATE String over 64000 Characters

### Input

Input table banking\_web\_clicks contains information for 5000 clicks to different pages in one session for one customer.

**banking\_web\_clicks**

customer_id	session_id	page	datestamp
540	1	ONLINE STATEMENT ENROLLMENT	2022-03-18 06:14:20.008294
540	1	CUSTOMER SUPPORT	2022-03-18 06:14:21.009358
540	1	ACCOUNT HISTORY	2022-03-18 06:14:22.010443
540	1	VIEW DEPOSIT DETAILS	2022-03-18 06:14:23.011530
540	1	PROFILE UPDATE	2022-03-18 06:14:24.012609
540	1	FREQUENTLY ASKED QUESTIONS	2022-03-18 06:14:25.013711
540	1	FUNDS TRANSFER	2022-03-18 06:14:26.014798
540	1	ACCOUNT HISTORY	2022-03-18 06:14:27.015879
540	1	ONLINE STATEMENT ENROLLMENT	2022-03-18 06:14:28.016968
540	1	BANK STATEMENT	2022-03-18 06:14:29.018053
540	1	COMPLAINTS	2022-03-18 06:14:30.019146
540	1	BANK STATEMENT	2022-03-18 06:14:31.020222
540	1	ACCOUNT SUMMARY	2022-03-18 06:14:32.021308
540	1	COMPLAINTS	2022-03-18 06:14:33.022394
540	1	FUNDS TRANSFER	2022-03-18 06:14:34.023481
...	...	...	...
540	1	CUSTOMER SUPPORT	2022-03-18 07:37:44.428571

### SQL Call

```
SELECT * FROM nPath (
  ON banking_web_clicks PARTITION BY session_id ORDER BY datestamp
  USING
    Mode (NONOVERLAPPING)
    Pattern ('A*')
    Symbols (TRUE AS A)
```

```

Result (
    FIRST (session_id OF A) AS session_id,
    ACCUMULATE (100000)(page OF A) AS path
)
) AS dt;

```

## Output

Because ACCUMULATE specifies a size greater than 64000, the data type of the path column is CLOB. The length of the value of path is 91009.

session_id	path
1	[ONLINE STATEMENT ENROLLMENT, CUSTOMER SUPPORT, ACCOUNT HISTORY, VIEW DEPOSIT DETAILS, PROFILE UPDATE, FREQUENTLY ASKED QUESTIONS, FUNDS TRANSFER, ACCOUNT HISTORY, ONLINE STATEMENT ENROLLMENT, BANK STATEMENT, COMPLAINTS, BANK STATEMENT, ACCOUNT SUMMARY, COMPLAINTS, FUNDS TRANSFER, ... , HOME PAGE]

## nPath Results Example: ACCUMULATE String over 64000 Characters, DISTINCT

### Input

Input table is banking\_web\_clicks (see [nPath Results Example: ACCUMULATE String over 64000 Characters](#)).

### SQL Call

```

SELECT * FROM nPath (
    ON banking_web_clicks PARTITION BY session_id ORDER BY datestamp
    USING
    Mode (NONOVERLAPPING)
    Pattern ('A*')
    Symbols (TRUE AS A)
    Result (
        FIRST (session_id OF A) AS session_id,
        ACCUMULATE (100000)(DISTINCT page OF A) AS path
    )
) AS dt;

```

## Output

Because ACCUMULATE specifies a size greater than 64000, the data type of the path column is CLOB, even though the length of the value of path is 219.

session_id	path
1	[ONLINE STATEMENT ENROLLMENT, CUSTOMER SUPPORT, ACCOUNT HISTORY, VIEW DEPOSIT DETAILS, PROFILE UPDATE, FREQUENTLY ASKED QUESTIONS, FUNDS TRANSFER, BANK STATEMENT, COMPLAINTS, ACCOUNT SUMMARY, HOME PAGE, ADD BENEFICIARY]

## nPath Results Example: ACCUMULATE String over 64000 Characters, CDISTINCT

### Input

Input table is banking\_web\_clicks (see [nPath Results Example: ACCUMULATE String over 64000 Characters](#)).

### SQL Call

```
SELECT * FROM nPath (
    ON banking_web_clicks PARTITION BY session_id ORDER BY datetimestamp
    USING
        Mode (NONOVERLAPPING)
        Pattern ('A*')
        Symbols (TRUE AS A)
        Result (
            FIRST (session_id OF A) AS session_id,
            ACCUMULATE (100000)(CDISTINCT page OF A) AS path
        )
) AS dt;
```

### Output

Because ACCUMULATE specifies a size greater than 64000, the data type of the path column is CLOB. The length of the value of path is 82887.

session_id	path
1	[ONLINE STATEMENT ENROLLMENT, CUSTOMER SUPPORT, ACCOUNT HISTORY, VIEW DEPOSIT DETAILS, PROFILE UPDATE, FREQUENTLY ASKED QUESTIONS, FUNDS TRANSFER, ACCOUNT HISTORY, ONLINE STATEMENT ENROLLMENT, BANK STATEMENT, COMPLAINTS, BANK STATEMENT, ACCOUNT SUMMARY, COMPLAINTS, FUNDS TRANSFER, ..., BANK STATEMENT, CUSTOMER SUPPORT]

## nPath Results Example: Hindi Input, ACCUMULATE FIRST\_NOTNULL

### Input

The example has two input tables that include Hindi characters.

हिंदी टेबल

सत्रआईडी	क्लिककरें	token	उत्पादकानाम	पेजकाप्रकार	रेफरर
1	06:59:13.000000	1		घर	गूगल डॉट कॉम
13	15:35:08.000000	15		घर	गूगल डॉट कॉम
400	10:00:00.000000	300		घर	
9000	05:30:15.000000	?		घर	
1	07:00:10.000000	11		घर	गूगल डॉट कॉम
9001	05:30:15.000000	?		लॉग इन	
400	10:05:04.000000	12	आकाशगंगा s4	चेक आउट	
9000	05:30:20.000000	?	लेनोवो g580	चेक आउट	
1	07:00:12.000000	111	iPod	चेक आउट	गूगल डॉट कॉम
9001	05:30:15.000000	?		घर	
400	10:05:03.000000	12		कागज एक	
9000	05:50:44.000000	?	लैपटॉप case	चेक आउट	
1	07:01:00.000000	1111	बोस	चेक आउट	
9001	05:30:20.000000	?	prod4	चेक आउट	
400	09:59:55.000000	12		लॉग इन	
9000	12:50:55.000000	?		लॉग आउट	
1	18:00:00.000000	10		लॉग इन	
9001	12:50:55.000000	?		लॉग आउट	
400	10:05:55.000000	18		लॉग आउट	
14	13:18:30.000000	2		घर	गूगल डॉट कॉम
1	18:00:10.000000	10		घर	
8000	16:00:00.000000	8001		लॉग इन	
400	10:05:02.000000	18		घर	
14	13:18:31.000000	8		कागज एक	

सत्रआईडी	क्लिकरें	token	उत्पादकानाम	पेजकाप्रकार	रेफर
1	18:00:15.000000	10	आई - फोन USB cable	चेक आउट	
8000	16:00:10.000000	8002		घर	
400	18:00:10.000000	18		घर	
14	13:18:32.000000	8		page2	
666	12:50:15.000000	40		घर	
8000	16:00:20.000000	8003	nexus7	चेक आउट	
400	18:05:10.000000	18	prod1	चेक आउट	
14	13:18:40.000000	20	आई - फोन	चेक आउट	
666	12:50:20.000000	41	लेनोवो g580	चेक आउट	
8000	16:00:40.000000	8004		लॉग आउट	
400	18:08:10.000000	100	prod2	चेक आउट	
14	13:19:00.000000	20	बोस	चेक आउट	
666	12:50:44.000000	42	लैपटॉप case	चेक आउट	
400	18:10:10.000000	150		लॉग आउट	
14	13:20:00.000000	20	सैमसंग	चेक आउट	
666	12:50:55.000000	50		लॉग आउट	
400	09:59:45.000000	150		लॉग इन	
500	08:15:12.000000	12		लॉग इन	
10000	16:00:00.000000	1		लॉग इन	
400	09:59:40.000000	210		लॉग इन	
500	08:15:15.000000	31		घर	
10000	16:00:10.000000	1		घर	
400	09:59:55.000000	220		लॉग इन	
500	08:15:20.000000	123		कागज एक	
10000	16:00:20.000000	2	nexus7	चेक आउट	
400	09:59:55.000000	220		लॉग इन	
500	08:16:00.000000	1231	आकाशगंगा चार्जर	चेक आउट	
10000	16:00:40.000000	4		लॉग आउट	
500	08:16:30.000000	1232	हेडफोन	चेक आउट	

सत्रआईडी	क्लिकरें	token	उत्पादकानाम	पेजकाप्रकार	रेफरर
2	15:34:25.000000	333		घर	गूगल डॉट कॉम
500	08:12:12.000000	1233		लॉग इन	
2	15:34:25.000000	333		लॉग आउट	
250	20:00:01.000000	8		घर	गो डैडी डॉट कॉम
250	20:02:00.000000	80	बोस	चेक आउट	
250	20:02:50.000000	81	itrip	चेक आउट	
250	20:03:00.000000	82	आई - फोन	चेक आउट	

**विज्ञापन**

रिलेसमय	channel	विज्ञापन	duration
20:02:01.000000	सीएनबीसी	13	1000
15:35:06.000000	सीएनबीसी	14	1000
15:34:26.000000	food network	11	1000
13:18:42.000000	espn	12	1000
07:00:20.000000	सीएनबीसी	10	1000

**SQL Call**

```

SELECT * FROM nPath (
    ON "हिंदी टेबल" PARTITION BY "सत्र आईडी" ORDER BY "क्लिक करें"
    ON "विज्ञापन" DIMENSION ORDER BY "रिले समय"
    USING
        Mode (NONOVERLAPPING)
        Pattern ('^(X|A)+.C')
        Symbols ( "पेज का प्रकार" IS NULL OR "पेज का प्रकार" IS NOT NULL AS X,
                  "पेज का प्रकार" = 'चेक आउट' AS C,
                  "विज्ञापन" IS NULL OR "विज्ञापन" IS NOT NULL AS A
        )
        Result (
            ACCUMULATE ("रेफरर" of ANY(X)) AS "रेफरल पथ" ,
            FIRST_NONNULL ("सत्र आईडी" of ANY(X)) AS "सत्र आईडी"
        )
) AS dt;

```

**Output**

रेफरल पथ	सत्रआईडी
[गूगल डॉट कॉम, याहू डॉट कॉम, याहू डॉट कॉम, , , ]	1
[, ]	8000
[गूगल डॉट कॉम, , , , ]	14
[गो डैडी डॉट कॉम, , ]	250
[, , , , , , , , ]	400
[, , , , ]	500
[, ]	666
[, ]	9001
[, ]	9000
[, ]	10000

**nPath Results Example: Hindi Input, ACCUMULATE DISTINCT and CDISTINCT****Input****unicode\_path**

id	price	event
2	2.20000000000000E 000	शब्द1
2	5.13300000000000E 001	शब्द2
2	9.88123000000000E 002	शब्द3
2	-1.20000000000000E-001	शब्द4
2	1.10000000000000E 000	शब्द5
2	1.12000000000000E 001	शब्द5
2	1.22000000000000E 001	शब्द5
2	1.20000000000000E 000	शब्द1

**SQL Call**

```
SELECT * FROM nPath (
  ON unicode_path PARTITION BY id
  USING
```

```

Mode (NONOVERLAPPING)
Pattern ('A*')
Symbols (true AS A)
Result (
    ACCUMULATE (DISTINCT event OF A DELIMITER ',') AS acc_result_distinct,
    ACCUMULATE (CDISTINCT event OF A DELIMITER ',') AS acc_result_cdistinct
)
) AS dt;

```

## Output

acc_result_distinct	acc_result_cdistinct
[ఈవట3, ఈవట5, ఈవట1, ఈవట4, ఈవట2]	[ఈవట3, ఈవట5, ఈవట1, ఈవట4, ఈవట2, ఈవట1]

## nPath Examples

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

- [Symbols and Symbol Predicates That Examples Use](#)
- [nPath ClickStream Data Examples](#)
- [nPath Range-Matching Examples](#)

## Symbols and Symbol Predicates That Examples Use

Symbol	Symbol Predicate
A	pageid IN (10, 25)
B	category = 10 OR (category = 20 AND pageid <> 33)
C	category IN (SELECT pageid FROM clicks1 GROUP BY userid HAVING COUNT(*) > 10)
D	referrer LIKE '%Amazon%'
X	TRUE

## nPath ClickStream Data Examples

### Input

This statement creates the input table of clickstream data that the examples use:

```
CREATE MULTISET TABLE clicks1 (
    userid INTEGER,
    sessionid INTEGER,
    pageid INTEGER,
    category INTEGER,
    ts TIMESTAMP FORMAT 'YYYY-MM-DDHH:MI:SS',
    referrer VARCHAR (256),
    val FLOAT
) PRIMARY INDEX ( userid );
```

This statement gets the pageid for each row and the pageid for the next row in sequence:

```
SELECT dt.sessionid, dt.pageid, dt.next_pageid FROM nPath (
    ON clicks1 PARTITION BY sessionid ORDER BY ts
    USING
    Mode (OVERLAPPING)
    Pattern ('A.B')
    Symbols (TRUE AS A, TRUE AS B)
    Result (
        FIRST(sessionid OF A) AS sessionid,
        FIRST (pageid OF A) AS pageid,
        FIRST (pageid OF B) AS next_pageid
    )
) AS dt;
```

### **Example: Counting Preceding Rows in a Sequence**

For each row, this invocation counts the number of preceding rows in a given sequence (including the current row). The ORDER BY clause specifies DESC because the pattern must be matched over the rows preceding the start row, while the semantics dictate that the pattern be matched over the rows following the start row.

```
SELECT dt.sessionid, dt.pageid, dt.countrank FROM nPath (
    ON clicks1 PARTITION BY sessionid ORDER BY ts DESC
    USING
    Mode (OVERLAPPING)
    Pattern ('A*')
    Symbols (TRUE AS A)
    Result (
        FIRST (sessionid OF A) AS sessionid,
        FIRST (pageid OF A) AS pageid,
        COUNT (* OF A) AS countrank
    )
) AS dt;
```

## Example: Complex Path Query

This query finds the user click-paths that start at pageid 50 and proceed either to pageid 80 or to pages in category 9 or category 10, finds the pageid of the last page in the path, counts the visits to page 80, and returns the maximum count for each last page, by which it sorts the output. The query ignores paths of fewer than five pages and pages for which category is less than zero.

```

SELECT dt.last_pageid, MAX(dt.count_page80) FROM nPath (
    ON (SELECT * FROM clicks1 WHERE category >= 0)
    PARTITION BY sessionid ORDER BY ts
    USING
        Pattern ('A.(B|C)*')
        Mode (OVERLAPPING)
        Symbols (
            pageid = 50 AS A,
            pageid = 80 AS B,
            pageid <> 80 AND category IN (9,10) AS C
        )
    Result (
        LAST(pageid OF ANY (A,B,C)) AS last_pageid,
        COUNT (* OF B) AS count_page80,
        COUNT (* OF ANY (A,B,C)) AS count_any
    )
) AS dt WHERE dt.count_any >= 5
GROUP BY dt.last_pageid
ORDER BY MAX(dt.count_page80);

```

## nPath Range-Matching Examples

The following examples use the information presented in this topic.

- [nPath Range-Matching Example: Accumulate Pages Visited in Each Session](#)
- [nPath Range-Matching Example: Find Sessions That Start at Home Page and Visit Page1](#)
- [nPath Range-Matching Example: Find Paths to Checkout Page for Purchases Over \\$200](#)
- [nPath Range-Matching Example: Use OVERLAPPING Mode](#)
- [nPath Range-Matching Example: Find First Product with Multiple Referrers in Any Session](#)
- [nPath Range-Matching Example: Find Data for Sessions That Checked Out 3-6 Products](#)
- [nPath Range-Matching Example: Find Data for Sessions That Checked Out at Least 3 Products](#)
- [nPath Range-Matching Example: Multiple Partitioned Input Tables and Dimension Input Table](#)

Whenever a user visits the home page and then visits checkout pages and buys increasingly expensive products, the nPath query returns the first purchase and the most expensive purchase.

**nPath Example Input Table: aggregate\_clicks**

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
1039	1	sneakers	home	2009-07-29 20:17:59	Company1	100
1039	2	books	home	2009-04-21 13:17:59	Company4	300
1039	3	television	home	2009-05-23 13:17:59	Company2	500
1039	4	envelopes	home	2009-07-16 11:17:59	Company3	10
1039	4	envelopes	home1	2009-07-16 11:18:16	Company3	10
1039	4	envelopes	page1	2009-07-16 11:18:18	Company3	10
1039	5	bookcases	home	2009-08-19 22:17:59	Company5	150
1039	5	bookcases	home1	2009-08-19 22:18:02	Company5	150
1039	5	bookcases	page1	2009-08-19 22:18:05	Company5	150
1039	5	bookcases	page2	2009-08-22 04:20:05	Company5	150
1039	5	bookcases	checkout	2009-08-24 14:30:05	Company5	150
1039	5	bookcases	page2	2009-08-27 23:03:05	Company5	150
1040	1	tables	home	2009-07-29 20:17:59	Company5	250
1040	2	Appliance	home	2009-04-21 13:17:59	Company6	1500
1040	3	laptops	home	2009-05-23 13:17:59	Company7	800
1040	4	chairs	home	2009-07-16 11:17:59	Company3	400
1040	4	chairs	home1	2009-07-16	Company3	400

userid	sessionid	productname	pagetype	clicktime	referrer	productprice
				11:18:16		
1040	4	chairs	page1	2009-07-16 11:18:18	Company3	400
1040	5	cellphones	home	2009-08-19 22:17:59	Company8	600
1040	5	cellphones	home1	2009-08-19 22:18:02	Company8	600
1040	5	cellphones	page1	2009-08-19 22:18:05	Company8	600
1040	5	cellphones	page2	2009-08-22 04:20:05	Company8	600
1040	5	cellphones	checkout	2009-08-24 14:30:05	Company8	600
1040	5	cellphones	page2	2009-08-27 23:03:05	Company8	600
...	...	...	...	...	...	...

## nPath Range-Matching Example: Accumulate Pages Visited in Each Session

### Input

The input table is aggregate\_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

### SQL-MapReduce Call

```
SELECT * FROM nPath (
  ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
  USING
    Mode (NONOVERLAPPING)
    Pattern ('A*')
    Symbols (TRUE AS A)
  Result (
    FIRST (sessionid OF A) AS sessionid,
    ACCUMULATE (pagetype OF A) AS path
  )
) AS dt ORDER BY dt.sessionid;
```

## Output

sessionid	path
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home]
2	[home, home, home, home, home, home, home, home, home, home1, page1, checkout, home, home]
3	[home, home, home, home, home, home, home, home, home1, page1, home, home1, page1, home]
4	[home, home, home, home, home, home, home1, home1, home1, page1, page1]
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]

## nPath Range-Matching Example: Find Sessions That Start at Home Page and Visit Page1

### Input

The input table is aggregate\_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

### SQL-MapReduce Call

```
SELECT * FROM nPath (
    ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
    USING
    Mode (NONOVERLAPPING)
    Pattern ('^H.A*.P1.A*')
    Symbols (pagetype='home' AS H, pagetype='page1' AS P1, TRUE AS A)
    Result (
        FIRST (sessionid OF A) AS sessionid,
        ACCUMULATE (pagetype OF ANY(H,P1,A)) AS path
    )
) AS dt ORDER BY dt.sessionid;
```

## Output

sessionid	path
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home]
2	[home, home, home, home, home, home, home, home, home, home1, page1, checkout, home, home]

sessionid	path
3	[home, home, home, home, home, home, home, home, home1, page1, home, home1, page1, home]
4	[home, home, home, home, home, home, home1, home1, home1, page1, page1, page1]
5	[home, home, home, home, home1, home1, home1, page1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]

## nPath Range-Matching Example: Find Paths to Checkout Page for Purchases Over \$200

### Input

The input table is aggregate\_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

### SQL-MapReduce Call

```
SELECT * FROM nPath (
    ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
    USING
    Mode (NONOVERLAPPING)
    Pattern ('A*.C+.A*')
    Symbols (
        productprice > 200 AND
        pagetype='checkout' AS C, TRUE AS A
    )
    Result (
        FIRST(sessionid OF A) AS sessionid,
        ACCUMULATE (pagetype OF ANY(A,C)) AS path,
        AVG (productprice OF ANY(A,C)) AS totalsum
    )
) AS dt ORDER BY dt.sessionid;
```

### Output

sessionid	path	totalsum
1	[home, home1, page1, home, home1, page1, home, home, home, home1, page1, checkout, home, home, home, home, home, home, home, home, home]	602.857142857143
5	[home, home, home, home, home1, home1, home1, page1, page1, page2, page2, page2, checkout, checkout, checkout, page2, page2, page2]	363.157894736842

## nPath Range-Matching Example: Use OVERLAPPING Mode

### Input

The input table is aggregate\_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

### SQL-MapReduce Call

```
SELECT * FROM nPath (
    ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
    USING
        Mode (OVERLAPPING)
        Pattern ('A.A')
        Symbols (TRUE AS A)
    Result (
        FIRST (sessionid OF A) AS sessionid,
        ACCUMULATE (pagetype OF A) AS path
    )
) AS dt ORDER BY dt.sessionid;
```

## nPath Range-Matching Example Output

sessionid	path
1	[home, home]
1	[checkout, home]
1	[page1, checkout]
1	[home1, page1]
1	[home, home1]
1	[home, home]
1	[home, home]

sessionid	path
1	[page1, home]
1	[home1, page1]
1	[home, home1]
1	[page1, home]
1	[home1, page1]
1	[home, home1]
2	[home, home]
2	[checkout, home]
2	[checkout, checkout]
...	...

## nPath Range-Matching Example: Find First Product with Multiple Referrers in Any Session

### Input

The input table is aggregate\_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

### SQL-MapReduce Call

```
SELECT * FROM nPath (
    ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
    USING
        Mode (NONOVERLAPPING)
        Pattern ('REFERRER{2,}')
        Symbols (referrer IS NOT NULL AS REFERRER)
        Result (
            FIRST (sessionid OF REFERRER) AS sessionid,
            FIRST (productname OF REFERRER) AS product
        )
) AS dt ORDER BY dt.sessionid;
```

### Output

sessionid	product
1	envelopes
2	tables

sessionid	product
3	bookcases
4	tables
5	Appliances

## nPath Range-Matching Example: Find Data for Sessions That Checked Out 3-6 Products

### Input

The input table is aggregate\_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

### SQL-MapReduce Call

```
SELECT * FROM nPath (
    ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
    USING
        Mode (NONOVERLAPPING)
        Pattern ('H+.D*.C{3,6}.D')
        Symbols (
            pagetype = 'home' AS H,
            pagetype='checkout' AS C,
            pagetype<>'home' AND pagetype<>'checkout' AS D
        )
    Result (
        FIRST (sessionid OF C) AS sessionid,
        max_choose (productprice, productname OF C) AS
        most_expensive_product,
        MAX (productprice OF C) AS max_price,
        min_choose (productprice, productname of C) AS
        least_expensive_product,
        MIN (productprice OF C) AS min_price)
    ) AS dt ORDER BY dt.sessionid;
```

### Output

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
5	cellphones	600	bookcases	150

## nPath Range-Matching Example: Find Data for Sessions That Checked Out at Least 3 Products

### Input

The input table is aggregate\_clicks, from [LAG and LEAD Expressions Example: No Alias for Input Query](#).

Modify the previous query call in [nPath Range-Matching Example: Find Data for Sessions That Checked Out 3-6 Products](#) to find sessions where the user checked out at least three products by changing the Pattern syntax element to:

```
Pattern ('H+.D*.C{3,}.D')
```

### SQL-MapReduce Call

```
SELECT * FROM nPath (
    ON aggregate_clicks PARTITION BY sessionid ORDER BY clicktime
    USING
        Mode (NONOVERLAPPING)
        Pattern ('H+.D*.C{3,}.D')
        Symbols (
            pagetype = 'home' AS H,
            pagetype='checkout' AS C,
            pagetype<>'home' AND pagetype<>'checkout' AS D
        )
    Result (
        FIRST(sessionid OF C) AS sessionid,
        max_choose(productprice, productname OF C) AS
        most_expensive_product,
        MAX (productprice OF C) AS max_price,
        min_choose (productprice, productname OF C) AS
        least_expensive_product,
        MIN (productprice OF C) AS min_price
    )
) AS dt ORDER BY dt.sessionid;
```

### Output

sessionid	most_expensive_product	max_price	least_expensive_product	min_price
5	cellphones	600	bookcases	150

## nPath Range-Matching Example: Multiple Partitioned Input Tables and Dimension Input Table

An e-commerce store wants to count the advertising impressions that lead to a user clicking an online advertisement. The example counts the online advertisements that the user viewed and the television advertisements that the user might have viewed.

### Input

**impressions**

userid	ts	imp
1	2012-01-01	ad1
1	2012-01-02	ad1
1	2012-01-03	ad1
1	2012-01-04	ad1
1	2012-01-05	ad1
1	2012-01-06	ad1
1	2012-01-07	ad1
2	2012-01-08	ad2
2	2012-01-09	ad2
2	2012-01-10	ad2
2	2012-01-11	ad2
...	...	...

**clicks2**

userid	ts	click
1	2012-01-01	ad1
2	2012-01-08	ad2
3	2012-01-16	ad3
4	2012-01-23	ad4
5	2012-02-01	ad5
6	2012-02-08	ad6
7	2012-02-14	ad7

userid	ts	click
8	2012-02-24	ad8
9	2012-03-02	ad9
10	2012-03-10	ad10
11	2012-03-18	ad11
12	2012-03-25	ad12
13	2012-03-30	ad13
14	2012-04-02	ad14
15	2012-04-06	ad15

**tv\_spots**

ts	tv_imp
2012-01-01	ad2
2012-01-02	ad2
2012-01-03	ad3
2012-01-04	ad4
2012-01-05	ad5
2012-01-06	ad6
2012-01-07	ad7
2012-01-08	ad8
2012-01-09	ad9
2012-01-10	ad10
2012-01-11	ad11
2012-01-12	ad12
2012-01-13	ad13
2012-01-14	ad14
2012-01-15	ad15

## SQL-MapReduce Call

The tables impressions and clicks have a user\_id column, but the table tv\_spots is only a record of television advertisements shown, which any user might have seen. Therefore, tv\_spots must be a dimension table.

```
SELECT * FROM nPath (
    ON impressions PARTITION BY userid ORDER BY ts
    ON clicks2 PARTITION BY userid ORDER BY ts
    ON tv_spots DIMENSION ORDER BY ts
    USING
        Mode (NONOVERLAPPING)
        Symbols (TRUE AS imp, TRUE AS click, TRUE AS tv_imp)
        Pattern ('(imp|tv_imp)*.click')
    Result (
        COUNT(* of imp) AS imp_cnt,
        COUNT (* of tv_imp) AS tv_imp_cnt
    )
) AS dt ORDER BY dt.imp_cnt;
```

## Output

dt.imp_cnt	tv_imp_cnt
18	0
19	0
19	0
20	0
21	0
22	0
22	0
22	0
22	0
23	0
23	0
23	0
24	0

dt.imp_cnt	tv_imp_cnt
25	0

## Sessionize

The Sessionize function maps each click in a session to a unique session identifier. A session is a sequence of clicks by one user that are separated by at most  $n$  seconds.

The function is useful for both sessionization and detecting web crawler (bot) activity. A typical use is to understand user browsing behavior on a website.

Sessionization is the process of identifying and grouping together all the interactions a user has with a website or application during a single visit, typically based on the user's activity within a certain timeframe, such as prior to 30 minutes of inactivity.

Clickstream data refers to the sequence of clicks and other user interactions with a website or application. Sessionization of clickstream data involves analyzing this data to identify each user's session, so you can better understand how users are interacting with the site and make improvements to user experience and engagement.

Consider an input clickstream table, where each row is a webpage click made by a particular user. The sessionization computation attempts to identify browsing sessions by grouping the clicks from each user based on the time-intervals between the clicks. If two clicks from the same user are made too far apart, as defined by a time-out threshold, the clicks are treated as though they are from two different browsing sessions.

The benefits of sessionization of clickstream data include:

- Personalization: By understanding a user's session, you can personalize the user experience by presenting content or products that are more relevant to their interests and behavior.
- Improving conversion rates: Sessionization can help you identify bottlenecks in the user journey that are causing users to drop off before completing a task. By identifying these issues and making improvements, you can increase conversion rates.
- Fraud detection: Sessionization can also help identify fraudulent activity, such as bots or automated scripts, by analyzing the patterns of user interactions within a session.

## Function Information

- [Sessionize Syntax](#)
- [Required Syntax Elements for Sessionize](#)
- [Optional Syntax Elements for Sessionize](#)
- [Sessionize Input](#)
- [Sessionize Output](#)
- [Example: How to Use Sessionize](#)

## Sessionize Syntax

```
SESSIONIZE (
    ON { table | view | (query) }
    PARTITION BY expression [,...]
    ORDER BY order_column [,...]
    USING
        TimeColumn ('time_column')
        TimeOut (session_timeout)
        [ ClickLag (min_click_lag) ]
        [ EmitNull ({'true'|'t'|'yes'|'y'|'1'|'false'|'f'|'no'|'n'|'0'})]
)
)
```

### Note:

You can call this function from:

- The FROM clause of a SELECT statement
- As part of a CREATE TABLE statement
- As part of a CREATE VIEW statement

## Required Syntax Elements for Sessionize

### ON clause

Accepts the clickstream input.

### TimeColumn

Specify the name of the input column that contains the click times.

The time\_column must also be an order\_column.

### TimeOut

Specify the number of seconds that the session times out. If timeout seconds elapse after a click, then the next click starts a new session. The data type of session\_timeout is DOUBLE PRECISION.

## Optional Syntax Elements for Sessionize

### ClickLag

Specify the minimum number of seconds (lag) between clicks, such as 0.2 seconds between each click, for the session user to be considered human. If time between clicks is less than

the lag timeframe, indicating that the user is a bot, the function ignores the session. The minimum lag must be less than the timeout threshold. The data type of *min\_click\_lag* is DOUBLE PRECISION.

Default: The function ignores no session, regardless of click frequency.

#### EmitNull

Specify whether to output rows that have NULL values in their session id and rapid fire columns, even if their *time\_column* has a NULL value.

Default: false.

## Sessionize Input

### Input Table Schema

Column	Data Type	Description
time_column	TIME, TIMESTAMP, INTEGER, BIGINT, SMALLINT, or DATE	Click times (in milliseconds if data type is INTEGER, BIGINT, or SMALLINT).
session_timeout	DOUBLE PRECISION	Timeout threshold for a session.
partition_column	Any	Column that partitions input. Input data must be partitioned such that each partition contains all rows of an entity. For example, an entity could be user ID if you want to analyze users' web behavior. It would require partitioning web actions by each user and then sessionizing them according to given criteria. This provides sessionized actions for each user. Conversely, an entity could be device ID if evaluating IoT device activity over network traffic.
order_column	Any	Column by which input data is ordered.

---

#### Note:

No input table column can have the name 'sessionid' or 'clicklag' because these are output table column names.

---

#### Tip:

To create a single timestamp column from separate date and time columns:

```
SELECT (datecolumn || ' ' || timecolumn)::timestamp AS
mytimestamp FROM table;
```

---

## Sessionize Output

### Output Table Schema

Column	Data Type	Description
input_column	Same as in input table	Column copied from input table. Function copies every input_column to output table.
sessionid	INTEGER or BIGINT	Identifier that function assigned to session.
clicklag	BYTEINT	Indicator that there were lags between clicks. Values are 1 if the session exceeded the threshold, and '0 otherwise.

## Example: How to Use Sessionize

Every complete example in this document is available in a zip file that you can download. The zip file includes a SQL script file that creates the input tables for the examples. If you are reading this document on <https://docs.teradata.com/>, you can download the zip file from the attachment  in the left sidebar.

### Sessionize Input

#### sessionize\_table

clicktime	userid	productname	pagetype	referrer	productprice
2023-03-17 11:31:59	333	Ipod	Checkout	www.yahoo.com	200.20
2023-03-17 12:07:20	67403	Home	NULL	NULL	NULL
2023-03-17 12:03:20	67403	Home	NULL	www.google.com	NULL
2023-03-17 11:50:00	67403	Home	NULL	www.google.com	NULL
2023-03-17 12:22:00	80000	Home	NULL	godaddy.com	NULL
2023-03-17 12:21:40	67403	Bose	Checkout	NULL	750.00
2023-03-17 12:26:40	880000	Iphone	Checkout	NULL	650.00
2023-03-17 11:40:00	333	Bose	Checkout	NULL	340.00
2023-03-17 12:13:20	67403	Iphone	Checkout	NULL	650.00
2023-03-17 11:46:40	333	Home	NULL	www.google.com	NULL
2023-03-17 12:05:00	67403	Home	NULL	NUJLL	NULL
2023-03-17 12:23:20	80000	Bose	Checkout	NULL	340.00
2023-03-17 11:30:00	333	Home	NULL	www.yahoo.com	NULL
2023-03-17 12:23:20	80000	Itrip	Checkout	NULL	450.00

## Sessionize SQL Call

```
SELECT * FROM SESSIONIZE (
    ON sessionize_table PARTITION BY user_id ORDER BY clicktime
    USING
        TimeColumn ('clicktime')
        TimeOut (300)
        ClickLag (0.2)
) ORDER BY user_id, clicktime;
```

## Sessionize Output

clicktime	userid	productname	pagetype	referrer	productprice	SessionID	Clicklag
2023-03-17 11:30:00	333	Home	NULL	www.yahoo.com	NULL	0	f
2023-03-17 11:31:59	333	Ipod	Checkout	www.yahoo.com	200.20	0	f
2023-03-17 11:40:00	333	Bose	Checkout	NULL	340.00	1	f
2023-03-17 11:46:40	333	Home	NULL	www.google.com	NULL	2	f
2023-03-17 11:50:00	67403	Home	NULL	www.google.com	NULL	0	f
2023-03-17 12:03:20	67403	Home	NULL	www.google.com	NULL	1	f
2023-03-17 12:05:00	67403	Home	NULL	NULL	NULL	1	f
2023-03-17 12:07:20	67403	Home	NULL	NULL	NULL	1	f
2023-03-17 12:13:20	67403	Iphone	Checkout	NULL	650.00	2	f
2023-03-17 12:21:40	67403	Bose	Checkout	NULL	750.00	3	f
2023-03-17 12:22:00	80000	Home	NULL	godaddy.com	NULL	0	t
2023-03-17 12:23:20	80000	Bose	Checkout	NULL	340.00	0	f
2023-03-17 12:23:20	80000	ltrip	Checkout	NULL	450.00	0	f
2023-03-17 12:26:40	880000	Iphone	Checkout	NULL	650.00	0	f

There's a new session recorded for each user if time between their clicks was more than 5 minutes (300 seconds). For userid 80000, time between two clicks was less than 0.2 seconds, and therefore recorded in clicklag column as 't' for True.

# How to Read Syntax

This document uses the following syntax conventions.

Syntax Convention	Meaning
KEYWORD	Keyword. Spell exactly as shown. Many environments are case-insensitive. Syntax shows keywords in uppercase unless operating system restrictions require them to be lowercase or mixed-case.
<i>variable</i>	Variable. Replace with actual value.
<i>number</i>	String of one or more digits. Do not use commas in numbers with more than three digits. Example: 10045
[ <i>x</i> ]	<i>x</i> is optional.
[ <i>x</i>   <i>y</i> ]	You can specify <i>x</i> , <i>y</i> , or nothing.
{ <i>x</i>   <i>y</i> }	You must specify either <i>x</i> or <i>y</i> .
<i>x</i> [ ... ]	You can repeat <i>x</i> , separating occurrences with spaces. Example: <i>x</i> <i>x</i> <i>x</i> See note after table.
<i>x</i> [ ,... ]	You can repeat <i>x</i> , separating occurrences with commas. Example: <i>x</i> , <i>x</i> , <i>x</i> See note after table.
<i>x</i> [ <i>delimiter</i> ... ]	You can repeat <i>x</i> , separating occurrences with specified delimiter. Examples: <ul style="list-style-type: none"> <li>• If <i>delimiter</i> is semicolon: <i>x</i>; <i>x</i>; <i>x</i></li> <li>• If <i>delimiter</i> is {,  OR}, you can do either of the following:<ul style="list-style-type: none"> <li>◦ <i>x</i>, <i>x</i>, <i>x</i></li> <li>◦ <i>x</i> OR <i>x</i> OR <i>x</i></li> </ul></li> </ul> See note after table.
( [ <i>x</i> [ [ , ] <i>y</i> ] ] )	Depends on context. See descriptions of syntax elements that use parentheses.

---

**Note:**

You can repeat only the immediately preceding item. For example, if the syntax is:

```
KEYWORD x [...]
```

You can repeat x. Do not repeat KEYWORD.

If there is no white space between x and the delimiter, the repeatable item is x and the delimiter. For example, if the syntax is:

```
[ x, [...] ] y
```

- You can omit x: y
  - You can specify x once: x, y
  - You can repeat x and the delimiter: x, x, x, y
-

# Additional Information

## Teradata Links

Link	Description
<a href="https://docs.teradata.com/">https://docs.teradata.com/</a>	Search Teradata Documentation, customize content to your needs, and download PDFs. Customers: Log in to access Orange Books.
<a href="https://support.teradata.com">https://support.teradata.com</a>	Helpful resources in one place: <ul style="list-style-type: none"> <li>• Support requests</li> <li>• Account management and software downloads</li> <li>• Knowledge base, community, and support policies</li> <li>• Product documentation</li> <li>• Learning resources, including Teradata University</li> </ul>
<a href="https://www.teradata.com/University/Overview">https://www.teradata.com/University/Overview</a>	Teradata education network
<a href="https://support.teradata.com/community">https://support.teradata.com/community</a>	Link to Teradata community

## Related Documentation

Title	Publication ID
<i>Teradata Vantage™ - Analytics Database Release Summary</i>	B035-1098
<i>Teradata Vantage™ - SQL Fundamentals</i>	B035-1141
<i>Teradata Vantage™ - SQL Functions, Expressions, and Predicates</i>	B035-1145
<i>Teradata Vantage™ - SQL Data Manipulation Language</i>	B035-1146
<i>International Character Set Support</i>	B035-1125