# Package 'Rodeo'

March 15, 2023

**Title** Rodeo

**Version** 0.0.1

**Date** 2023-03-15

**Maintainer** Adrian Antico <adrianantico@gmail.com>

**Description** R optimized data engineering operations

**URL** https://github.com/AdrianAntico/Rodeo

**BugReports** https://github.com/AdrianAntico/Rodeo/issues

**Depends** R (>= 3.5.0)

**Imports** bit64, data.table, doParallel, foreach, lubridate, timeDate

**Suggests** knitr, rmarkdown, gridExtra

**VignetteBuilder** knitr

**Contact** Adrian Antico

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**NeedsCompilation** no

**RoxygenNote** 7.2.1

**Author** Adrian Antico [aut, cre]

**ByteCompile** TRUE

## R topics documented:

---

`AutoClustering` *AutoClustering*

---

## Description

AutoClustering adds a column to your original data with a cluster number identifier. You can run request an autoencoder to be built to reduce the dimensionality of your data before running the clusering algo.

## Usage

```
AutoClustering(
  data,
  FeatureColumns = NULL,
  ModelID = "TestModel",
  SavePath = NULL,
  NThreads = 8,
  MaxMemory = "28G",
  MaxClusters = 50,
  ClusterMetric = "totss",
  RunDimReduction = TRUE,
  ShrinkRate = (sqrt(5) - 1)/2,
  Epochs = 5L,
  L2_Reg = 0.1,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.9,
  ElasticAveragingRegularization = 0.001
)
```

## Arguments

| | |
|---|---|
| data | is the source time series data.table |
| FeatureColumns | Independent variables |
| ModelID | For naming the files to save |
| SavePath | Directory path for saving models |
| NThreads | set based on number of threads your machine has available |
| MaxMemory | set based on the amount of memory your machine has available |
| MaxClusters | number of factors to test out in k-means to find the optimal number |
| ClusterMetric | pick the metric to identify top model in grid tune c('totss','betweenss','withinss') |
| RunDimReduction | If TRUE, an autoencoder will be built to reduce the feature space. Otherwise, all features in FeatureColumns will be used for clustering |
| ShrinkRate | Node shrink rate for H2OAutoencoder. See that function for details. |
| Epochs | For the autoencoder |
| L2_Reg | For the autoencoder |
| ElasticAveraging | For the autoencoder |
| ElasticAveragingMovingRate | For the autoencoder |
| ElasticAveragingRegularization | For the autoencoder |

## Value

Original data.table with added column with cluster number identifier

## Author(s)

Adrian Antico

## See Also

Other Unsupervised Learning: AutoClusteringScoring(), H2OIsolationForestScoring(), H2OIsolationForest()

## Examples

```
## Not run:
#########################
# Training Setup
#########################

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)
```

```
# Run function
data <- AutoQuant::AutoClustering(
  data,
  FeatureColumns = names(data)[2:(ncol(data)-1)],
  ModelID = 'TestModel',
  SavePath = getwd(),
  NThreads = 8,
  MaxMemory = '28G',
  MaxClusters = 50,
  ClusterMetric = 'totss',
  RunDimReduction = TRUE,
  ShrinkRate = (sqrt(5) - 1) / 2,
  Epochs = 5L,
  L2_Reg = 0.10,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.90,
  ElasticAveragingRegularization = 0.001)

#########################
# Scoring Setup
#########################

Sys.sleep(10)

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
data <- AutoQuant::AutoClusteringScoring(
  data,
  FeatureColumns = names(data)[2:(ncol(data)-1)],
  ModelID = 'TestModel',
  SavePath = getwd(),
  NThreads = 8,
  MaxMemory = '28G',
  DimReduction = TRUE)

## End(Not run)
```

---

`AutoClusteringScoring` *AutoClusteringScoring*

---

## Description

AutoClusteringScoring adds a column to your original data with a cluster number identifier. You can run request an autoencoder to be built to reduce the dimensionality of your data before running the clusering algo.

## Usage

```
AutoClusteringScoring(
  data,
  FeatureColumns = NULL,
  ModelID = "TestModel",
  SavePath = NULL,
  NThreads = 8,
  MaxMemory = "28G",
  DimReduction = TRUE
)
```

## Arguments

| | |
|---|---|
| data | is the source time series data.table |
| FeatureColumns | Independent variables |
| ModelID | This is returned from the training run in the output list with element named 'model_name'. It's not identical to the ModelID used in training due to the grid tuning. |
| SavePath | Directory path for saving models |
| NThreads | set based on number of threads your machine has available |
| MaxMemory | set based on the amount of memory your machine has available |
| DimReduction | Set to TRUE if you set RunDimReduction in the training version of this function |

## Value

Original data.table with added column with cluster number identifier

## Author(s)

Adrian Antico

## See Also

Other Unsupervised Learning: AutoClustering(), H2OIsolationForestScoring(), H2OIsolationForest()

## Examples

```
## Not run:
#########################
# Training Setup
#########################

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)
```

```
# Run function
data <- AutoQuant::AutoClustering(
  data,
  FeatureColumns = names(data)[2:(ncol(data)-1)],
  ModelID = 'TestModel',
  SavePath = getwd(),
  NThreads = 8,
  MaxMemory = '28G',
  MaxClusters = 50,
  ClusterMetric = 'totss',
  RunDimReduction = TRUE,
  ShrinkRate = (sqrt(5) - 1) / 2,
  Epochs = 5L,
  L2_Reg = 0.10,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.90,
  ElasticAveragingRegularization = 0.001)

##########################
# Scoring Setup
##########################

Sys.sleep(10)

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
  ZIP = 0,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run function
data <- AutoQuant::AutoClusteringScoring(
  data,
  FeatureColumns = names(data)[2:(ncol(data)-1)],
  ModelID = 'TestModel',
  SavePath = getwd(),
  NThreads = 8,
  MaxMemory = '28G',
  DimReduction = TRUE)

## End(Not run)
```

---

AutoDataPartition *AutoDataPartition*

---

## Description

This function will take your ratings matrix and model and score your data in parallel.

## Usage

```
AutoDataPartition(
  data,
  NumDataSets = 3L,
  Ratios = c(0.7, 0.2, 0.1),
  PartitionType = "random",
  StratifyColumnNames = NULL,
  TimeColumnName = NULL
)
```

## Arguments

| | |
|---|---|
| data | Source data to do your partitioning on |
| NumDataSets | The number of total data sets you want built |
| Ratios | A vector of values for how much data each data set should get in each split. E.g. c(0.70, 0.20, 0.10) |
| PartitionType | Set to either "random", "timeseries", or "time". With "random", your data will be paritioned randomly (with stratified sampling if column names are supplied). With "timeseries", you can partition by time with a stratify option (so long as you have an equal number of records for each strata). With "time" you will have data sets generated so that the training data contains the earliest records in time, validation data the second earliest, test data the third earliest, etc. |
| StratifyColumnNames | Supply column names of categorical features to use in a stratified sampling procedure for partitioning the data. Partition type must be "random" to use this option |
| TimeColumnName | Supply a date column name or a name of a column with an ID for sorting by time such that the smallest number is the earliest in time. |

## Value

Returns a list of data.tables

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000,
  ID = 2,
```

```
  ZIP = 0,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run data partitioning function
dataSets <- AutoQuant::AutoDataPartition(
  data,
  NumDataSets = 3L,
  Ratios = c(0.70,0.20,0.10),
  PartitionType = "random",
  StratifyColumnNames = NULL,
  TimeColumnName = NULL)

# Collect data
TrainData <- dataSets$TrainData
ValidationData <- dataSets$ValidationData
TestData <- dataSets$TestData
```

---

`AutoDiffLagN` *AutoDiffLagN*

---

## Description

AutoDiffLagN create differences for selected numerical columns

## Usage

```
AutoDiffLagN(
  data,
  DateVariable = NULL,
  GroupVariables = NULL,
  DiffVariables = NULL,
  DiffDateVariables = NULL,
  DiffGroupVariables = NULL,
  NLag1 = 0L,
  NLag2 = 1L,
  Type = "lag",
  Sort = FALSE,
  RemoveNA = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | Source data |
| `DateVariable` | Date column used for sorting |
| `GroupVariables` | Difference data by group |
| `DiffVariables` | Column names of numeric columns to difference |
| `DiffDateVariables` | Columns names for date variables to difference. Output is a numeric value representing the difference in days. |

DiffGroupVariables
: Column names for categorical variables to difference. If no change then the output is 'No_Change' else 'New=NEWVAL Old=OLDVAL' where NEWVAL and OLDVAL are placeholders for the actual values

NLag1
: If the diff calc, we have column 1 - column 2. NLag1 is in reference to column 1. If you want to take the current value minus the previous weeks value, supply a zero. If you want to create a lag2 - lag4 NLag1 gets a 2.

NLag2
: If the diff calc, we have column 1 - column 2. NLag2 is in reference to column 2. If you want to take the current value minus the previous weeks value, supply a 1. If you want to create a lag2 - lag4 NLag1 gets a 4.

Type
: 'lag' or 'lead'

Sort
: TRUE to sort your data inside the function

RemoveNA
: Set to TRUE to remove rows with NA generated by the lag operation

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 3L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Store Cols to diff
Cols <- names(data)[which(unlist(data[, lapply(.SD, is.numeric)]))]

# Clean data before running AutoDiffLagN
data <- AutoQuant::ModelDataPrep(data = data, Impute = FALSE, CharToFactor = FALSE, FactorToChar = TRUE)

# Run function
data <- AutoQuant::AutoDiffLagN(
  data,
  DateVariable = "DateTime",
```

```
  GroupVariables = c("Factor_1", "Factor_2"),
  DiffVariables = Cols,
  DiffDateVariables = NULL,
  DiffGroupVariables = NULL,
  NLag1 = 0L,
  NLag2 = 1L,
  Sort = TRUE,
  RemoveNA = TRUE)

## End(Not run)
```

---

`AutoInteraction` *AutoInteraction*

---

## Description

AutoInteraction creates interaction variables from your numerical features in your data. Supply a set of column names to utilize and set the interaction level. Supply a character vector of columns to exclude and the function will ignore those features.

## Usage

```
AutoInteraction(
  data = NULL,
  NumericVars = NULL,
  InteractionDepth = 2,
  Center = TRUE,
  Scale = TRUE,
  SkipCols = NULL,
  Scoring = FALSE,
  File = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | Source data.table |
| `InteractionDepth` | The max K in N choose K. If NULL, K will loop through 1 to length(NumVars). Default is 2 for pairwise interactions |
| `Center` | TRUE to center the data |
| `Scale` | TRUE to scale the data |
| `SkipCols` | Use this to exclude features from being created. An example could be, you build a model with all variables and then use the varaible importance list to determine which features aren't necessary and pass that set of features into this argument as a character vector. |
| `Scoring` | Defaults to FALSE. Set to TRUE for generating these columns in a model scoring setting |
| `File` | When Scoring is set to TRUE you have to supply either the .Rdata list with lookup values for recreating features or a pathfile to the .Rdata file with the lookup values. If you didn't center or scale the data then this argument can be ignored. |

NumVars — Names of numeric columns (if NULL, all numeric and integer columns will be used)

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoLagRollMode(), AutoLagRollStatsScori
AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(),
AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(),
DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(),
PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:

########################################
# Feature Engineering for Model Training
########################################

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Print number of columns
print(ncol(data))

# Store names of numeric and integer cols
Cols <-names(data)[c(which(unlist(lapply(data, is.numeric))),
                     which(unlist(lapply(data, is.integer))))]

# Model Training Feature Engineering
system.time(data <- AutoQuant::AutoInteraction(
  data = data,
  NumericVars = Cols,
  InteractionDepth = 4,
  Center = TRUE,
  Scale = TRUE,
  SkipCols = NULL,
  Scoring = FALSE,
  File = getwd()))

# user  system elapsed
# 0.30    0.11    0.41
```

```
# Print number of columns
print(ncol(data))

########################################
# Feature Engineering for Model Scoring
########################################

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Print number of columns
print(ncol(data))

# Reduce to single row to mock a scoring scenario
data <- data[1L]

# Model Scoring Feature Engineering
system.time(data <- AutoQuant::AutoInteraction(
  data = data,
  NumericVars = names(data)[
    c(which(unlist(lapply(data, is.numeric))),
      which(unlist(lapply(data, is.integer))))],
  InteractionDepth = 4,
  Center = TRUE,
  Scale = TRUE,
  SkipCols = NULL,
  Scoring = TRUE,
  File = file.path(getwd(), "Standardize.Rdata")))

# user  system elapsed
# 0.19    0.00    0.19

# Print number of columns
print(ncol(data))

## End(Not run)
```

---

AutoLagRollMode *AutoLagRollMode*

---

## Description

Create lags and rolling modes for categorical variables.

## Usage

```
AutoLagRollMode(
  data,
  Lags = 1,
  ModePeriods = 0,
  Targets = NULL,
  GroupingVars = NULL,
  SortDateName = NULL,
  WindowingLag = 0,
  Type = c("Lag"),
  SimpleImpute = TRUE,
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | A data.table you want to run the function on |
| Lags | A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1. |
| ModePeriods | A numberic vector of window sizes |
| Targets | A character vector of the column names for the reference column in which you will build your lags and rolling stats |
| GroupingVars | A character vector of categorical variable names you will build your lags and rolling stats by |
| SortDateName | The column name of your date column used to sort events over time |
| WindowingLag | Set to 0 to build rolling stats off of target columns directly or set to 1 to build the rolling stats off of the lag-1 target |
| Type | List either "Lag" if you want features built on historical values or "Lead" if you want features built on future values |
| SimpleImpute | Set to TRUE for factor level imputation of "0" and numeric imputation of -1 |
| Debug | = FALSE |

## Value

data.table of original data plus created lags, rolling stats, and time between event lags and rolling stats

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollStatsScori
AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(),
AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(),
DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(),
PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# NO GROUPING CASE: Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- AutoQuant::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 2L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)
  datatemp[, Factor1 := eval(Level)]
  if(Count == 1L) {
    data <- data.table::copy(datatemp)
  } else {
    data <- data.table::rbindlist(
      list(data, data.table::copy(datatemp)))
  }
  Count <- Count + 1L
}

# NO GROUPING CASE: Create rolling modes for categorical features
data <- AutoQuant::AutoLagRollMode(
  data,
  Lags           = seq(1,5,1),
  ModePeriods    = seq(2,5,1),
  Targets        = c("Factor_1"),
  GroupingVars   = NULL,
  SortDateName   = "DateTime",
  WindowingLag   = 1,
  Type           = "Lag",
  SimpleImpute   = TRUE)

# GROUPING CASE: Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- AutoQuant::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 2L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)
  datatemp[, Factor1 := eval(Level)]
  if(Count == 1L) {
    data <- data.table::copy(datatemp)
  } else {
    data <- data.table::rbindlist(
      list(data, data.table::copy(datatemp)))
  }
  Count <- Count + 1L
```

```
}

# GROUPING CASE: Create rolling modes for categorical features
data <- AutoQuant::AutoLagRollMode(
  data,
  Lags           = seq(1,5,1),
  ModePeriods    = seq(2,5,1),
  Targets        = c("Factor_1"),
  GroupingVars   = "Factor_2",
  SortDateName   = "DateTime",
  WindowingLag   = 1,
  Type           = "Lag",
  SimpleImpute   = TRUE)

## End(Not run)
```

---

AutoLagRollStats *AutoLagRollStats*

---

## Description

AutoLagRollStats Builds lags and a large variety of rolling statistics with options to generate them for hierarchical categorical interactions.

## Usage

```
AutoLagRollStats(
  data,
  Targets = NULL,
  HierarchyGroups = NULL,
  IndependentGroups = NULL,
  DateColumn = NULL,
  TimeUnit = NULL,
  TimeUnitAgg = NULL,
  TimeGroups = NULL,
  TimeBetween = NULL,
  RollOnLag1 = TRUE,
  Type = "Lag",
  SimpleImpute = TRUE,
  Lags = NULL,
  MA_RollWindows = NULL,
  SD_RollWindows = NULL,
  Skew_RollWindows = NULL,
  Kurt_RollWindows = NULL,
  Quantile_RollWindows = NULL,
  Quantiles_Selected = NULL,
  ShortName = TRUE,
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | A data.table you want to run the function on |
| Targets | A character vector of the column names for the reference column in which you will build your lags and rolling stats |
| HierarchyGroups | A vector of categorical column names that you want to have generate all lags and rolling stats done for the individual columns and their full set of interactions. |
| IndependentGroups | A vector of categorical column names that you want to have run independently of each other. This will mean that no interaction will be done. |
| DateColumn | The column name of your date column used to sort events over time |
| TimeUnit | List the time aggregation level for the time between events features, such as "hour", "day", "weeks", "months", "quarter", or "year" |
| TimeUnitAgg | List the time aggregation of your data that you want to use as a base time unit for your features. E.g. "raw" or "day" |
| TimeGroups | A vector of TimeUnits indicators to specify any time-aggregated GDL features you want to have returned. E.g. c("raw" (no aggregation is done),"hour","day","week","month","quarter","year") |
| TimeBetween | Specify a desired name for features created for time between events. Set to NULL if you don't want time between events features created. |
| RollOnLag1 | Set to FALSE to build rolling stats off of target columns directly or set to TRUE to build the rolling stats off of the lag-1 target |
| Type | List either "Lag" if you want features built on historical values or "Lead" if you want features built on future values |
| SimpleImpute | Set to TRUE for factor level imputation of "0" and numeric imputation of -1 |
| Lags | A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1. |
| MA_RollWindows | A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations. |
| SD_RollWindows | A numeric vector of Standard Deviation rolling statistics window sizes you want to utilize in the calculations. |
| Skew_RollWindows | A numeric vector of Skewness rolling statistics window sizes you want to utilize in the calculations. |
| Kurt_RollWindows | A numeric vector of Kurtosis rolling statistics window sizes you want to utilize in the calculations. |
| Quantile_RollWindows | A numeric vector of Quantile rolling statistics window sizes you want to utilize in the calculations. |
| Quantiles_Selected | Select from the following c("q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60"," q65", "q70", "q75", "q80", "q85", "q90", "q95") |
| ShortName | Default TRUE. If FALSE, Group Variable names will be added to the rolling stat and lag names. If you plan on have multiple versions of lags and rollings stats by different group variables then set this to FALSE. |
| Debug | Set to TRUE to get a print of which steps are running |

## Value

data.table of original data plus created lags, rolling stats, and time between event lags and rolling stats

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- AutoQuant::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 0L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)
  datatemp[, Factor1 := eval(Level)]
  if(Count == 1L) {
    data <- data.table::copy(datatemp)
  } else {
    data <- data.table::rbindlist(
      list(data, data.table::copy(datatemp)))
  }
  Count <- Count + 1L
}

# Add scoring records
data <- AutoQuant::AutoLagRollStats(
  data                 = data,
  DateColumn           = "DateTime",
  Targets              = "Adrian",
  HierarchyGroups      = NULL,
  IndependentGroups    = c("Factor1"),
  TimeUnitAgg          = "days",
  TimeGroups           = c("days","weeks","months","quarters"),
  TimeBetween          = NULL,
  TimeUnit             = "days",
  RollOnLag1           = TRUE,
  Type                 = "Lag",
  SimpleImpute         = TRUE,
```

```
 Lags            = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarters" =
 MA_RollWindows     = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarter
 SD_RollWindows     = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarter
 Skew_RollWindows    = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarte
 Kurt_RollWindows    = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarte
 Quantile_RollWindows = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quart
  Quantiles_Selected   = c('q5','q50'),
  Debug                = FALSE)

## End(Not run)
```

---

AutoLagRollStatsScoring

*AutoLagRollStatsScoring*

---

## Description

AutoLagRollStatsScoring Builds lags and a large variety of rolling statistics with options to generate them for hierarchical categorical interactions.

## Usage

```
AutoLagRollStatsScoring(
  data,
  RowNumsID = "temp",
  RowNumsKeep = 1,
  Targets = NULL,
  HierarchyGroups = NULL,
  IndependentGroups = NULL,
  DateColumn = NULL,
  TimeUnit = "day",
  TimeUnitAgg = "day",
  TimeGroups = "day",
  TimeBetween = NULL,
  RollOnLag1 = 1,
  Type = "Lag",
  SimpleImpute = TRUE,
  Lags = NULL,
  MA_RollWindows = NULL,
  SD_RollWindows = NULL,
  Skew_RollWindows = NULL,
  Kurt_RollWindows = NULL,
  Quantile_RollWindows = NULL,
  Quantiles_Selected = NULL,
  ShortName = TRUE,
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | A data.table you want to run the function on |

RowNumsID — The name of your column used to id the records so you can specify which rows to keep

RowNumsKeep — The RowNumsID numbers that you want to keep

Targets — A character vector of the column names for the reference column in which you will build your lags and rolling stats

HierarchyGroups — A vector of categorical column names that you want to have generate all lags and rolling stats done for the individual columns and their full set of interactions.

IndependentGroups — Only supply if you do not want HierarchyGroups. A vector of categorical column names that you want to have run independently of each other. This will mean that no interaction will be done.

DateColumn — The column name of your date column used to sort events over time

TimeUnit — List the time aggregation level for the time between events features, such as "hour", "day", "weeks", "months", "quarter", or "year"

TimeUnitAgg — List the time aggregation of your data that you want to use as a base time unit for your features. E.g. "day",

TimeGroups — A vector of TimeUnits indicators to specify any time-aggregated GDL features you want to have returned. E.g. c("hour", "day","week","month","quarter","year"). STILL NEED TO ADD these '1min', '5min', '10min', '15min', '30min', '45min'

TimeBetween — Specify a desired name for features created for time between events. Set to NULL if you don't want time between events features created.

RollOnLag1 — Set to FALSE to build rolling stats off of target columns directly or set to TRUE to build the rolling stats off of the lag-1 target

Type — List either "Lag" if you want features built on historical values or "Lead" if you want features built on future values

SimpleImpute — Set to TRUE for factor level imputation of "0" and numeric imputation of -1

Lags — A numeric vector of the specific lags you want to have generated. You must include 1 if WindowingLag = 1.

MA_RollWindows — A numeric vector of the specific rolling statistics window sizes you want to utilize in the calculations.

SD_RollWindows — A numeric vector of Standard Deviation rolling statistics window sizes you want to utilize in the calculations.

Skew_RollWindows — A numeric vector of Skewness rolling statistics window sizes you want to utilize in the calculations.

Kurt_RollWindows — A numeric vector of Kurtosis rolling statistics window sizes you want to utilize in the calculations.

Quantile_RollWindows — A numeric vector of Quantile rolling statistics window sizes you want to utilize in the calculations.

Quantiles_Selected — Select from the following c("q5", "q10", "q15", "q20", "q25", "q30", "q35", "q40", "q45", "q50", "q55", "q60"," q65", "q70", "q75", "q80", "q85", "q90", "q95")

ShortName — Default TRUE. If FALSE, Group Variable names will be added to the rolling stat and lag names. If you plan on have multiple versions of lags and rollings stats by different group variables then set this to FALSE.

Debug — Set to TRUE to get a print out of which step you are on

## Value

data.table of original data plus created lags, rolling stats, and time between event lags and rolling stats

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
# Create fake Panel Data----
Count <- 1L
for(Level in LETTERS) {
  datatemp <- AutoQuant::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 0L,
    ZIP = 0L,
    FactorCount = 0L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)
  datatemp[, Factor1 := eval(Level)]
  if(Count == 1L) {
    data1 <- data.table::copy(datatemp)
  } else {
    data1 <- data.table::rbindlist(
      list(data1, data.table::copy(datatemp)))
  }
  Count <- Count + 1L
}

# Create ID columns to know which records to score
data1[, ID := .N:1L, by = "Factor1"]
data.table::set(data1, i = which(data1[["ID"]] == 2L), j = "ID", value = 1L)

# Score records
data1 <- AutoQuant::AutoLagRollStatsScoring(

  # Data
  data                = data1,
  RowNumsID           = "ID",
  RowNumsKeep         = 1,
  DateColumn          = "DateTime",
  Targets             = "Adrian",
  HierarchyGroups     = NULL,
  IndependentGroups   = c("Factor1"),
```

```
  # Services
  TimeBetween          = NULL,
  TimeGroups           = c("days","weeks","months","quarters"),
  TimeUnit             = "day",
  TimeUnitAgg          = "day",
  RollOnLag1           = TRUE,
  Type                 = "Lag",
  SimpleImpute         = TRUE,

  # Calculated Columns
 Lags            = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarters" =
 MA_RollWindows     = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarter
 SD_RollWindows     = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarter
 Skew_RollWindows    = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarte
 Kurt_RollWindows    = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quarte
 Quantile_RollWindows = list("days" = c(seq(1,5,1)), "weeks" = c(seq(1,3,1)), "months" = c(seq(1,2,1)), "quart
  Quantiles_Selected   = c('q5','q50'),
  Debug                 = FALSE)
```

---

`AutoTransformationCreate`
*AutoTransformationCreate*

---

## Description

AutoTransformationCreate is a function for automatically identifying the optimal transformations for numeric features and transforming them once identified. This function will loop through your selected transformation options (YeoJohnson, BoxCox, Asinh, Asin, and Logit) and find the one that produces data that is the closest to normally distributed data. It then makes the transformation and collects the metadata information for use in the AutoTransformationScore() function, either by returning the objects (always) or saving them to file (optional).

## Usage

```
AutoTransformationCreate(
  data,
  ColumnNames = NULL,
  Methods = c("BoxCox", "YeoJohnson", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin",
    "Logit", "Identity"),
  Path = NULL,
  TransID = "ModelID",
  SaveOutput = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | This is your source data |
| `ColumnNames` | List your columns names in a vector, for example, c("Target", "IV1") |
| `Methods` | Choose from "YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Asin", "Logit", and "Identity". Note, LogPlus1 runs |
| `Path` | Set to the directly where you want to save all of your modeling files |

| | |
|---|---|
| TransID | Set to a character value that corresponds with your modeling project |
| SaveOutput | Set to TRUE to save necessary file to run AutoTransformationScore() |

## Value

data with transformed columns and the transformation object for back-transforming later

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create Fake Data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 2L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Columns to transform
Cols <- names(data)[1L:11L]
print(Cols)

# Run function
data <- AutoQuant::AutoTransformationCreate(
  data,
  ColumnNames = Cols,
 Methods = c("YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit", "Identity"),
  Path = getwd(),
  TransID = "Trans",
  SaveOutput = TRUE)

## End(Not run)
```

---

# AutoTransformationScore

*AutoTransformationScore() is a the complimentary function to AutoTransformationCreate()*

---

## Description

AutoTransformationScore() is a the compliment function to AutoTransformationCreate(). Automatically apply or inverse the transformations you identified in AutoTransformationCreate() to other data sets. This is useful for applying transformations to your validation and test data sets for modeling. It's also useful for back-transforming your target and prediction columns after you have build and score your models so you can obtain statistics on the original features.

## Usage

```
AutoTransformationScore(
  ScoringData,
  FinalResults,
  Type = "Inverse",
  TransID = "TestModel",
  Path = NULL
)
```

## Arguments

| | |
|---|---|
| ScoringData | This is your source data |
| FinalResults | This is the FinalResults output object from AutoTransformationCreate(). |
| Type | Set to "Inverse" to back-transfrom or "Apply" for applying the transformation. |
| TransID | Set to a character value that corresponds with your modeling project |
| Path | Set to the directly where you want to save all of your modeling files |

## Value

data with transformed columns

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create Fake Data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 2L,
  AddDate = FALSE,
  Classification = FALSE,
```

```
  MultiClass = FALSE)

# Columns to transform
Cols <- names(data)[1L:11L]
print(Cols)

data <- data[1]

# Run function
Output <- AutoQuant::AutoTransformationCreate(
  data,
  ColumnNames = Cols,
 Methods = c("YeoJohnson", "BoxCox", "Asinh", "Log", "LogPlus1", "Sqrt", "Asin", "Logit", "Identity"),
  Path = getwd(),
  TransID = "Model_1",
  SaveOutput = TRUE)

# Output
data <- Output$Data
TransInfo <- Output$FinalResults

# Back Transform
data <- AutoQuant::AutoTransformationScore(
  data,
  FinalResults = TransInfo,
  Path = NULL,
  TransID = "Model_1")

## End(Not run)
```

---

AutoWord2VecModeler *AutoWord2VecModeler*

---

## Description

This function allows you to automatically build a word2vec model and merge the data onto your supplied dataset

## Usage

```
AutoWord2VecModeler(
  data,
  BuildType = "Combined",
  stringCol = c("Text_Col1", "Text_Col2"),
  KeepStringCol = FALSE,
  model_path = NULL,
  vects = 100,
  MinWords = 1,
  WindowSize = 12,
  Epochs = 25,
  SaveModel = "standard",
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G",
  ModelID = "Model_1"
)
```

## Arguments

| | |
|---|---|
| data | Source data table to merge vects onto |
| BuildType | Choose from "individual" or "combined". Individual will build a model for every text column. Combined will build a single model for all columns. |
| stringCol | A string name for the column to convert via word2vec |
| KeepStringCol | Set to TRUE if you want to keep the original string column that you convert via word2vec |
| model_path | A string path to the location where you want the model and metadata stored |
| vects | The number of vectors to retain from the word2vec model |
| MinWords | For H2O word2vec model |
| WindowSize | For H2O word2vec model |
| Epochs | For H2O word2vec model |
| SaveModel | Set to "standard" to save normally; set to "mojo" to save as mojo. NOTE: while you can save a mojo, I haven't figured out how to score it in the AutoH20Scoring function. |
| Threads | Number of available threads you want to dedicate to model building |
| MaxMemory | Amount of memory you want to dedicate to model building |
| ModelID | Name for saving to file |

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:

# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create Model and Vectors
```

```
data <- AutoQuant::AutoWord2VecModeler(
  data,
  BuildType = "individual",
  stringCol = c("Comment"),
  KeepStringCol = FALSE,
  ModelID = "Model_1",
  model_path = getwd(),
  vects = 10,
  MinWords = 1,
  WindowSize = 1,
  Epochs = 25,
  SaveModel = "standard",
  Threads = max(1,parallel::detectCores()-2),
  MaxMemory = "28G")

# Remove data
rm(data)

# Create fake data for mock scoring
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Give h2o a few seconds
Sys.sleep(5L)

# Create vectors for scoring
data <- AutoQuant::AutoWord2VecScoring(
  data,
  BuildType = 'individual',
  ModelObject = NULL,
  ModelID = "Model_1",
  model_path = getwd(),
  stringCol = "Comment",
  KeepStringCol = FALSE,
  H2OStartUp = TRUE,
  H2OShutdown = TRUE,
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G")

## End(Not run)
```

---

AutoWord2VecScoring *AutoWord2VecScoring*

---

## Description

AutoWord2VecScoring is for scoring models generated by AutoWord2VecModeler()

## Usage

```
AutoWord2VecScoring(
  data,
  BuildType = "individual",
  ModelObject = NULL,
  ModelID = "Model_1",
  model_path = NULL,
  stringCol = NULL,
  KeepStringCol = FALSE,
  H2OStartUp = TRUE,
  H2OShutdown = TRUE,
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G"
)
```

## Arguments

| | |
|---|---|
| data | data.table |
| BuildType | "individual" or "combined". Used to locate model in file |
| ModelObject | NULL if you want it loaded in the function |
| ModelID | Same as in training |
| model_path | Location of model |
| stringCol | Columns to transform |
| KeepStringCol | FALSE to remove string col after creating vectors |
| H2OStartUp | = TRUE, |
| Threads | max(1L, parallel::detectCores() - 2L) |
| MaxMemory | "28G" |

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
```

```
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create Model and Vectors
data <- AutoQuant::AutoWord2VecModeler(
  data,
  BuildType = "individual",
  stringCol = c("Comment"),
  KeepStringCol = FALSE,
  ModelID = "Model_1",
  model_path = getwd(),
  vects = 10,
  MinWords = 1,
  WindowSize = 1,
  Epochs = 25,
  SaveModel = "standard",
  Threads = max(1,parallel::detectCores()-2),
  MaxMemory = "28G")

# Remove data
rm(data)

# Create fake data for mock scoring
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = TRUE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create vectors for scoring
data <- AutoQuant::AutoWord2VecScoring(
  data,
  BuildType = "individual",
  ModelObject = NULL,
  ModelID = "Model_1",
  model_path = getwd(),
  stringCol = "Comment",
  KeepStringCol = FALSE,
  H2OStartUp = TRUE,
  H2OShutdown = TRUE,
  Threads = max(1L, parallel::detectCores() - 2L),
  MaxMemory = "28G")
```

```
## End(Not run)
```

---

`BuildBinary` *BuildBinary*

---

## Description

Build package binary

## Usage

```
BuildBinary(Root = NULL)
```

## Arguments

| | |
|---|---|
| `Root` | NULL will setwd to project root as defined in function |

## Author(s)

Adrian Antico

## See Also

Other Utilities: `Install()`, `UpdateDocs()`

---

`CategoricalEncoding` *CategoricalEncoding*

---

## Description

Categorical encoding for factor and character columns

## Usage

```
CategoricalEncoding(
  data = NULL,
  ML_Type = "classification",
  GroupVariables = NULL,
  TargetVariable = NULL,
  Method = NULL,
  SavePath = NULL,
  Scoring = FALSE,
  ImputeValueScoring = NULL,
  ReturnFactorLevelList = TRUE,
  SupplyFactorLevelList = NULL,
  KeepOriginalFactors = TRUE,
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | Source data.table |
| ML_Type | Only use with Method "credibility'. Select from 'classification' or 'regression'. |
| GroupVariables | Columns to encode |
| Method | Method to utilize. Choose from 'credibility', 'target_encoding', 'woe', 'm_estimator', 'poly_encode', 'backward_difference', 'helmert'. Default is 'credibility' which is more specifically, Bulhmann Credibility |
| SavePath | Path to save artifacts for recreating in scoring environments |
| Scoring | Set to TRUE for scoring mode. |
| ImputeValueScoring | If levels cannot be matched on scoring data you can supply a value to impute the NA's. Otherwise, leave NULL and manage them outside the function |
| ReturnFactorLevelList | TRUE by default. Returns a list of the factor variable and transformations needed for regenerating them in a scoring environment. Alternatively, if you save them to file, they can be called for use in a scoring environment. |
| SupplyFactorLevelList | The FactorCompenents list that gets returned. Supply this to recreate features in scoring environment |
| KeepOriginalFactors | Defaults to TRUE. Set to FALSE to remove the original factor columns |
| Debug | = FALSE |
| TargetVariabl | Target column name |

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create fake data with 10 categorical
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 1000000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 10L,
  AddDate = FALSE,
  Classification = TRUE,
  MultiClass = FALSE)

# Take your pick
```

```
Meth <- c('m_estimator',
          'credibility',
          'woe',
          'target_encoding',
          'poly_encode',
          'backward_difference',
          'helmert')

# Pass to function
MethNum <- 1

# Mock test data with same factor levels
test <- data.table::copy(data)

# Run in Train Mode
data <- AutoQuant::CategoricalEncoding(
  data = data,
  ML_Type = "classification",
  GroupVariables = paste0("Factor_", 1:10),
  TargetVariable = "Adrian",
  Method = Meth[MethNum],
  SavePath = getwd(),
  Scoring = FALSE,
  ReturnFactorLevelList = FALSE,
  SupplyFactorLevelList = NULL,
  KeepOriginalFactors = FALSE,
  Debug = FALSE)

# View results
print(data)

# Run in Score Mode by pulling in the csv's
test <- AutoQuant::CategoricalEncoding(
  data = data,
  ML_Type = "classification",
  GroupVariables = paste0("Factor_", 1:10),
  TargetVariable = "Adrian",
  Method = Meth[MethNum],
  SavePath = getwd(),
  Scoring = TRUE,
  ImputeValueScoring = 222,
  ReturnFactorLevelList = FALSE,
  SupplyFactorLevelList = NULL,
  KeepOriginalFactors = FALSE,
  Debug = FALSE)

## End(Not run)
```

---

`CreateCalendarVariables` *CreateCalendarVariables*

---

## Description

CreateCalendarVariables Rapidly creates calendar variables based on the date column you provide

## Usage

```
CreateCalendarVariables(
  data,
  DateCols = NULL,
  AsFactor = FALSE,
  TimeUnits = "wday",
  CachePath = NULL,
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | This is your data |
| DateCols | Supply either column names or column numbers of your date columns you want to use for creating calendar variables |
| AsFactor | Set to TRUE if you want factor type columns returned; otherwise integer type columns will be returned |
| TimeUnits | Supply a character vector of time units for creating calendar variables. Options include: "second", "minute", "hour", "wday", "mday", "yday", "week", "isoweek", "wom" (week of month), "month", "quarter", "year" |
| CachePath | Path to data in a local directory. .csv only for now |
| Debug | = FALSE |

## Value

Returns your data.table with the added calendar variables at the end

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create fake data with a Date column----
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.75,
  N = 25000L,
  ID = 2L,
  ZIP = 0L,
  FactorCount = 4L,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)
```

```
for(i in seq_len(20L)) {
  print(i)
  data <- data.table::rbindlist(
    list(data, AutoQuant::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 2L,
    ZIP = 0L,
    FactorCount = 4L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)))
}

# Create calendar variables - automatically excludes
#   the second, minute, and hour selections since
#   it is not timestamp data
runtime <- system.time(
  data <- AutoQuant::CreateCalendarVariables(
    data = data,
    DateCols = "DateTime",
    AsFactor = FALSE,
    TimeUnits = c("second",
                  "minute",
                  "hour",
                  "wday",
                  "mday",
                  "yday",
                  "week",
                  "isoweek",
                  "wom",
                  "month",
                  "quarter",
                  "year")))
head(data)
print(runtime)

## End(Not run)
```

---

`CreateHolidayVariables` *CreateHolidayVariables*

---

## Description

CreateHolidayVariables Rapidly creates holiday count variables based on the date columns you provide

## Usage

```
CreateHolidayVariables(
  data,
  DateCols = NULL,
  LookbackDays = NULL,
```

```
  HolidayGroups = c("USPublicHolidays", "EasterGroup", "ChristmasGroup",
    "OtherEcclesticalFeasts"),
  Holidays = NULL,
  Print = FALSE,
  CachePath = NULL,
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | This is your data |
| DateCols | Supply either column names or column numbers of your date columns you want to use for creating calendar variables |
| LookbackDays | Default NULL which investigates Date - Lag1Date to compute Holiday's per period. Otherwise it will lookback LokkbackDays. |
| HolidayGroups | Pick groups |
| Holidays | Pick holidays |
| Print | Set to TRUE to print iteration number to console |
| CachePath | = NULL |
| Debug | = FALSE |

## Value

Returns your data.table with the added holiday indicator variable

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create fake data with a Date----
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.75,
  N = 25000L,
  ID = 2L,
  ZIP = 0L,
  FactorCount = 4L,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)
for(i in seq_len(20L)) {
  print(i)
  data <- data.table::rbindlist(list(data,
```

```
  AutoQuant::FakeDataGenerator(
    Correlation = 0.75,
    N = 25000L,
    ID = 2L,
    ZIP = 0L,
    FactorCount = 4L,
    AddDate = TRUE,
    Classification = FALSE,
    MultiClass = FALSE)))
}
# Run function and time it
runtime <- system.time(
  data <- AutoQuant::CreateHolidayVariables(
    data,
    DateCols = "DateTime",
    LookbackDays = NULL,
    HolidayGroups = c("USPublicHolidays","EasterGroup",
      "ChristmasGroup","OtherEcclesticalFeasts"),
    Holidays = NULL,
    Print = FALSE))
head(data)
print(runtime)

## End(Not run)
```

---

DummifyDT *DummifyDT*

---

## Description

DummifyDT creates dummy variables for the selected columns. Either one-hot encoding, N+1 columns for N levels, or N columns for N levels.

## Usage

```
DummifyDT(
  data,
  cols,
  TopN = NULL,
  KeepFactorCols = FALSE,
  OneHot = FALSE,
  SaveFactorLevels = FALSE,
  SavePath = NULL,
  ImportFactorLevels = FALSE,
  FactorLevelsList = NULL,
  ClustScore = FALSE,
  ReturnFactorLevels = FALSE,
  GroupVar = FALSE
)
```

## Arguments

| | |
|---|---|
| data | The data set to run the micro auc on |

| | |
|---|---|
| cols | A vector with the names of the columns you wish to dichotomize |
| TopN | Default is NULL. Scalar to apply to all categorical columns or a vector to apply to each categorical variable. Only create dummy variables for the TopN number of levels. Will be either TopN or max(levels) |
| KeepFactorCols | Set to TRUE to keep the original columns used in the dichotomization process |
| OneHot | Set to TRUE to run one hot encoding, FALSE to generate N columns for N levels |
| SaveFactorLevels | Set to TRUE to save unique levels of each factor column to file as a csv |
| SavePath | Provide a file path to save your factor levels. Use this for models that you have to create dummy variables for. |
| ImportFactorLevels | Instead of using the data you provide, import the factor levels csv to ensure you build out all of the columns you trained with in modeling. |
| FactorLevelsList | Supply a list of factor variable levels |
| ClustScore | This is for scoring AutoKMeans. It converts the added dummy column names to conform with H2O dummy variable naming convention |
| ReturnFactorLevels | If you want a named list of all the factor levels returned, set this to TRUE. Doing so will cause the function to return a list with the source data.table and the list of factor variables' levels |
| GroupVar | Ignore this |

## Value

Either a data table with new dummy variables columns and optionally removes base columns (if ReturnFactorLevels is FALSE), otherwise a list with the data.table and a list of the factor levels.

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 10L,
```

```
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Create dummy variables
data <- AutoQuant::DummifyDT(
  data = data,
  cols = c("Factor_1",
           "Factor_2",
           "Factor_3",
           "Factor_4",
           "Factor_5",
           "Factor_6",
           "Factor_8",
           "Factor_9",
           "Factor_10"),
  TopN = c(rep(3,9)),
  KeepFactorCols = TRUE,
  OneHot = FALSE,
  SaveFactorLevels = TRUE,
  SavePath = getwd(),
  ImportFactorLevels = FALSE,
  FactorLevelsList = NULL,
  ClustScore = FALSE,
  ReturnFactorLevels = FALSE)

# Create Fake Data for Scoring Replication
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.85,
  N = 25000,
  ID = 2L,
  ZIP = 0,
  FactorCount = 10L,
  AddDate = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Scoring Version
data <- AutoQuant::DummifyDT(
  data = data,
  cols = c("Factor_1",
           "Factor_2",
           "Factor_3",
           "Factor_4",
           "Factor_5",
           "Factor_6",
           "Factor_8",
           "Factor_9",
           "Factor_10"),
  TopN = c(rep(3,9)),
  KeepFactorCols = TRUE,
  OneHot = FALSE,
  SaveFactorLevels = TRUE,
  SavePath = getwd(),
  ImportFactorLevels = TRUE,
  FactorLevelsList = NULL,
  ClustScore = FALSE,
```

```
  ReturnFactorLevels = FALSE)

## End(Not run)
```

---

H2OAutoencoder *H2OAutoencoder*

---

## Description

H2OAutoencoder for anomaly detection and or dimensionality reduction

## Usage

```
H2OAutoencoder(
  AnomalyDetection = FALSE,
  DimensionReduction = TRUE,
  data,
  Features = NULL,
  RemoveFeatures = FALSE,
  NThreads = max(1L, parallel::detectCores() - 2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  model_path = NULL,
  LayerStructure = NULL,
  NodeShrinkRate = (sqrt(5) - 1)/2,
  ReturnLayer = 4L,
  ReturnFactorCount = NULL,
  per_feature = TRUE,
  Activation = "Tanh",
  Epochs = 5L,
  L2 = 0.1,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.9,
  ElasticAveragingRegularization = 0.001
)
```

## Arguments

| | |
|---|---|
| AnomalyDetection | Set to TRUE to run anomaly detection |
| DimensionReduction | Set to TRUE to run dimension reduction |
| data | The data.table with the columns you wish to have analyzed |
| Features | NULL Column numbers or column names |
| RemoveFeatures | Set to TRUE if you want the features you specify in the Features argument to be removed from the data returned |
| NThreads | max(1L, parallel::detectCores()-2L) |
| MaxMem | "28G" |

| | |
|---|---|
| H2OStart | TRUE to start H2O inside the function |
| H2OShutdown | Setting to TRUE will shutdown H2O when it done being used internally. |
| ModelID | "TestModel" |
| model_path | If NULL no model will be saved. If a valid path is supplied the model will be saved there |
| LayerStructure | If NULL, layers and sizes will be created for you, using NodeShrinkRate and 7 layers will be created. |
| NodeShrinkRate | = (sqrt(5) - 1) / 2, |
| ReturnLayer | Which layer of the NNet to return. Choose from 1-7 with 4 being the layer with the least amount of nodes |
| ReturnFactorCount | Default is NULL. If you supply a number, the final layer will be that number. Otherwise, it will be based on the NodeShrinkRate math. |
| per_feature | Set to TRUE to have per feature anomaly detection generated. Otherwise and overall value will be generated |
| Activation | Choose from "Tanh", "TanhWithDropout", "Rectifier", "RectifierWithDropout","Maxout", "MaxoutWithDropout" |
| Epochs | Quantile value to find the cutoff value for classifying outliers |
| L2 | Specify the amount of memory to allocate to H2O. E.g. "28G" |
| ElasticAveraging | Specify the number of threads (E.g. cores * 2) |
| ElasticAveragingMovingRate | Specify the number of decision trees to build |
| ElasticAveragingRegularization | Specify the row sample rate per tree |

## Value

A data.table

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore( AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
############################
# Training
############################

# Create simulated data
```

```
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
Output <- AutoQuant::H2OAutoencoder(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  Features = names(data)[2L:(ncol(data)-1L)],
  per_feature = FALSE,
  RemoveFeatures = FALSE,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O Environment
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,

  # H2O ML Args
  LayerStructure = NULL,
  NodeShrinkRate = (sqrt(5) - 1) / 2,
  ReturnLayer = 4L,
  ReturnFactorCount = NULL,
  Activation = "Tanh",
  Epochs = 5L,
  L2 = 0.10,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.90,
  ElasticAveragingRegularization = 0.001)

# Inspect output
data <- Output$Data
Model <- Output$Model

# If ValidationData is not null
ValidationData <- Output$ValidationData

############################
# Scoring
############################
```

```
# Create simulated data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- AutoQuant::H2OAutoencoderScoring(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  Features = names(data)[2L:ncol(data)],
  RemoveFeatures = TRUE,
  per_feature = FALSE,
  ModelObject = NULL,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O args
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ReturnLayer = 4L)

## End(Not run)
```

---

H2OAutoencoderScoring *H2OAutoencoderScoring*

---

## Description

H2OAutoencoderScoring for anomaly detection and or dimensionality reduction

## Usage

```
H2OAutoencoderScoring(
  data,
  Features = NULL,
  RemoveFeatures = FALSE,
  ModelObject = NULL,
  AnomalyDetection = TRUE,
```

```
  DimensionReduction = TRUE,
  ReturnLayer = 4L,
  per_feature = TRUE,
  NThreads = max(1L, parallel::detectCores() - 2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  model_path = NULL
)
```

## Arguments

| | |
|---|---|
| data | The data.table with the columns you wish to have analyzed |
| Features | NULL Column numbers or column names |
| RemoveFeatures | Set to TRUE if you want the features you specify in the Features argument to be removed from the data returned |
| ModelObject | If NULL then the model will be loaded from file. Otherwise, it will use what is supplied |
| AnomalyDetection | Set to TRUE to run anomaly detection |
| DimensionReduction | Set to TRUE to run dimension reduction |
| ReturnLayer | Which layer of the NNet to return. Choose from 1-7 with 4 being the layer with the least amount of nodes |
| per_feature | Set to TRUE to have per feature anomaly detection generated. Otherwise and overall value will be generated |
| NThreads | max(1L, parallel::detectCores()-2L) |
| MaxMem | "28G" |
| H2OStart | TRUE to start H2O inside the function |
| H2OShutdown | Setting to TRUE will shutdown H2O when it done being used internally. |
| ModelID | "TestModel" |
| model_path | If NULL no model will be saved. If a valid path is supplied the model will be saved there |

## Value

A data.table

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
############################
# Training
############################

# Create simulated data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- AutoQuant::H2OAutoencoder(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  ValidationData = NULL,
  Features = names(data)[2L:(ncol(data)-1L)],
  per_feature = FALSE,
  RemoveFeatures = TRUE,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O Environment
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,

  # H2O ML Args
  LayerStructure = NULL,
  ReturnLayer = 4L,
  Activation = "Tanh",
  Epochs = 5L,
  L2 = 0.10,
  ElasticAveraging = TRUE,
  ElasticAveragingMovingRate = 0.90,
  ElasticAveragingRegularization = 0.001)

############################
# Scoring
############################
```

```
# Create simulated data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 1000L,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  AddComment = FALSE,
  ZIP = 2L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- AutoQuant::H2OAutoencoderScoring(

  # Select the service
  AnomalyDetection = TRUE,
  DimensionReduction = TRUE,

  # Data related args
  data = data,
  Features = names(data)[2L:ncol(data)],
  RemoveFeatures = TRUE,
  per_feature = FALSE,
  ModelObject = NULL,
  ModelID = "TestModel",
  model_path = getwd(),

  # H2O args
  NThreads = max(1L, parallel::detectCores()-2L),
  MaxMem = "28G",
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ReturnLayer = 4L)

## End(Not run)
```

---

`H2OIsolationForest` *H2OIsolationForest*

---

## Description

H2OIsolationForestScoring for dimensionality reduction and / or anomaly detection

## Usage

```
H2OIsolationForest(
  data,
  Features = NULL,
  IDcols = NULL,
  ModelID = "TestModel",
  SavePath = NULL,
```

```
  Threshold = 0.975,
  MaxMem = "28G",
  NThreads = -1,
  NTrees = 100,
  MaxDepth = 8,
  MinRows = 1,
  RowSampleRate = (sqrt(5) - 1)/2,
  ColSampleRate = 1,
  ColSampleRatePerLevel = 1,
  ColSampleRatePerTree = 1,
  CategoricalEncoding = c("AUTO"),
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | The data.table with the columns you wish to have analyzed |
| Features | A character vector with the column names to utilize in the isolation forest |
| IDcols | A character vector with the column names to not utilize in the isolation forest but have returned with the data output. Otherwise those columns will be removed |
| ModelID | Name for model that gets saved to file if SavePath is supplied and valid |
| SavePath | Path directory to store saved model |
| Threshold | Quantile value to find the cutoff value for classifying outliers |
| MaxMem | Specify the amount of memory to allocate to H2O. E.g. "28G" |
| NThreads | Specify the number of threads (E.g. cores * 2) |
| NTrees | Specify the number of decision trees to build |
| MaxDepth | Max tree depth |
| MinRows | Minimum number of rows allowed per leaf |
| RowSampleRate | Number of rows to sample per tree |
| ColSampleRate | Sample rate for each split |
| ColSampleRatePerLevel | Sample rate for each level |
| ColSampleRatePerTree | Sample rate per tree |
| CategoricalEncoding | Choose from "AUTO", "Enum", "OneHotInternal", "OneHotExplicit", "Binary", "Eigen", "LabelEncoder", "SortByResponse", "EnumLimited" |
| Debug | Debugging |

## Value

Source data.table with predictions. Note that any columns not listed in Features nor IDcols will not be returned with data. If you want columns returned but not modeled, supply them as IDcols

## Author(s)

Adrian Antico

## See Also

Other Unsupervised Learning: AutoClusteringScoring(), AutoClustering(), H2OIsolationForestScoring()

## Examples

```
## Not run:
# Create simulated data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- AutoQuant::H2OIsolationForest(
  data,
  Features = names(data)[2L:ncol(data)],
  IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
  ModelID = "Adrian",
  SavePath = getwd(),
  Threshold = 0.95,
  MaxMem = "28G",
  NThreads = -1,
  NTrees = 100,
  MaxDepth = 8,
  MinRows = 1,
  RowSampleRate = (sqrt(5)-1)/2,
  ColSampleRate = 1,
  ColSampleRatePerLevel = 1,
  ColSampleRatePerTree = 1,
  CategoricalEncoding = c("AUTO"),
  Debug = TRUE)

# Remove output from data and then score
data[, eval(names(data)[17:ncol(data)]) := NULL]

# Run algo
Outliers <- AutoQuant::H2OIsolationForestScoring(
  data,
  Features = names(data)[2:ncol(data)],
  IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  SavePath = getwd(),
  Threshold = 0.95,
  MaxMem = "28G",
  NThreads = -1,
  Debug = FALSE)
```

```
## End(Not run)
```

---

H2OIsolationForestScoring
*H2OIsolationForestScoring*

---

## Description

H2OIsolationForestScoring for dimensionality reduction and / or anomaly detection scoring on new data

## Usage

```
H2OIsolationForestScoring(
  data,
  Features = NULL,
  IDcols = NULL,
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  SavePath = NULL,
  Threshold = 0.975,
  MaxMem = "28G",
  NThreads = -1,
  Debug = FALSE
)
```

## Arguments

| | |
|---|---|
| data | The data.table with the columns you wish to have analyzed |
| Features | A character vector with the column names to utilize in the isolation forest |
| IDcols | A character vector with the column names to not utilize in the isolation forest but have returned with the data output. Otherwise those columns will be removed |
| H2OStart | TRUE to have H2O started inside function |
| H2OShutdown | TRUE to shutdown H2O inside function |
| ModelID | Name for model that gets saved to file if SavePath is supplied and valid |
| SavePath | Path directory to store saved model |
| Threshold | Quantile value to find the cutoff value for classifying outliers |
| MaxMem | Specify the amount of memory to allocate to H2O. E.g. "28G" |
| NThreads | Specify the number of threads (E.g. cores * 2) |
| Debug | Debugging |

## Value

Source data.table with predictions. Note that any columns not listed in Features nor IDcols will not be returned with data. If you want columns returned but not modeled, supply them as IDcols

## Author(s)

Adrian Antico

## See Also

Other Unsupervised Learning: AutoClusteringScoring(), AutoClustering(), H2OIsolationForest()

## Examples

```
## Not run:
# Create simulated data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.70,
  N = 50000,
  ID = 2L,
  FactorCount = 2L,
  AddDate = TRUE,
  ZIP = 0L,
  TimeSeries = FALSE,
  ChainLadderData = FALSE,
  Classification = FALSE,
  MultiClass = FALSE)

# Run algo
data <- AutoQuant::H2OIsolationForest(
  data,
  Features = names(data)[2L:ncol(data)],
  IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
  ModelID = "Adrian",
  SavePath = getwd(),
  Threshold = 0.95,
  MaxMem = "28G",
  NThreads = -1,
  NTrees = 100,
  SampleRate = (sqrt(5)-1)/2,
  MaxDepth = 8,
  MinRows = 1,
  ColSampleRate = 1,
  ColSampleRatePerLevel = 1,
  ColSampleRatePerTree = 1,
  CategoricalEncoding = c("AUTO"),
  Debug = TRUE)

# Remove output from data and then score
data[, eval(names(data)[17:ncol(data)]) := NULL]

# Run algo
Outliers <- AutoQuant::H2OIsolationForestScoring(
  data,
  Features = names(data)[2:ncol(data)],
  IDcols = c("Adrian", "IDcol_1", "IDcol_2"),
  H2OStart = TRUE,
  H2OShutdown = TRUE,
  ModelID = "TestModel",
  SavePath = getwd(),
  Threshold = 0.95,
```

```
  MaxMem = "28G",
  NThreads = -1,
  Debug = FALSE)

## End(Not run)
```

---

`hello` *Hello, World!*

---

## Description

Prints 'Hello, world!'.

## Usage

```
hello()
```

## Examples

```
hello()
```

---

`Install` *Install*

---

## Description

To install the package

## Usage

```
Install(Root = NULL)
```

## Arguments

| | |
|---|---|
| `Root` | NULL will setwd to project root as defined in function |

## Author(s)

Adrian Antico

## See Also

Other Utilities: `BuildBinary()`, `UpdateDocs()`

## Mode *Mode*

### Description

Statistical mode. Only returns the first mode if there are many

### Usage

```
Mode(x)
```

### Arguments

| | |
|---|---|
| x | vector |

### Author(s)

Adrian Antico

## ModelDataPrep *ModelDataPrep*

### Description

This function replaces inf values with NA, converts characters to factors, and imputes with constants

### Usage

```
ModelDataPrep(
  data,
  Impute = TRUE,
  CharToFactor = TRUE,
  FactorToChar = FALSE,
  IntToNumeric = TRUE,
  LogicalToBinary = FALSE,
  DateToChar = FALSE,
  IDateConversion = FALSE,
  RemoveDates = FALSE,
  MissFactor = "0",
  MissNum = -1,
  IgnoreCols = NULL
)
```

## Arguments

| | |
|---|---|
| data | This is your source data you'd like to modify |
| Impute | Defaults to TRUE which tells the function to impute the data |
| CharToFactor | Defaults to TRUE which tells the function to convert characters to factors |
| FactorToChar | Converts to character |
| IntToNumeric | Defaults to TRUE which tells the function to convert integers to numeric |
| LogicalToBinary | Converts logical values to binary numeric values |
| DateToChar | Converts date columns into character columns |
| IDateConversion | Convert IDateTime to POSIXct and IDate to Date types |
| RemoveDates | Defaults to FALSE. Set to TRUE to remove date columns from your data.table |
| MissFactor | Supply the value to impute missing factor levels |
| MissNum | Supply the value to impute missing numeric values |
| IgnoreCols | Supply column numbers for columns you want the function to ignore |

## Value

Returns the original data table with corrected values

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
# Create fake data
data <- AutoQuant::FakeDataGenerator(
  Correlation = 0.75,
  N = 250000L,
  ID = 2L,
  ZIP = 0L,
  FactorCount = 6L,
  AddDate = TRUE,
  Classification = FALSE,
  MultiClass = FALSE)

# Check column types
str(data)

# Convert some factors to character
data <- AutoQuant::ModelDataPrep(
```

```
  data,
  Impute       = TRUE,
  CharToFactor = FALSE,
  FactorToChar = TRUE,
  IntToNumeric = TRUE,
  LogicalToBinary = FALSE,
  DateToChar   = FALSE,
  IDateConversion = FALSE,
  RemoveDates  = TRUE,
  MissFactor   = "0",
  MissNum      = -1,
  IgnoreCols   = c("Factor_1"))

# Check column types
str(data)

## End(Not run)
```

---

PercRank *PercRank*

---

## Description

Generate percent ranks for multiple variables, by groups if provided, and with a selected granularity

## Usage

```
PercRank(
  data,
  ColNames,
  GroupVars = NULL,
  Granularity = 0.001,
  ScoreTable = FALSE
)
```

## Arguments

| | |
|---|---|
| data | Source data.table |
| ColNames | Character vector of column names |
| GroupVars | Character vector of column names to have percent ranks by the group levels |
| Granularity | Provide a value such that data.table::frank(Variable) * (1 / Granularity) / .N * Granularity. Default is 0.001 |
| ScoreTable | = FALSE. Set to TRUE to get the reference values for applying to new data. Pass to scoring version of this function |

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
data <- data.table::fread(file.choose())
x <- PercRank(data, ColNames = c('Weekly_Sales', 'XREG1'), GroupVars = c('Region','Store','Dept'), Granularity

## End(Not run)
```

---

PercRankScoring *PercRankScoring*

---

## Description

Generate percent ranks for multiple variables, by groups if provided, and with a selected granularity, via list passed from PercRank

## Usage

```
PercRankScoring(data, ScoreTable, GroupVars = NULL, RollDirection = "forward")
```

## Arguments

| | |
|---|---|
| data | Source data.table |
| ScoreTable | list of values returned from PercRank |
| GroupVars | Character vector of column names to have percent ranks by the group levels |
| RollDirection | "forward" or "backward" |

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

Standardize *Standardize*

## Description

Generate standardized values for multiple variables, by groups if provided, and with a selected granularity

## Usage

```
Standardize(
  data,
  ColNames,
  GroupVars = NULL,
  Center = TRUE,
  Scale = TRUE,
  ScoreTable = FALSE
)
```

## Arguments

| | |
|---|---|
| data | Source data.table |
| ColNames | Character vector of column names |
| GroupVars | Character vector of column names to have percent ranks by the group levels |
| Center | TRUE |
| Scale | TRUE |
| ScoreTable | FALSE. Set to TRUE to return a data.table that can be used to apply or back-transform via StandardizeScoring |

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
data <- data.table::fread(file.choose())
x <- Standardize(data = data, ColNames = c('Weekly_Sales', 'XREG3'), GroupVars = c('Region','Store','Dept'), Ce

## End(Not run)
```

StandardizeScoring *StandardizeScoring*

## Description

Generate standardized values for multiple variables, by groups if provided, and with a selected granularity

## Usage

```
StandardizeScoring(data, ScoreTable, Apply = "apply", GroupVars = NULL)
```

## Arguments

| | |
|---|---|
| data | Source data.table |
| Apply | 'apply' or 'backtransform' |
| GroupVars | Character vector of column names to have percent ranks by the group levels |
| ColNames | Character vector of column names |
| Center | TRUE |
| Scale | TRUE |

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), Standardize(), TimeSeriesFillRoll(), TimeSeriesFill()

## Examples

```
## Not run:
x <- Standardize(data = data, ColNames = c('Weekly_Sales', 'XREG1'), GroupVars = c('Region','Store','Dept'), Ce

## End(Not run)
```

`TimeSeriesFill` *TimeSeriesFill*

## Description

TimeSeriesFill For Completing Time Series Data For Single Series or Time Series by Group

## Usage

```
TimeSeriesFill(
  data = NULL,
  TargetColumn = NULL,
  DateColumnName = NULL,
  GroupVariables = NULL,
  TimeUnit = "days",
  FillType = "maxmax",
  MaxMissingPercent = 0.05,
  SimpleImpute = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | Supply your full series data set here |
| `TargetColumn` | = NULL |
| `DateColumnName` | Supply the name of your date column |
| `GroupVariables` | Supply the column names of your group variables. E.g. "Group" or c("Group1","Group2") |
| `TimeUnit` | Choose from "second", "minute", "hour", "day", "week", "month", "quarter", "year" |
| `FillType` | Choose from maxmax - Fill from the absolute min date to the absolute max date, minmax - Fill from the max date of the min set to the absolute max date, maxmin - Fill from the absolute min date to the min of the max dates, or minmin - Fill from the max date of the min dates to the min date of the max dates |
| `MaxMissingPercent` | The maximum amount of missing values an individual series can have to remain and be imputed. Otherwise, they are discarded. |
| `SimpleImpute` | Set to TRUE or FALSE. With TRUE numeric cols will fill NAs with a 0 and non-numeric cols with a "0" |

## Value

Returns a data table with missing time series records filled (currently just zeros)

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFillRoll()

## Examples

```
## Not run:

# Pull in data
data <- data.table::fread("https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

# Run function
data <- TimeSeriesFill(
  data,
  DateColumnName = "Date",
  GroupVariables = c("Store","Dept"),
  TimeUnit = "weeks",
  FillType = "maxmax",
  SimpleImpute = FALSE)

# data <- data.table::fread("https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")
# DateColumnName = "Date"
# GroupVariables = c("Store","Dept")
# TimeUnit = "weeks"
# FillType = "maxmax" # "minmin" # "maxmin" # "dynamic:method" # "minmax" #
# SimpleImpute = FALSE


## End(Not run)
```

---

# TimeSeriesFillRoll *TimeSeriesFillRoll*

---

## Description

TimeSeriesFillRoll For Completing Time Series Data For Single Series or Time Series by Group

## Usage

```
TimeSeriesFillRoll(
  data = NULL,
  DateColumnName = NULL,
  RollVars = NULL,
  NonRollVars = NULL,
  GroupVariables = NULL,
  RollDirection = "backward",
  TimeUnit = "days",
  SimpleImpute = FALSE
)
```

## Arguments

| | |
|---|---|
| data | Supply your full series data set here |
| DateColumnName | Supply the name of your date column |
| RollVars | = NULL, |
| NonRollVars | = NULL, |
| GroupVariables | Supply the column names of your group variables. E.g. "Group" or c("Group1","Group2") |
| RollDirection | 'backward' or 'forward' |
| TimeUnit | Choose from "second", "minute", "hour", "day", "week", "month", "quarter", "year" |
| SimpleImpute | Set to TRUE or FALSE. With TRUE numeric cols will fill NAs with a 0 and non-numeric cols with a "0" |

## Value

Returns a data table with missing time series records filled (currently just zeros)

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: AutoDataPartition(), AutoDiffLagN(), AutoInteraction(), AutoLagRollMode(), AutoLagRollStatsScoring(), AutoLagRollStats(), AutoTransformationCreate(), AutoTransformationScore(), AutoWord2VecModeler(), AutoWord2VecScoring(), CategoricalEncoding(), CreateCalendarVariables(), CreateHolidayVariables(), DummifyDT(), H2OAutoencoderScoring(), H2OAutoencoder(), ModelDataPrep(), PercRankScoring(), PercRank(), StandardizeScoring(), Standardize(), TimeSeriesFill()

## Examples

```
## Not run:

# Pull in data
data <- data <- data.table::fread("https://www.dropbox.com/s/2str3ek4f4cheqi/walmart_train.csv?dl=1")

# Run function
data <- TimeSeriesFillRoll(
  data,
  RollVars = c('Net_Revenue', 'Units', 'SIZE_UNITS', 'Liters', 'Accum_Units'),
 NonRollVars = c('Diff_1_DATE_ISO','Net_Revenue_PerDay','Liters_PerDay','Units_PerDay'),
  DateColumnName = "Date",
  GroupVariables = c("Store","Dept"),
  RollDirection = 'backward',
  TimeUnit = "weeks",
  SimpleImpute = FALSE)

## End(Not run)
```

UpdateDocs *UpdateDocs*

## Description

Update helf files and reference manual

## Usage

```
UpdateDocs(BuildVignette = FALSE, Root = NULL)
```

## Author(s)

Adrian Antico

## See Also

Other Utilities: `BuildBinary()`, `Install()`

# Index