

# Package ‘RemixAutoML’

March 20, 2019

**Title** Remix Automated Machine Learning

**Version** 1.0

**Description** Automate and ensure high quality output for most of your machine learning and data science tasks. We have high quality functions that run at efficient speed with minimal memory constraints. The library contains functions for supervised learning, unsupervised learning, feature engineering, model evaluation and interpretation, along with some helper functions for graphing.

**Depends** R ( $\geq$  3.5.0)

**SystemRequirements** Java ( $\geq$  7.0)

**License** GPL-2

**Encoding** UTF-8

**Language** en-US

**URL** <https://github.com/AdrianAntico/RemixAutoML>

**BugReports** <https://github.com/AdrianAntico/RemixAutoML>

**Contact** Adrian Antico

**LazyData** true

**RoxygenNote** 6.1.1

**Imports** data.table, zoo, h2o, lubridate, ggplot2, caTools, forecast, prophet, tsoutliers, stringr, itertools, doParallel, parallel, scatterplot3d, RColorBrewer, grid, monreg, tm, wordcloud, foreach, pROC

**Suggests** testthat, sde, knitr, rmarkdown

**VignetteBuilder** knitr

**Author** Adrian Antico

**Maintainer** Adrian Antico <[adrianantico@gmail.com](mailto:adrianantico@gmail.com)>

**Date** '2019-03-20'

**Views** MachineLearning,  
AutomatedSupervisedLearning,  
SupervisedLearning,  
AutomatedUnsupervisedLearning,  
UnsupervisedLearning,  
Clustering,  
AnomalyDetection,

FeatureEngineering,  
 VariableCreation,  
 ModelEvaluation,  
 FeatureInterpretation,  
 VariableInterpretation,  
 VariableImportance,  
 AutomatedTimeSeriesForecasting,  
 TimeSeries,  
 Forecasting

## R topics documented:

AutoH20Modeler . . . . .	2
AutoTS . . . . .	9
ChartTheme . . . . .	10
DT_GDL_Feature_Engineering . . . . .	11
DummifyDT . . . . .	13
EvalPlot . . . . .	14
FAST_GDL_Feature_Engineering . . . . .	15
GDL_Feature_Engineering . . . . .	17
GenTSAnomVars . . . . .	18
GLRM_KMeans_Col . . . . .	20
ModelDataPrep . . . . .	21
multiplot . . . . .	22
nlsModelFit . . . . .	23
ParDepCalPlots . . . . .	24
percRank . . . . .	25
PrintObjectsSize . . . . .	26
RedYellowGreen . . . . .	27
RemixTheme . . . . .	28
ResidualOutliers . . . . .	29
Scoring_GDL_Feature_Engineering . . . . .	30
SimpleCap . . . . .	32
tempDatesFun . . . . .	32
threshOptim . . . . .	33
tokenizeH20 . . . . .	34
Word2VecModel . . . . .	35
WordFreq . . . . .	36
<b>Index</b>	<b>38</b>

---

AutoH20Modeler

*An Automated Machine Learning Framework using H20*

---

## Description

Steps in the function include: 1. Logic: Error checking in the modeling arguments from your Construction file 2. ML: Build grid-tuned models and baseline models for comparison and checks which one performs better on validation data 3. Evaluation: Collects the performance metrics for both 4. Evaluation: Generates calibration plots (and boxplots for regression) for the winning model 5. Evaluation: Generates partial dependence calibration

plots (and boxplots for regression) for the winning model 6. Evaluation: Generates variable importance tables and a table of non-important features 7. Production: Creates a storage file containing: model name, model path, grid tune performance, baseline performance, and threshold (if classification) and stores that file in your model\_path location

## Usage

```
AutoH20Modeler(Construct, max_memory, ratios, BL_Trees, nthreads,
  model_path, MaxRuntimeSeconds = 3600, MaxModels = 30,
  TrainData = NULL, TestData = NULL)
```

## Arguments

Construct	Core instruction file for automation (see Details below for more information on this)
max_memory	The ceiling amount of memory H20 will utilize
ratios	The percentage of train samples from source data (remainder goes to validation set)
BL_Trees	The number of trees to build in baseline GBM or RandomForest
nthreads	Set the number of threads to run function
model_path	Directory path for where you want your models saved
MaxRuntimeSeconds	Number of seconds of run time for grid tuning
MaxModels	Number of models you'd like to have returned
TrainData	Set to NULL or supply a data.table for training data
TestData	Set to NULL or supply a data.table for validation data

## Details

The Construct file must be a data.table and the columns need to be in the correct order (see examples). Character columns must be converted to type "Factor". You must remove date columns or convert them to "Factor". For classification models, your target variable needs to be a (0,1) of type "Factor." See the examples below for help with setting up the Construct file for various modeling target variable types. There are examples for regression, classification, multinomial, and quantile regression.

Let's go over the construct file, column by column. The Targets column is where you specify the column number of your target variable (in quotes, e.g. "c(1)"). The Distribution column is where you specify the distribution type for the modeling task. For classification use bernoulli, for multilabel use multinomial, for quantile use quantile, and for regression, you can choose from the list available in the H20 docs, such as gaussian, poisson, gamma, etc. It's not set up to handle tweedie distributions currently but I can add support if there is demand. The Loss column tells H20 which metric to use for the loss metrics. For regression, I typically use "mse", quantile regression, "mae", classification "auc", and multinomial "logloss". For deeplearning models, you need to use "quadratic", "absolute", and "crossentropy". The Quantile column tells H20 which quantile to use for quantile regression (in decimal form). The ModelName column is the name you wish to give your model as a prefix. The Algorithm column is the model you wish to use: gbm, randomForest, deeplearning, AutoML, XGBoost, LightGBM. The dataName column is the name of your data. The TargetCol column is the column number of your target variable. The FeatureCols column is the column numbers of your features. The CreateDate column is for tracking your

model build dates. The GridTune column is a TRUE / FALSE column for whether you want to run a grid tune model for comparison. The ExportValidData column is a TRUE / FALSE column indicating if you want to export the validation data. The ParDep column is where you put the number of partial dependence calibration plots you wish to generate. The PD\_Data column is where you specify if you want to generate the partial dependence plots on "All" data, "Validate" data, or "Train" data. The ThreshType column is for classification models. You can specify "f1", "f2", "f0point5", or "CS" for cost sensitive. The FSC column is the feature selection column. Specify the percentage importance cutoff to create a table of "unimportant" features. The tpProfit column is for when you specify "CS" in the ThreshType column. This is your true positive profit. The tnProfit column is for when you specify "CS" in the ThreshType column. This is your true negative profit. The fpProfit column is for when you specify "CS" in the ThreshType column. This is your false positive profit. The fnProfit column is for when you specify "CS" in the ThreshType column. This is your false negative profit. The SaveModel column is a TRUE / FALSE indicator. If you are just testing out models, set this to FALSE. The SaveModelType column is where you specify if you want a "standard" model object saved or a "mojo" model object saved. The PredsAllData column is a TRUE / FALSE column. Set to TRUE if you want all the predicted values returns (for all data). The TargetEncoding column let's you specify the column number of features you wish to run target encoding on. Set to NA to not run this feature. The SupplyData column lets you supply the data names for training and validation data. Set to NULL if you want the data partitioning to be done internally.

## Value

Returns saved models, corrected Construct file, variable importance tables, evaluation and partial dependence calibration plots, model performance measure, etc.

## Author(s)

Adrian Antico

## See Also

Other Supervised Learning: [AutoTS](#), [nlsModelFit](#)

## Examples

```
## Not run:
# Classification Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
```

```

                                sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':= ' (x1 = NULL, x2 = NULL)]
aa[, target := as.factor(ifelse(target > 0.5,1,0))]
Construct <- data.table::data.table(Targets = rep("target",3),
                                Distribution = c("bernoulli",
                                                  "bernoulli",
                                                  "bernoulli"),
                                Loss = c("AUC", "AUC", "CrossEntropy"),
                                Quantile = rep(NA,3),
                                ModelName = c("GBM", "DRF", "DL"),
                                Algorithm = c("gbm",
                                              "randomForest",
                                              "deeplearning"),
                                dataName = rep("aa",3),
                                TargetCol = rep(c("1"),3),
                                FeatureCols = rep(c("2:11"),3),
                                CreateDate = rep(Sys.time(),3),
                                GridTune = rep(FALSE,3),
                                ExportValidData = rep(TRUE,3),
                                ParDep = rep(2,3),
                                PD_Data = rep("All",3),
                                ThreshType = rep("f1",3),
                                FSC = rep(0.001,3),
                                tpProfit = rep(NA,3),
                                tnProfit = rep(NA,3),
                                fpProfit = rep(NA,3),
                                fnProfit = rep(NA,3),
                                SaveModel = rep(FALSE,3),
                                SaveModelType = c("Mojo", "standard", "mojo"),
                                PredsAllData = rep(TRUE,3),
                                TargetEncoding = rep(NA,3),
                                SupplyData = rep(FALSE,3))

AutoH20Modeler(Construct,
               max_memory = "28G",
               ratios = 0.75,
               BL_Trees = 500,
               nthreads = 5,
               model_path = getwd(),
               MaxRuntimeSeconds = 3600,
               MaxModels = 30,
               TrainData = NULL,
               TestData = NULL)

# Multinomial Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +

```

```

                                sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':= ' (x1 = NULL, x2 = NULL)]
aa[, target := as.factor(ifelse(target < 0.33,"A",ifelse(target < 0.66, "B","C")))]
Construct <- data.table::data.table(Targets = rep("target",3),
                                Distribution = c("multinomial",
                                                "multinomial",
                                                "multinomial"),
                                Loss = c("logloss","logloss","CrossEntropy"),
                                Quantile = rep(NA,3),
                                ModelName = c("GBM","DRF","DL"),
                                Algorithm = c("gbm",
                                                "randomForest",
                                                "deeplearning"),
                                dataName = rep("aa",3),
                                TargetCol = rep(c("1"),3),
                                FeatureCols = rep(c("2:11"),3),
                                CreateDate = rep(Sys.time(),3),
                                GridTune = rep(FALSE,3),
                                ExportValidData = rep(TRUE,3),
                                ParDep = rep(NA,3),
                                PD_Data = rep("All",3),
                                ThreshType = rep("f1",3),
                                FSC = rep(0.001,3),
                                tpProfit = rep(NA,3),
                                tnProfit = rep(NA,3),
                                fpProfit = rep(NA,3),
                                fnProfit = rep(NA,3),
                                SaveModel = rep(FALSE,3),
                                SaveModelType = c("Mojo","standard","mojo"),
                                PredsAllData = rep(TRUE,3),
                                TargetEncoding = rep(NA,3),
                                SupplyData = rep(FALSE,3))

AutoH20Modeler(Construct,
               max_memory = "28G",
               ratios = 0.75,
               BL_Trees = 500,
               nthreads = 5,
               model_path = getwd(),

```

```

MaxRuntimeSeconds = 3600,
MaxModels = 30,
TrainData = NULL,
TestData = NULL)

# Regression Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':= ' (x1 = NULL, x2 = NULL)]
Construct <- data.table::data.table(Targets = rep("target",3),
                                     Distribution = c("gaussian",
                                                       "gaussian",
                                                       "gaussian"),
                                     Loss = c("MSE", "MSE", "Quadratic"),
                                     Quantile = rep(NA,3),
                                     ModelName = c("GBM", "DRF", "DL"),
                                     Algorithm = c("gbm",
                                                    "randomForest",
                                                    "deeplearning"),
                                     dataName = rep("aa",3),
                                     TargetCol = rep(c("1"),3),
                                     FeatureCols = rep(c("2:11"),3),
                                     CreateDate = rep(Sys.time(),3),
                                     GridTune = rep(FALSE,3),
                                     ExportValidData = rep(TRUE,3),
                                     ParDep = rep(2,3),
                                     PD_Data = rep("All",3),
                                     ThreshType = rep("f1",3),
                                     FSC = rep(0.001,3),
                                     tpProfit = rep(NA,3),
                                     tnProfit = rep(NA,3),
                                     fpProfit = rep(NA,3),
                                     fnProfit = rep(NA,3),
                                     SaveModel = rep(FALSE,3),
                                     SaveModelType = c("Mojo", "standard", "mojo"),

```

```

PredsAllData      = rep(TRUE,3),
TargetEncoding    = rep(NA,3),
SupplyData        = rep(FALSE,3))

AutoH20Modeler(Construct,
  max_memory = "28G",
  ratios = 0.75,
  BL_Trees = 500,
  nthreads = 5,
  model_path = getwd(),
  MaxRuntimeSeconds = 3600,
  MaxModels = 30,
  TrainData = NULL,
  TestData = NULL)

# Quantile Regression Example
Correl <- 0.85
aa <- data.table::data.table(target = runif(1000))
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, Independent_Variable1 := log(pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable2 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable3 := exp(pnorm(Correl * x1 +
                                       sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable4 := exp(exp(pnorm(Correl * x1 +
                                             sqrt(1-Correl^2) * qnorm(x2))))]
aa[, Independent_Variable5 := sqrt(pnorm(Correl * x1 +
                                         sqrt(1-Correl^2) * qnorm(x2)))]
aa[, Independent_Variable6 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^0.10]
aa[, Independent_Variable7 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^0.25]
aa[, Independent_Variable8 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^0.75]
aa[, Independent_Variable9 := (pnorm(Correl * x1 +
                                     sqrt(1-Correl^2) * qnorm(x2)))^2]
aa[, Independent_Variable10 := (pnorm(Correl * x1 +
                                      sqrt(1-Correl^2) * qnorm(x2)))^4]
aa[, ':= ' (x1 = NULL, x2 = NULL)]
Construct <- data.table::data.table(Targets = rep("target",3),
  Distribution      = c("quantile",
                        "quantile"),
  Loss              = c("MAE", "Absolute"),
  Quantile          = rep(0.75,2),
  ModelName         = c("GBM", "DL"),
  Algorithm         = c("gbm",
                        "deeplearning"),
  dataName          = rep("aa",2),
  TargetCol         = rep(c("1"),2),
  FeatureCols       = rep(c("2:11"),2),
  CreateDate        = rep(Sys.time(),2),
  GridTune          = rep(FALSE,2),
  ExportValidData   = rep(TRUE,2),
  ParDep            = rep(4,2),
  PD_Data           = rep("All",2),
  ThreshType        = rep("f1",2),

```



```

FSC                = rep(0.001,2),
tpProfit           = rep(NA,2),
tnProfit           = rep(NA,2),
fpProfit           = rep(NA,2),
fnProfit           = rep(NA,2),
SaveModel          = rep(FALSE,2),
SaveModelType      = c("Mojo","mojo"),
PredsAllData       = rep(TRUE,2),
TargetEncoding     = rep(NA,2),
SupplyData         = rep(FALSE,2))

AutoH2OModeler(Construct,
  max_memory = "28G",
  ratios = 0.75,
  BL_Trees = 500,
  nthreads = 5,
  model_path = getwd(),
  MaxRuntimeSeconds = 3600,
  MaxModels = 30,
  TrainData = NULL,
  TestData = NULL)

## End(Not run)

```

AutoTS

*AutoTS is an automated time series modeling function*

## Description

AutoTS builds the best time series models for each type, compares all types, selects the winner, and generates a forecast.

## Usage

```

AutoTS(data, TargetName = "Targets", DateName = "DateTime",
  FCPeriods = 30, HoldOutPeriods = 30, TimeUnit = "day", Lags = 25,
  SLags = 2, NumCores = 4, SkipModels = NULL, StepWise = TRUE)

```

## Arguments

<code>data</code>	is the source time series data.table
<code>TargetName</code>	is the name of the dependent variable in your data.table
<code>DateName</code>	is the name of the date column in your data.table
<code>FCPeriods</code>	is the number of periods into the future you wish to forecast
<code>HoldOutPeriods</code>	is the number of periods to use for validation testing
<code>TimeUnit</code>	is the level of aggregation your dataset comes in
<code>Lags</code>	is the number of lags you wish to test in various models (same with moving averages)
<code>SLags</code>	is the number of seasonal lags you wish to test in various models (same with moving averages)
<code>NumCores</code>	is the number of cores available on your computer

SkipModels	Don't run specified models - e.g. exclude all models "ARFIMA" "ARIMA" "ETS" "NNET" "TBATS" "TSLM" "PROPHET"
StepWise	Set to TRUE to have ARIMA and ARFIMA run a stepwise selection process. Otherwise, all models will be generated in parallel execution, but still run much slower.

**Value**

If Ensemble is TRUE, return a data.table object with a date column and the forecasts, an evaluation data set, and an ensemble training data set (all in a list). If Ensemble is FALSE, then all items returned except the ensemble training set.

**Author(s)**

Adrian Antico

**See Also**

Other Supervised Learning: [AutoH2OModeler](#), [nlsModelFit](#)

**Examples**

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(1000,
    mean = 50,
    sd = 20),
    filter=rep(1,10),
    circular=TRUE))
data[, temp := seq(1:1000)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
output <- AutoTS(data,
  TargetName = "Target",
  DateName = "DateTime",
  FCPeriods = 30,
  HoldOutPeriods = 30,
  TimeUnit = "day",
  Lags = 5,
  SLags = 1,
  NumCores = 4,
  SkipModels = NULL,
  StepWise = TRUE)
```

---

ChartTheme

---

*ChartTheme function is a ggplot theme generator for ggplots*


---

**Description**

This function helps your ggplots look professional with the choice of the two main colors that will dominate the theme

**Usage**

```
ChartTheme(Size = 12)
```

**Arguments**

Size                      The size of the axis labels and title

**Value**

An object to pass along to ggplot objects following the "+" sign

**Author(s)**

Adrian Antico

**See Also**

Other Misc: [PrintObjectsSize](#), [RemixTheme](#), [SimpleCap](#), [WordFreq](#), [multiplot](#), [percRank](#), [tempDatesFun](#), [tokenizeH20](#)

**Examples**

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(1000,
    mean = 50,
    sd = 20),
    filter=rep(1,10),
    circular=TRUE))
data[, temp := seq(1:1000)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
p <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime, y = Target)) + ggplot2::geom_line()
p <- p + ChartTheme(Size = 12)
```

---

DT\_GDL\_Feature\_Engineering

*An Automated Feature Engineering Function Using data.table  
frollmean*

---

**Description**

Builds autoregressive and moving average from target columns and distributed lags and distributed moving average for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and moving averages. This function works for data with groups and without groups.

**Usage**

```
DT_GDL_Feature_Engineering(data, lags = c(seq(1, 50, 1)),
  periods = c(seq(5, 95, 5)), statsNames = c("MA"),
  targets = c("qty"), groupingVars = c("Group1", "Group2"),
  sortDateName = c("date"), timeDiffTarget = c("TimeDiffName"),
  timeAgg = c("days"), WindowingLag = 0, Type = c("Lag"),
  Timer = TRUE, SkipCols = NULL, SimpleImpute = TRUE)
```



```

targets      = c("Target"),
groupingVars  = NULL,
sortDateName = "DateTime",
timeDiffTarget = c("Time_Gap"),
timeAgg      = c("days"),
WindowingLag = 1,
Type         = "Lag",
Timer        = TRUE,
SkipCols     = FALSE,
SimpleImpute = TRUE)

```

DummifyDT

*DummifyDT creates dummy variables for the selected columns.***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, KeepBaseCols = TRUE, OneHot = TRUE)
```

**Arguments**

<code>data</code>	the data set to run the micro auc on
<code>cols</code>	a vector with the names of the columns you wish to dichotomize
<code>KeepBaseCols</code>	set to TRUE to keep the original columns used in the dichotomization process
<code>OneHot</code>	Set to TRUE to run one hot encoding, FALSE to generate N columns for N levels

**Value**

data table with new dummy variables columns and optionally removes base columns

**Author(s)**

Adrian Antico

**See Also**

Other Feature Engineering: [DT\\_GDL\\_Feature\\_Engineering](#), [FAST\\_GDL\\_Feature\\_Engineering](#), [GDL\\_Feature\\_Engineering](#), [ModelDataPrep](#), [Scoring\\_GDL\\_Feature\\_Engineering](#), [Word2VecModel](#)

## Examples

```
test <- data.table::data.table(Value = runif(100000),
                              FactorCol = sample(x = c(letters,
                                                        LETTERS,
                                                        paste0(letters,letters),
                                                        paste0(LETTERS,LETTERS),
                                                        paste0(letters,LETTERS),
                                                        paste0(LETTERS,letters)),
                              size = 100000,
                              replace = TRUE))

test <- DummifyDT(data = test,
                  cols = "FactorCol",
                  KeepBaseCols = FALSE)

ncol(test)
test[, sum(FactorCol_gg)]
```

---

EvalPlot	<i>Function automatically builds calibration plots for model evaluation</i>
----------	-----------------------------------------------------------------------------

---

## Description

This function automatically builds calibration plots and calibration boxplots for model evaluation using regression, quantile regression, and binary and multinomial classification

## Usage

```
EvalPlot(data, PredColName = c("PredictedValues"),
          ActColName = c("ActualValues"), type = c("calibration"),
          bucket = 0.05, agrfun = function(x) base::mean(x, na.rm = TRUE))
```

## Arguments

data	Data containing predicted values and actual values for comparison
PredColName	String representation of column name with predicted values from model
ActColName	String representation of column name with actual values from model
type	Calibration or boxplot - calibration aggregated data based on summary statistic; boxplot shows variation
bucket	Number of buckets to partition the space on (0,1) for evaluation
aggrfun	The statistics function used in aggregation, listed as a function

## Value

Calibration plot or boxplot

## Author(s)

Adrian Antico

## See Also

Other Model Evaluation and Interpretation: [ParDepCalPlots](#), [RedYellowGreen](#), [threshOptim](#)

**Examples**

```
## Not run:
EvalPlot(data,
  PredColName = "predict",
  ActColName = "target",
  type = "calibration",
  bucket = 0.05,
  aggrfun = function(x) quantile(x, probs = 0.50, na.rm = TRUE))

## End(Not run)
```

---

FAST\_GDL\_Feature\_Engineering

*An Fast Automated Feature Engineering Function*


---

**Description**

For models with target variables within the realm of the current time frame but not too far back in time, this function creates autoregressive and rolling stats from target columns and distributed lags and distributed rolling stats for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and rolling stats. This function works for data with groups and without groups.

**Usage**

```
FAST_GDL_Feature_Engineering(data, lags = c(1:5), periods = c(seq(10,
  50, 10)), statsFUNs = c("mean", "median", "sd", "quantile85",
  "quantile95"), statsNames = c("mean", "median", "sd", "quantile85",
  "quantile95"), targets = c("Target"),
  groupingVars = c("GroupVariable"), sortDateName = c("DateTime"),
  timeDiffTarget = NULL, timeAgg = c("hours"), WindowingLag = 1,
  Type = c("Lag"), Timer = FALSE, SkipCols = FALSE,
  SimpleImpute = TRUE, AscRowByGroup = c("temp"), RecordsKeep = 1)
```

**Arguments**

<b>data</b>	The data source you want to run the function on
<b>lags</b>	The list of specific lags you want to have generated
<b>periods</b>	The number of periods for the rolling stats
<b>statsFUNs</b>	List of functions for your rolling windows, such as mean, sd, min, max, quantile
<b>statsNames</b>	The corresponding names to append to your colnames created associated with statsFuns
<b>targets</b>	The column(s) in which you will build your lags and rolling stats
<b>groupingVars</b>	Categorical variables you will build your lags and rolling stats by
<b>sortDateName</b>	String name of your core date column in your transaction data
<b>timeDiffTarget</b>	List a name in order to create time between events with associated lags and rolling features
<b>timeAgg</b>	Unit of time to aggregate by

WindowingLag	Build moving stats off of target column(s) or one of their lags (1+)
Type	input "Lag" if you want features built on historical values; use "Lead" if you want features built on future values
Timer	Set to TRUE if you want a time run for the operation; useful when there is grouping
SkipCols	Defaults to NULL; otherwise name the vector containing the names of columns to skip
SimpleImpute	Set to TRUE for factor level imputation of "0" and numeric imputation of -1
AscRowByGroup	Required to have a column with a Row Number by group (if grouping) with 1 being the record for scoring (typically the most current in time)
RecordsKeep	List the number of records to retain (1 for last record, 2 for last 2 records, etc.)

**Value**

data.table of original data plus newly created features

**Author(s)**

Adrian Antico

**See Also**

Other Feature Engineering: [DT\\_GDL\\_Feature\\_Engineering](#), [DummifyDT](#), [GDL\\_Feature\\_Engineering](#), [ModelDataPrep](#), [Scoring\\_GDL\\_Feature\\_Engineering](#), [Word2VecModel](#)

**Examples**

```

N = 25116
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(N,
    mean = 50,
    sd = 20),
    filter=rep(1,10),
    circular=TRUE))
data[, temp := seq(1:N)][, DateTime := DateTime - temp]
data <- data[order(DateTime)]
data <- FAST_GDL_Feature_Engineering(data,
  lags          = c(1:5),
  periods       = c(seq(10,50,10)),
  statsFUNs     = c("mean",
    "median",
    "sd",
    "quantile85",
    "quantile95"),
  statsNames    = c("mean",
    "median",
    "sd",
    "quantile85",
    "quantile95"),
  targets       = c("Target"),
  groupingVars  = NULL,
  sortDateName  = "DateTime",

```



```

timeDiffTarget = c("Time_Gap"),
timeAgg        = "days",
WindowingLag   = 1,
Type           = "Lag",
Timer          = TRUE,
SkipCols       = FALSE,
SimpleImpute   = TRUE,
AscRowByGroup  = "temp")

```

---

GDL\_Feature\_Engineering

*An Automated Feature Engineering Function*


---

## Description

Builds autoregressive and rolling stats from target columns and distributed lags and distributed rolling stats for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and rolling stats. This function works for data with groups and without groups.

## Usage

```

GDL_Feature_Engineering(data, lags = c(seq(1, 5, 1)), periods = c(3, 5,
  10, 15, 20, 25), statsFUNs = c(function(x) quantile(x, probs = 0.1,
  na.rm = TRUE), function(x) quantile(x, probs = 0.9, na.rm = TRUE),
  function(x) base::mean(x, na.rm = TRUE), function(x) sd(x, na.rm = TRUE),
  function(x) quantile(x, probs = 0.25, na.rm = TRUE), function(x)
  quantile(x, probs = 0.75, na.rm = TRUE)), statsNames = c("q10", "q90",
  "mean", "sd", "q25", "q75"), targets = c("qty"),
  groupingVars = c("Group1", "Group2"), sortDateName = c("date"),
  timeDiffTarget = c("TimeDiffName"), timeAgg = c("days"),
  WindowingLag = 0, Type = c("Lag"), Timer = TRUE, SkipCols = NULL,
  SimpleImpute = TRUE)

```

## Arguments

<code>data</code>	The data source you want to run the function on
<code>lags</code>	The list of specific lags you want to have generated
<code>periods</code>	The number of periods to use for rolling stats
<code>statsFUNs</code>	List of functions for your rolling windows, such as mean, sd, min, max, quantile
<code>statsNames</code>	The corresponding names to append to your colnames created associated with statsFUNs
<code>targets</code>	The column(s) in which you will build your lags and rolling stats
<code>groupingVars</code>	Categorical variables you will build your lags and rolling stats by
<code>sortDateName</code>	String name of your core date column in your transaction data
<code>timeDiffTarget</code>	List a name in order to create time between events with associated lags and rolling features
<code>timeAgg</code>	Unit of time to aggregate by

WindowingLag	Build moving stats off of target column(s) or one of their lags (1+)
Type	input "Lag" if you want features built on historical values; use "Lead" if you want features built on future values
Timer	Set to TRUE if you want a time run for the operation; useful when there is grouping
SkipCols	Defaults to NULL; otherwise name the vector containing the names of columns to skip
SimpleImpute	Set to TRUE for factor level imputation of "0" and numeric imputation of -1

### Value

data.table of original data plus newly created features

### Author(s)

Adrian Antico

### See Also

Other Feature Engineering: [DT\\_GDL\\_Feature\\_Engineering](#), [DummifyDT](#), [FAST\\_GDL\\_Feature\\_Engineering](#), [ModelDataPrep](#), [Scoring\\_GDL\\_Feature\\_Engineering](#), [Word2VecModel](#)

### Examples

```
N = 25116
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(N,
    mean = 50,
    sd = 20),
    filter=rep(1,10),
    circular=TRUE))
data[, temp := seq(1:N)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
data <- GDL_Feature_Engineering(data,
  lags = c(seq(1,5,1)),
  periods = c(3,5,10,15,20,25),
  statsFUNs = c(function(x) quantile(x, probs = 0.20, na.rm = TRUE),
    function(x) quantile(x, probs = 0.80, na.rm = TRUE),
    function(x) mean(x, na.rm = TRUE),
    function(x) sd(x, na.rm = TRUE),
    function(x) quantile(x, probs = 0.10, na.rm = TRUE),
    function(x) quantile(x, probs = 0.90, na.rm = TRUE)),
  statsNames = c("min", "max", "mean", "sd", "q20", "q80"),
  targets = c("Target"),
  groupingVars = NULL,
  sortDateName = "DateTime",
  timeDiffTarget = c("Time_Gap"),
  timeAgg = "days",
  WindowingLag = 1,
  Type = "Lag",
  Timer = TRUE,
  SkipCols = FALSE,
  SimpleImpute = TRUE)
```

---

GenTSAnomVars	<i>GenTSAnomVars is an automated z-score anomaly detection via GLM-like procedure</i>
---------------	---------------------------------------------------------------------------------------

---

## Description

GenTSAnomVars is an automated z-score anomaly detection via GLM-like procedure. Data is z-scaled and grouped by factors and time periods to determine which points are above and below the control limits in a cumulative time fashion. Then a cumulative rate is created as the final variable. Set KeepAllCols to FALSE to utilize the intermediate features to create rolling stats from them.

## Usage

```
GenTSAnomVars(data, ValueCol = "Value", GroupVar1 = "SKU",
  GroupVar2 = NULL, DateVar = "DATE", High = 1.96, Low = -1.96,
  KeepAllCols = FALSE, DataScaled = TRUE)
```

## Arguments

data	the source residuals data.table
ValueCol	the numeric column to run anomaly detection over
GroupVar1	this is a group by variable
GroupVar2	this is another group by variable
DateVar	this is a time variable for grouping
High	this is the threshold on the high end
Low	this is the threshold on the low end
KeepAllCols	set to TRUE to remove the intermediate features
DataScaled	set to TRUE if you already scaled your data

## Value

The original data.table with the added columns merged in

## Author(s)

Adrian Antico

## See Also

Other Unsupervised Learning: [GLRM\\_KMeans.Col](#), [ResidualOutliers](#)

## Examples

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
  Target = stats::filter(rnorm(1000,
    mean = 50,
    sd = 20),
    filter=rep(1,10),
    circular=TRUE))
```

```

data[, temp := seq(1:10000)][, DateTime := DateTime - temp][, temp := NULL]
data <- data[order(DateTime)]
x <- data.table::as.data.table(sde::GBM(N=10000)*1000)
data[, predicted := x[-1,]]
stuff    <- GenTSAnomVars(data,
                           ValueCol   = "Target",
                           GroupVar1  = NULL,
                           GroupVar2  = NULL,
                           DateVar    = "DateTime",
                           High        = 1.96,
                           Low         = -1.96,
                           KeepAllCols = TRUE,
                           DataScaled = FALSE)

```

---

GLRM_KMeans_Col	<i>GLRM_KMeans_Col Automated row clustering for mixed column types</i>
-----------------	------------------------------------------------------------------------

---

## Description

GLRM\_KMeans\_Col adds a column to your original data with a cluster number identifier. Uses glrm (grid tune-able) and then k-means to find optimal k.

## Usage

```

GLRM_KMeans_Col(data, GridTuneGLRM = TRUE, GridTuneKMeans = TRUE,
  nthreads = 4, MaxMem = "14G", glrmCols = 3:ncol(data),
  IgnoreConstCols = TRUE, glrmFactors = 5, Loss = "Absolute",
  glrmMaxIters = 1000, SVDMethod = "Randomized",
  MaxRunTimeSecs = 3600, KMeansK = 50, KMeansMetric = "totss")

```

## Arguments

<code>data</code>	is the source time series data.table
<code>GridTuneGLRM</code>	If you want to grid tune the glrm model, set to TRUE, FALSE otherwise
<code>GridTuneKMeans</code>	If you want to grid tune the KMeans model, set to TRUE, FALSE otherwise
<code>nthreads</code>	set based on number of threads your machine has available
<code>MaxMem</code>	set based on the amount of memory your machine has available
<code>glrmCols</code>	the column numbers for the glrm
<code>IgnoreConstCols</code>	tell H2O to ignore any columns that have zero variance
<code>glrmFactors</code>	similar to the number of factors to return from PCA
<code>Loss</code>	set to one of "Quadratic", "Absolute", "Huber", "Poisson", "Hinge", "Logistic", "Periodic"
<code>glrmMaxIters</code>	max number of iterations
<code>SVDMethod</code>	choose from "Randomized", "GramSVD", "Power"
<code>MaxRunTimeSecs</code>	set the timeout for max run time
<code>KMeansK</code>	number of factors to test out in k-means to find the optimal number
<code>KMeansMetric</code>	pick the metric to identify top model in grid tune c("totss", "betweeness", "withinss")

**Value**

Original data.table with added column with cluster number identifier

**Author(s)**

Adrian Antico

**See Also**

Other Unsupervised Learning: [GenTSAnomVars](#), [ResidualOutliers](#)

**Examples**

```
## Not run:
data <- data.table::as.data.table(iris)
data <- GLRM_KMeans_Col(data,
                        GridTuneGLRM = TRUE,
                        GridTuneKMeans = TRUE,
                        nthreads = 8,
                        MaxMem = "28G",
                        glrmCols = 1:(ncol(data)-1),
                        IgnoreConstCols = TRUE,
                        glrmFactors = 2,
                        Loss = "Absolute",
                        glrmMaxIters = 1000,
                        SVDMethod = "Randomized",
                        MaxRunTimeSecs = 3600,
                        KMeansK = 5)

unique(data[["Species"]])
unique(data[["ClusterID"]])
temp <- data[, mean(ClusterID), by = "Species"]
Setosa <- round(temp[Species == "setosa", V1][[1]],0)
Versicolor <- round(temp[Species == "versicolor", V1][[1]],0)
Virginica <- round(temp[Species == "virginica", V1][[1]],0)
data[, Check := "a"]
data[ClusterID == eval(Setosa), Check := "setosa"]
data[ClusterID == eval(Virginica), Check := "virginica"]
data[ClusterID == eval(Versicolor), Check := "versicolor"]
data[, Acc := as.numeric(ifelse(Check == Species, 1, 0))]
data[, mean(Acc)][[1]]

## End(Not run)
```

---

ModelDataPrep

*Final Data Preparation Function*


---

**Description**

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

**Usage**

```
ModelDataPrep(data, Impute = TRUE, CharToFactor = TRUE,
               MissFactor = "0", MissNum = -1)
```

**Arguments**

<code>data</code>	This is your source data you'd like to modify
<code>Impute</code>	Defaults to TRUE which tells the function to impute the data
<code>CharToFactor</code>	Defaults to TRUE which tells the function to convert characters to factors
<code>MissFactor</code>	Supply the value to impute missing factor levels
<code>MissNum</code>	Supply the value to impute missing numeric values

**Value**

Returns the original data table with corrected values

**Author(s)**

Adrian Antico

**See Also**

Other Feature Engineering: [DT\\_GDL\\_Feature\\_Engineering](#), [DummifyDT](#), [FAST\\_GDL\\_Feature\\_Engineering](#), [GDL\\_Feature\\_Engineering](#), [Scoring\\_GDL\\_Feature\\_Engineering](#), [Word2VecModel](#)

**Examples**

```
data <- data.table::data.table(Value = runif(100000),
                               FactorCol = as.character(sample(x = c(letters,
                                                                       LETTERS,
                                                                       paste0(letters, letters),
                                                                       paste0(LETTERS, LETTERS),
                                                                       paste0(letters, LETTERS),
                                                                       paste0(LETTERS, letters)),
                               size = 100000,
                               replace = TRUE)))

data <- ModelDataPrep(data,
                      Impute = TRUE,
                      CharToFactor = TRUE,
                      MissFactor = "0",
                      MissNum    = -1)
```

---

multiplot

*Multiplot is a function for combining multiple plots*

---

**Description**

Sick of copying this one into your code? Well, not anymore.

**Usage**

```
multiplot(..., plotlist = NULL, cols = 2, layout = NULL)
```

**Arguments**

<code>...</code>	Passthrough arguments
<code>plotlist</code>	This is the list of your charts
<code>cols</code>	This is the number of columns in your multiplot
<code>layout</code>	Leave NULL

**Value**

Multiple ggplots on a single image

**Author(s)**

Adrian Antico

**See Also**

Other Misc: [ChartTheme](#), [PrintObjectsSize](#), [RemixTheme](#), [SimpleCap](#), [WordFreq](#), [percRank](#), [tempDatesFun](#), [tokenizeH20](#)

**Examples**

```
## Not run:
multiplot(plotlist = list(p1,p2,p3,p4), cols = 2)

## End(Not run)
```

---

nlsModelFit	<i>nlsModelFit is a function for automatically building nls models</i>
-------------	------------------------------------------------------------------------

---

**Description**

This function will build models for 9 different nls models, along with a non-parametric monotonic regression and a polynomial regression. The models are evaluated, a winner is picked, and the predicted values are stored in your data table.

**Usage**

```
nlsModelFit(data, y, x, monotonic = TRUE)
```

**Arguments**

<code>data</code>	Data is the data table you are building the modeling on
<code>y</code>	Y is the target variable name in quotes
<code>x</code>	X is the independent variable name in quotes
<code>monotonic</code>	This is a TRUE/FALSE indicator - choose TRUE if you want monotonic regression over polynomial regression

**Value**

A data table with your original column replaced by the nls model predictions

**Author(s)**

Adrian Antico

**See Also**

Other Supervised Learning: [AutoH2OModeler](#), [AutoTS](#)

**Examples**

```
data <- data.table::data.table(Variable = seq(1,500,1), Target = rep(1, 500))
for (i in as.integer(1:500)) {
  if(i == 1) {
    var <- data[i, "Variable"][[1]]
    data.table::set(data, i = i, j = 2L, value = var * (1 + runif(1)/100))
  } else {
    var = data[i-1, "Target"][[1]]
    data.table::set(data, i = i, j = 2L, value = var * (1 + runif(1)/100))
  }
}

# To keep original values
data1 <- data.table::copy(data)

# Merge and Model data
data2 <- merge(data1,
  nlsModelFit(data = data, y = "Target", x = "Variable", monotonic = FALSE),
  by = "Variable",
  all = TRUE)

# Plot graphs of predicted vs actual
p <- ggplot2::ggplot(data2, ggplot2::aes(x = Variable)) +
  ggplot2::geom_line(ggplot2::aes(y = data2[["Target.x"]], color = "blue")) +
  ggplot2::geom_line(ggplot2::aes(y = data2[["Target.y"]], color = "red")) +
  ChartTheme(Size = 12) + ggplot2::ggtitle("Growth Models") +
  ggplot2::ylab("Target Variable") + ggplot2::xlab("Independent Variable")
```

---

ParDepCalPlots

*Function automatically builds partial dependence calibration plots  
for model evaluation*

---

**Description**

This function automatically builds partial dependence calibration plots and partial dependence calibration boxplots for model evaluation using regression, quantile regression, and binary and multinomial classification

**Usage**

```
ParDepCalPlots(data, PredColName = c("PredictedValues"),
  ActColName = c("ActualValues"),
  IndepVar = c("Independent_Variable_Name"), type = c("calibration"),
  bucket = 0.05, FactLevels = 10, Function = function(x)
  base::mean(x, na.rm = TRUE))
```



**Arguments**

<code>data</code>	Data containing predicted values and actual values for comparison
<code>PredColName</code>	String representation of the column name with predicted values from model
<code>ActColName</code>	String representation of the column name with actual values from model
<code>IndepVar</code>	String representation of the column name with the independent variable of choice
<code>type</code>	Calibration or boxplot - calibration aggregated data based on summary statistic; boxplot shows variation
<code>bucket</code>	Number of buckets to partition the space on (0,1) for evaluation
<code>FactLevels</code>	The number of levels to show on the chart (1. Levels are chosen based on frequency; 2. all other levels grouped and labeled as "Other")
<code>Function</code>	Supply the function you wish to use for aggregation.

**Value**

Partial dependence calibration plot or boxplot

**Author(s)**

Adrian Antico

**See Also**

Other Model Evaluation and Interpretation: [EvalPlot](#), [RedYellowGreen](#), [threshOptim](#)

**Examples**

```
## Not run:
ParDepCalPlots(data,
  PredColName = "predict",
  ActColName  = "target",
  IndepVar    = "Independent_Variable",
  type        = "boxplot",
  bucket      = 0.05,
  FactLevels  = 10,
  Function     = function(x) mean(x, na.rm = TRUE))

## End(Not run)
```

---

percRank

*Percentile rank function*

---

**Description**

This function computes percentile ranks for each row in your data like Excel's PERCENT\_RANK

**Usage**

```
percRank(x)
```

**Arguments**

x                      X is your variable of interest

**Value**

vector of percentile ranks

**Author(s)**

Adrian Antico

**See Also**

Other Misc: [ChartTheme](#), [PrintObjectsSize](#), [RemixTheme](#), [SimpleCap](#), [WordFreq](#), [multiplot](#), [tempDatesFun](#), [tokenizeH20](#)

**Examples**

```
## Not run:
percRank(x)

## End(Not run)
```

---

PrintObjectsSize	<i>PrintObjectsSize prints out the top N objects and their associated sizes, sorted by size</i>
------------------	-------------------------------------------------------------------------------------------------

---

**Description**

PrintObjectsSize prints out the top N objects and their associated sizes, sorted by size

**Usage**

```
PrintObjectsSize(N = 10)
```

**Arguments**

N                      The number of objects to display

**Value**

The objects in your environment and their sizes

**Author(s)**

Adrian Antico

**See Also**

Other Misc: [ChartTheme](#), [RemixTheme](#), [SimpleCap](#), [WordFreq](#), [multiplot](#), [percRank](#), [tempDatesFun](#), [tokenizeH20](#)

**Examples**

```
## Not run:
PrintObjectsSize(N = 10)

## End(Not run)
```

---

RedYellowGreen	<i>RedYellowGreen is for determining the optimal thresholds for binary classification when do-nothing is an option</i>
----------------	------------------------------------------------------------------------------------------------------------------------

---

**Description**

This function will find the optimal thresholds for applying the main label and for finding the optimal range for doing nothing when you can quantify the cost of doing nothing

**Usage**

```
RedYellowGreen(data, PredictColNumber = 2, ActualColNumber = 1,
  TruePositiveCost = 0, TrueNegativeCost = 0,
  FalsePositiveCost = -10, FalseNegativeCost = -50, MidTierCost = -2,
  Cores = 8, Precision = 0.01)
```

**Arguments**

data	data is the data table with your predicted and actual values from a classification model
PredictColNumber	The column number where the actual target variable is located (in binary form)
ActualColNumber	The column number where the predicted values are located
TruePositiveCost	This is the utility for generating a true positive prediction
TrueNegativeCost	This is the utility for generating a true negative prediction
FalsePositiveCost	This is the cost of generating a false positive prediction
FalseNegativeCost	This is the cost of generating a false negative prediction
MidTierCost	This is the cost of doing nothing (or whatever it means to not classify in your case)
Cores	Number of cores on your machine
Precision	Set the decimal number to increment by between 0 and 1

**Value**

A data table with all evaluated strategies, parameters, and utilities, along with a 3d scatterplot of the results

**Author(s)**

Adrian Antico

**See Also**

Other Model Evaluation and Interpretation: [EvalPlot](#), [ParDepCalPlots](#), [threshOptim](#)

**Examples**

```
## Not run:
data <- RedYellowGreen(data,
                        PredictColNumber = 1,
                        ActualColNumber = 2,
                        TruePositiveCost = 0,
                        TrueNegativeCost = 0,
                        FalsePositiveCost = -1,
                        FalseNegativeCost = -2,
                        MidTierCost = -0.5)

## End(Not run)
```

---

RemixTheme

*RemixTheme function is a ggplot theme generator for ggplots*


---

**Description**

This function adds the Remix Theme to ggplots

**Usage**

```
RemixTheme()
```

**Value**

An object to pass along to ggplot objects following the "+" sign

**Author(s)**

DougVegas

**See Also**

Other Misc: [ChartTheme](#), [PrintObjectsSize](#), [SimpleCap](#), [WordFreq](#), [multiplot](#), [percRank](#), [tempDatesFun](#), [tokenizeH20](#)

**Examples**

```
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
                               Target = stats::filter(rnorm(1000,
                                                           mean = 50,
                                                           sd = 20),
                                                       filter=rep(1,10),
                                                       circular=TRUE))
data[, temp := seq(1:1000)][, DateTime := DateTime - temp][, temp := NULL]
```

```
data <- data[order(DateTime)]
p <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime, y = Target)) + ggplot2::geom_line()
p <- p + RemixTheme()
```

---

ResidualOutliers	<i>ResidualOutliers is an automated time series outlier detection function</i>
------------------	--------------------------------------------------------------------------------

---

## Description

ResidualOutliers is an automated time series outlier detection function that utilizes tsoutliers and auto.arima.

## Usage

```
ResidualOutliers(data, maxN = 5, cvar = 4)
```

## Arguments

data	the source residuals data.table
maxN	the largest lag or moving average (seasonal too) values for the arima fit
cvar	the t-stat value for tsoutliers

## Value

A data.table with outliers, the arima model, and residuals from the arima fit

## Author(s)

Adrian Antico

## See Also

Other Unsupervised Learning: [GLRM\\_KMeans\\_Col](#), [GenTSAnomVars](#)

## Examples

```
data <- data.table::data.table(a = seq(0,10000,1),
                             predicted = sde::GBM(N=10000)*1000)[-1,]
data <- data.table::data.table(a = seq(1,10000,1),
                             predicted = sde::rcCIR(n=10000,
                                                    Dt=0.1,
                                                    x0=1,
                                                    theta=c(6,2,2)))
data <- data.table::data.table(a = seq(1,10000,1),
                             predicted = sde::rsOU(n=10000,
                                                    theta=c(0,2,1)))

stuff <- ResidualOutliers(data = data, maxN = 5, cvar = 4)
data <- stuff[[1]]
model <- stuff[[2]]
resid <- stuff[[3]]
outliers <- data[type != "<NA>"]
```

---

Scoring\_GDL\_Feature\_Engineering

*An Automated Scoring Feature Engineering Function*


---

**Description**

For scoring purposes (brings back a single row by group), this function creates autoregressive and rolling stats from target columns and distributed lags and distributed rolling stats for independent features distributed across time. On top of that, you can also create time between instances along with their associated lags and rolling stats. This function works for data with groups and without groups.

**Usage**

```
Scoring_GDL_Feature_Engineering(data, lags = c(1:6, 12, seq(24, 168,
  24)), periods = c(6, 12, 24, 72, 168, 720, 4320, 8640),
  statsFUNs = c(function(x) base::mean(x, na.rm = TRUE), function(x)
  base::sd(x, na.rm = TRUE)), statsNames = c("mean", "sd"),
  targets = c("Target"), groupingVars = c("GroupVariable"),
  sortDateName = c("DateTime"), timeDiffTarget = c("Time_Gap"),
  timeAgg = c("days"), WindowingLag = 1, Type = c("Lag"),
  Timer = FALSE, SkipCols = FALSE, SimpleImpute = TRUE,
  AscRowByGroup = c("temp"), RecordsKeep = 1)
```

**Arguments**

<b>data</b>	The data source you want to run the function on
<b>lags</b>	The list of specific lags you want to have generated
<b>periods</b>	The number of periods in the rolling statistics
<b>statsFUNs</b>	List of functions for your rolling windows, such as mean, sd, min, max, quantile
<b>statsNames</b>	The corresponding names to append to your colnames created associated with statsFuns
<b>targets</b>	The column(s) in which you will build your lags and rolling stats
<b>groupingVars</b>	Categorical variables you will build your lags and rolling stats by
<b>sortDateName</b>	String name of your core date column in your transaction data
<b>timeDiffTarget</b>	List a name in order to create time between events with associated lags and rolling features
<b>timeAgg</b>	Unit of time to aggregate by
<b>WindowingLag</b>	Build moving stats off of target column(s) or one of their lags (1+)
<b>Type</b>	input "Lag" if you want features built on historical values; use "Lead" if you want features built on future values
<b>Timer</b>	Set to TRUE if you want a time run for the operation; useful when there is grouping
<b>SkipCols</b>	Defaults to NULL; otherwise name the vector containing the names of columns to skip



---

SimpleCap	<i>SimpleCap function is for capitalizing the first letter of words</i>
-----------	-------------------------------------------------------------------------

---

**Description**

SimpleCap function is for capitalizing the first letter of words (need I say more?)

**Usage**

```
SimpleCap(x)
```

**Arguments**

x	Column of interest
---	--------------------

**Value**

An object to pass along to ggplot objects following the "+" sign

**Author(s)**

Adrian Antico

**See Also**

Other Misc: [ChartTheme](#), [PrintObjectsSize](#), [RemixTheme](#), [WordFreq](#), [multiplot](#), [percRank](#), [tempDatesFun](#), [tokenizeH20](#)

**Examples**

```
x <- "adrian"
x <- SimpleCap(x)
```

---

tempDatesFun	<i>tempDatesFun Convert Excel datetime char columns to Date columns</i>
--------------	-------------------------------------------------------------------------

---

**Description**

tempDatesFun takes the Excel datetime column, which imports as character, and converts it into a date type

**Usage**

```
tempDatesFun(x)
```

**Arguments**

x	The column of interest
---	------------------------



**Value**

An object to pass along to ggplot objects following the "+" sign

**Author(s)**

Adrian Antico

**See Also**

Other Misc: [ChartTheme](#), [PrintObjectsSize](#), [RemixTheme](#), [SimpleCap](#), [WordFreq](#), [multiplot](#), [percRank](#), [tokenizeH20](#)

**Examples**

```
## Not run:
Cdata[, DAY_DATE := tempDatesFun(DAY_DATE)]
Cdata[, DAY_DATE := base::as.Date(DAY_DATE, "%m/%d/%Y")]

## End(Not run)
```

---

threshOptim

*Utility maximizing thresholds for binary classification*

---

**Description**

This function will return the utility maximizing threshold for future predictions along with the data generated to estimate the threshold

**Usage**

```
threshOptim(data, actTar = "target", predTar = "p1", tpProfit = 0,
            tnProfit = 0, fpProfit = -1, fnProfit = -2)
```

**Arguments**

data	data is the data table you are building the modeling on
actTar	The column name where the actual target variable is located (in binary form)
predTar	The column name where the predicted values are located
tpProfit	This is the utility for generating a true positive prediction
tnProfit	This is the utility for generating a true negative prediction
fpProfit	This is the cost of generating a false positive prediction
fnProfit	This is the cost of generating a false negative prediction

**Value**

Optimal threshold and corresponding utilities for the range of thresholds tested

**Author(s)**

Adrian Antico

## See Also

Other Model Evaluation and Interpretation: [EvalPlot](#), [ParDepCalPlots](#), [RedYellowGreen](#)

## Examples

```
## Not run:
data <- threshOptim(data      = data,
                    actTar    = "target",
                    predTar    = "p1",
                    tpProfit   = 0,
                    tnProfit   = 0,
                    fpProfit   = -1,
                    fnProfit   = -2)
optimalThreshold <- data[[1]]
allResults       <- data[[2]]

## End(Not run)
```

---

tokenizeH20

*For NLP work*

---

## Description

This function tokenizes data

## Usage

```
tokenizeH20(data)
```

## Arguments

data                    The text data

## Author(s)

Adrian Antico at [RemixInstitute.com](http://RemixInstitute.com)

## See Also

Other Misc: [ChartTheme](#), [PrintObjectsSize](#), [RemixTheme](#), [SimpleCap](#), [WordFreq](#), [multiplot](#), [percRank](#), [tempDatesFun](#)

---

Word2VecModel

*Automated word2vec data generation via H2O*


---

## Description

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

## Usage

```
Word2VecModel(data, stringCol = c("Text_Col1", "Text_Col2"),
  KeepStringCol = FALSE, model_path = getwd(),
  ModelID = c("Text_Col1", "Text_Col2"), vects = 5,
  SaveStopWords = FALSE, MinWords = 1, WindowSize = 1, Epochs = 25,
  StopWords = NULL, SaveModel = "standard", Threads = 4,
  MaxMemory = "14G")
```

## Arguments

<code>data</code>	Source data table to merge vects onto
<code>stringCol</code>	A string name for the column to convert via word2vec
<code>KeepStringCol</code>	Set to TRUE if you want to keep the original string column that you convert via word2vec
<code>model_path</code>	A string path to the location where you want the model and metadata stored
<code>ModelID</code>	A vector of your model names
<code>vects</code>	The number of vectors to retain from the word2vec model
<code>SaveStopWords</code>	Set to TRUE to save the stop words used
<code>MinWords</code>	For H2O word2vec model
<code>WindowSize</code>	For H2O word2vec model
<code>Epochs</code>	For H2O word2vec model
<code>StopWords</code>	For H2O word2vec model
<code>SaveModel</code>	Set to "standard" to save normally; set to "mojo" to save as mojo
<code>Threads</code>	Number of available threads you want to dedicate to model building
<code>MaxMemory</code>	Amount of memory you want to dedicate to model building

## Author(s)

Adrian Antico

## See Also

Other Feature Engineering: [DT\\_GDL\\_Feature\\_Engineering](#), [DummifyDT](#), [FAST\\_GDL\\_Feature\\_Engineering](#), [GDL\\_Feature\\_Engineering](#), [ModelDataPrep](#), [Scoring\\_GDL\\_Feature\\_Engineering](#)

## Examples

```
## Not run:
data <- Word2VecModel(data,
                      stringCol = c("Text_Col1",
                                    "Text_Col2"),
                      KeepStringCol = FALSE,
                      model_path = getwd(),
                      ModelID      = c("Text_Col1",
                                    "Text_Col2"),
                      vects       = 5,
                      SaveStopWords = FALSE,
                      MinWords     = 1,
                      WindowSize   = 1,
                      Epochs       = 25,
                      StopWords    = NULL,
                      SaveModel    = "standard",
                      Threads      = 4,
                      MaxMemory    = "14G")

## End(Not run)
```

---

WordFreq

*Automated Word Frequency and Word Cloud Creation*


---

## Description

This function builds a word frequency table and a word cloud. It prepares data, cleans text, and generates output.

## Usage

```
WordFreq(data, TextColName = "DESCR",
          ClusterCol = "ClusterAllNoTarget", ClusterID = 0,
          RemoveEnglishStopwords = TRUE, Stemming = TRUE,
          StopWords = c("bla", "blab2"))
```

## Arguments

<code>data</code>	Source data table
<code>TextColName</code>	A string name for the column
<code>ClusterCol</code>	Set to NULL to ignore, otherwise set to Cluster column name (or factor column name)
<code>ClusterID</code>	Must be set if ClusterCol is defined. Set to cluster ID (or factor level)
<code>RemoveEnglishStopwords</code>	Set to TRUE to remove English stop words, FALSE to ignore
<code>Stemming</code>	Set to TRUE to run stemming on your text data
<code>StopWords</code>	Add your own stopwords, in vector format

## Author(s)

Adrian Antico

**See Also**

Other Misc: [ChartTheme](#), [PrintObjectsSize](#), [RemixTheme](#), [SimpleCap](#), [multiplot](#), [percRank](#), [tempDatesFun](#), [tokenizeH20](#)

**Examples**

```
## Not run:
data <- WordFreq(data,
  TextColName = "DESCR",
  ClusterCol = "ClusterAllNoTarget",
  ClusterID = 0,
  RemoveEnglishStopwords = TRUE,
  Stemming = TRUE,
  StopWords = c("bla1", "bla2")

## End(Not run)
```

# Index

AutoH20Modeler, [2](#), [10](#), [23](#)  
AutoTS, [4](#), [9](#), [23](#)  
  
ChartTheme, [10](#), [23](#), [26](#), [28](#), [32–34](#), [37](#)  
  
DT\_GDL\_Feature\_Engineering, [11](#), [13](#), [16](#),  
[18](#), [22](#), [31](#), [35](#)  
DummifyDT, [12](#), [13](#), [16](#), [18](#), [22](#), [31](#), [35](#)  
  
EvalPlot, [14](#), [25](#), [27](#), [34](#)  
  
FAST\_GDL\_Feature\_Engineering, [12](#), [13](#), [15](#),  
[18](#), [22](#), [31](#), [35](#)  
  
GDL\_Feature\_Engineering, [12](#), [13](#), [16](#), [17](#),  
[22](#), [31](#), [35](#)  
GenTSAnomVars, [18](#), [20](#), [29](#)  
GLRM\_KMeans\_Col, [19](#), [20](#), [29](#)  
  
ModelDataPrep, [12](#), [13](#), [16](#), [18](#), [21](#), [31](#), [35](#)  
multiplot, [11](#), [22](#), [26](#), [28](#), [32–34](#), [37](#)  
  
nlsModelFit, [4](#), [10](#), [23](#)  
  
ParDepCalPlots, [14](#), [24](#), [27](#), [34](#)  
percRank, [11](#), [23](#), [25](#), [26](#), [28](#), [32–34](#), [37](#)  
PrintObjectsSize, [11](#), [23](#), [26](#), [26](#), [28](#),  
[32–34](#), [37](#)  
  
RedYellowGreen, [14](#), [25](#), [27](#), [34](#)  
RemixTheme, [11](#), [23](#), [26](#), [28](#), [32–34](#), [37](#)  
ResidualOutliers, [19](#), [20](#), [29](#)  
  
Scoring\_GDL\_Feature\_Engineering, [12](#), [13](#),  
[16](#), [18](#), [22](#), [30](#), [35](#)  
SimpleCap, [11](#), [23](#), [26](#), [28](#), [32](#), [33](#), [34](#), [37](#)  
  
tempDatesFun, [11](#), [23](#), [26](#), [28](#), [32](#), [32](#), [34](#), [37](#)  
threshOptim, [14](#), [25](#), [27](#), [33](#)  
tokenizeH20, [11](#), [23](#), [26](#), [28](#), [32](#), [33](#), [34](#), [37](#)  
  
Word2VecModel, [12](#), [13](#), [16](#), [18](#), [22](#), [31](#), [35](#)  
WordFreq, [11](#), [23](#), [26](#), [28](#), [32–34](#), [36](#)