RemixAutoML Library Introduction

Adrian Antico

2019-04-17

Contact Info

LinkedIn: https://www.linkedin.com/in/adrian-antico/

Remix Instute: https://www.remyxcourses.com or https://www.remixinstitute.ai

Vignette Intent

This vignette is designed to give you the highlights of the set of automated machine learning functions available in the RemixAutoML package. To see the functions in action, visit the Remyx Courses website for the free course at **https://www.remyxcourses.com** and walk through them (and check our the other courses too!).

Package Goals

The **RemixAutoML** package (*Remix Automated Machine Learning*) is designed to automate and optimize the quality of machine learning, the pace of development, along with the handling of big data and the processing time of data management. The library has been a development task at Remix Institute over the course of the past year to consolidate all of our winning methods for successfully completing machine learning and data science consulting projects. These were actual projects at Fortune 500 companies, Fortune 100 companies, tech startups, and other consulting clients. We are avid R users and feel that the R community could benefit from its release.

Package Design Philosophy

Core packages utilized

The two core packages RemixAutoML relies on are H2O and data.table. There are other packages used, for example, the forecast package, but H2O and data.table are used the most. I use data.table for data wrangling of all internal functions due to its ability to handle big data with minimal memory and the speed at which their functions process data. I chose to use H2O and their machine learning algorithms because of their high quality results, flexibility of use, ease of operationalization, and ability to manage big data. I use these functions routinely for machine learning projects and they continue to outperform every other package / software I test them against. Many of the other R packages for modeling or data manipulation have slow run times and fail once I get going with bigger data. As a simple example, sometimes I do a one-hot encoding for testing out keras models and I can just run my DummifyDT function to create those. I've tried out a few others with fast runtimes but they fail on bigger data, mess up the column ordering, and don't offer the flexibility of creating one-hot encoding features versus standard dichotomized features.

Machine learning methodology

The package is designed to give your models to the best chance at being the most accurate for your machine learning tasks. There are functions in here to help you get the most out of your data for the models to take advantage of. There are three types of features you need to manage for machine learning: numeric,

factor, and text data. The functions in here will help you squeeze as much information out of your data set as possible. Then you feed those data sets into the best machine learning algorithms where you can automatically grid-tune and let them compete against each other.

Handling numerical features

The typical challenge with numerical data is modeling nonlinear relationships. The models used handle that aspect so you don't have to worry about it. Interactions of numerical data are also handled internally as tree-based models and deep learning do well at capturing those relationships. The areas of numerical data typically missed relates to the effect of numerical features across time. The existence of a specific value can have impacts over time. Trends in the features and target are also predictive of future events. The length of history you account for also matters. The *GDL* functions will let you generate lags from the target variable and other numeric features, along with rolling statistics on the target variable and other numerical features. You can also specify to return information about the numerical time between records and generate their lags and rolling statistics on the target and other numerical features.

Handling categorical features

In other modeling frameworks, such as Python, you need to convert your categorical features into dummy variables. This means you need to take care of that coding task along with managing that in a production environment. With this package, categorical features are handled internally within the modeling function **AutoH2OModeler()**. Turning your categorical features into dummy variables in problematic for high cardinality factor variables. There are other approaches and the AutoH20Modeler will actually test out the other methods to see which ones offer the best performance. So you don't have to deal with the coding and management of factor variables and you get better performance.

Handling text data

We are living in a world now where text data is becoming more readily available. Now, you can simply run the **AutoWord2VecModeler()** function to build models for all your text columns, save the models, and recreate them on the fly with the **AutoH2OScoring()** function in your production setting.

Machine learning algorithms

I've seen several packages and automated machine learning software that brags about all the models they have available. However, when you've been doing this work as long as I have, you eventually realize that most of the models out there aren't competitive. In fact, H2O was smart not to go after all the possible algorithms out there and instead focused on enhancing the models with a track record of being the best for regression, quantile regression, classification, and multinomial classification. The grid-tuning capabilities are exellent for squeezing that last drop of gain out of your machine learning project. Most of the gains with come from the feature engineering but you won't be able to squeeze more gains out of other package models. On top of that, you can manage the amount of memory and threads you are willing to use for your machine learning tasks and when you work in a shared environment, that can come in handy.

Interpreting your models

Once you have your models developed, you or a boss may want to see what features are most important and their relationship to the target variable. The functions available are also run internally to build out partial dependence calibration plots. These show you the modeled predicted relationship along with the empirical relationship in the same graph to show the end user how accurate the relationship is what the relationship is (even for categorical variables). Variable importance tables are also available to view along with evaluation calibration plots and boxplots.

Operationalizing models with ease

The scoring process in machine learning is typically straightforward. We have an all-purpose scoring function to score all your supervised machine learning models, text models, and clustering models, regardless of type (mojo or standard model files).

How to install H2O

Follow this link to install H2O if it isn't on your machine already. Install H2O

There are seven categories of functions (currently) in this library I'll go over:

- Automated Supervised Learning
- Automated Unsupervised Learning
- Automated Model Evaluation
- Automated Feature Interpretation
- Automated Cost Sensitive Optimization
- Automated Feature Engineering
- A Few Miscellaneous Functions

Automated Supervised Learning Functions

Functions include:

- `AutoH2OModeler()`
- `AutoH2OScoring()`
- `AutoTS()`
- `AutoRecommender()`
- `AutoRecommenderScoring()`
- `AutoNLS()`

AutoH2OModeler()

The supervised learning functions handle multiple tasks internally. The **AutoH2OModeler** function can build any number of H2O models, automatically compare hyper-parameter tuned versions to baseline versions, selecting a winner, saving the model evaluation and feature interpretation metrics / graphs, along with storing models and their metadata to refer to them later in a production setting.

The models available include:

- Gradient Boosting Machines
- LightGBM (Linux only)
- Distributed Random Forest
- XGBoost (Linux only)
- Deeplearning
- AutoML (for Windows users XGBoost and LightGBM are not available)

For Windows users (Mac?), XGBoost is not available and therefore neither is LightGBM (XGBoost and LightGBM are not utlized in AutoML model selection with Windows).

AutoH2OScoring()

This function is the complement of the **AutoH2OModeler**, **AutoKMeans**, and **AutoWord2VecModeler** functions. Specify which rows of your model metadata collection file to run and **AutoH2OScoring** will return a list of predicted values, where each element of the list is a set of predicted values from the model it ran. For the **AutoH2OModeler** you will generate a file called grid_tuned_paths.Rdata which contains the path to your models (among other items) that you can pass along to the **AutoH2OScoring** function to automatically score your models. For the **AutoKMeans** you will generate a file called KMeansModelFile.Rdata which contains the paths to the models for scoring your GLRM and KMeans models. For the **AutoWord2VecModeler** you will generate a file called StoreFile.Rdata which contains the paths to your word2vec models for scoring.

In total, the AutoH2OScoring function can score

- Regression models
- Quantile regression Models
- Binary classification Models
- Multinomial classification Models
- Multioutcome multinomial classification models
- Generalized low rank dimensionality reduction models
- KMeans clustering models
- Word2vec models

AutoTS()

Another automated supervised learning function we have is an automated time series modeling function (AutoTS) that optimally builds out seven types of time series forecasting models, compares them on holdout data, picks a winner, rebuilds the winner on full data, and generates the forecasts for the number of desired periods. The intent is to make these processes fast, easy, and of high quality. Every model makes use of the optimal settings of their paramters to give them the best chance of being the best. Each model uses a Box-Cox transformation on the target variable and all predictions are back-transformed. It also compares model-based frequency determination versus user-supplied (for the TimeUnit argument) along with the option to have imputation and outlier replacement conducted.

The competing models include:

- DSHW (Double Seasonal Holt Winters)
- ARFIMA (Autoregressive Fractional Integrated Moving Average)
- ARIMA (Autoregressive Integrated Moving Average)
- ETS (Exponential Smoothing and Holt Winters)
- TBATS (Exonential Smoothing State Space Model with Box-Cox Transformation, ARMA Errors, Trend and Seasonal Components)
- TSLM (Time Series Linear Model)
- NN (Autoregressive Neural Network)

AutoRecommender()

This function builds out several variations of collaborative filtering models on a binary ratings matrix. To automatically build the binary ratings matrix, see **RecomDataCreate**.

The competing models include:

- RandomItems
- PopularItems
- UserBasedCF
- ItemBasedCF
- AssociationRules

AutoRecommenderScoring()

This function will automatically score your winning model. Simply feed in your data and the winning model returned from the **AutoRecommender** function and this function will generate a table of several recommended products (by rank) for each entity.

AutoNLS()

This automated supervised learning function builds nonlinear regression models for a more niche set of tasks. It's set up to generate interpolation predictions, such as smoothing cost curves for optimization tasks. It returns the interpolated data, the winning model name, the model object, and the evaluation metrics table.

The competing models include:

- Asymptotic
- Asymptotic through origin
- Asymptotic with offset
- Bi-exponential
- Four parameter logistic
- Three parameter logistic
- Gompertz
- Michal Menton
- Weibull
- Polynomial regression or monotonic regression

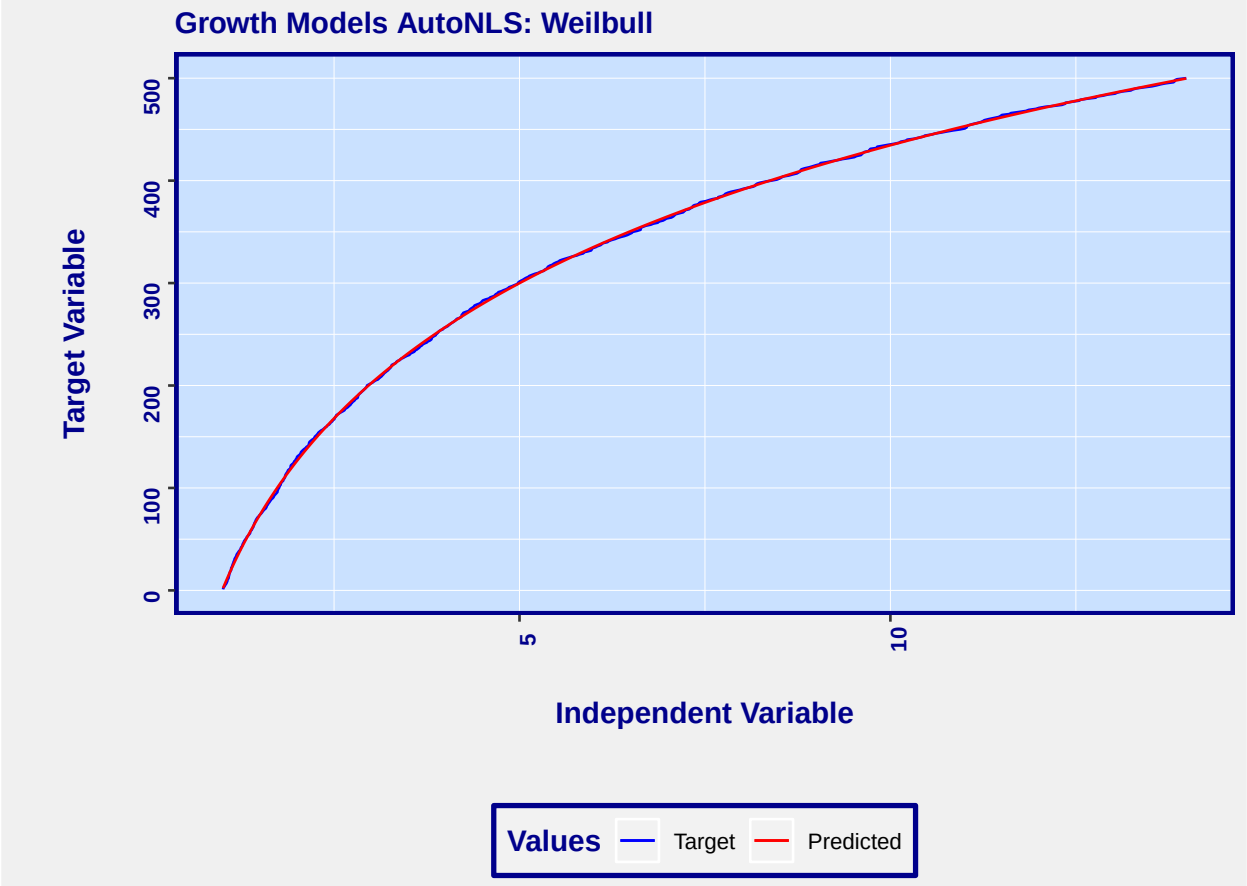
Example of AutoNLS()

Find more demos at **https://www.remyxcourses.com/course?courseid=intro-to-remixautoml-in-r**

```
library(RemixAutoML)

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

```
      data.table::set(data,
                      i = i,
                      j = 2L,
                      value = var * (1 + runif(1) / 100))
    }
  }

  # Add jitter to Target
  data[, Target := jitter(Target,
                          factor = 0.50)]

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

  # Build models
  data11 <- AutoNLS(
    data = data,
    y = "Target",
    x = "Variable",
    monotonic = TRUE
  )

  # Join predictions to source data
  data2 <- merge(
    data1,
    data11$PredictionData,
    by = "Variable",
    all = FALSE
  )

# Plot output
ggplot2::ggplot(data2, ggplot2::aes(x = Variable)) +
  ggplot2::geom_line(ggplot2::aes(y = data2[["Target.x"]],
                                  color = "Target")) +
  ggplot2::geom_line(ggplot2::aes(y = data2[["Target.y"]],
                                  color = "Predicted")) +
  RemixAutoML::ChartTheme(Size = 12) +
  ggplot2::ggtitle(paste0("Growth Models AutoNLS: ",
                          data11$ModelName)) +
  ggplot2::ylab("Target Variable") +
  ggplot2::xlab("Independent Variable") +
  ggplot2::scale_colour_manual("Values",
                               breaks = c("Target",
                                          "Predicted"),
                               values = c("red",
                                          "blue"))

# Print model makeup and evaluation metrics
summary(data11$ModelObject)
#>
#> Formula: Target ~ SSweibull(Variable, Asym, Drop, lrc, pwr)
#>
#> Parameters:
```

```
#>        Estimate Std. Error t value Pr(>|t|)
#> Asym 1265.85129   85.12954   14.87   <2e-16 ***
#> Drop 2316.77479  167.16341   13.86   <2e-16 ***
#> lrc    -0.50130    0.01139  -44.02   <2e-16 ***
#> pwr     0.22840    0.01619   14.11   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 1.646 on 496 degrees of freedom
#>
#> Number of iterations to convergence: 0
#> Achieved convergence tolerance: 4.19e-07
data11$EvaluationMetrics
#>        ModelName MeanAbsError
#> 1:      Weilbull     1.360774
#> 2:      Gompertz    16.478122
#> 3: Michal_Menton    17.963190
#> 4:      Logistic    22.189842
#> 5:          Poly    82.627968
```



Example of AutoTS()

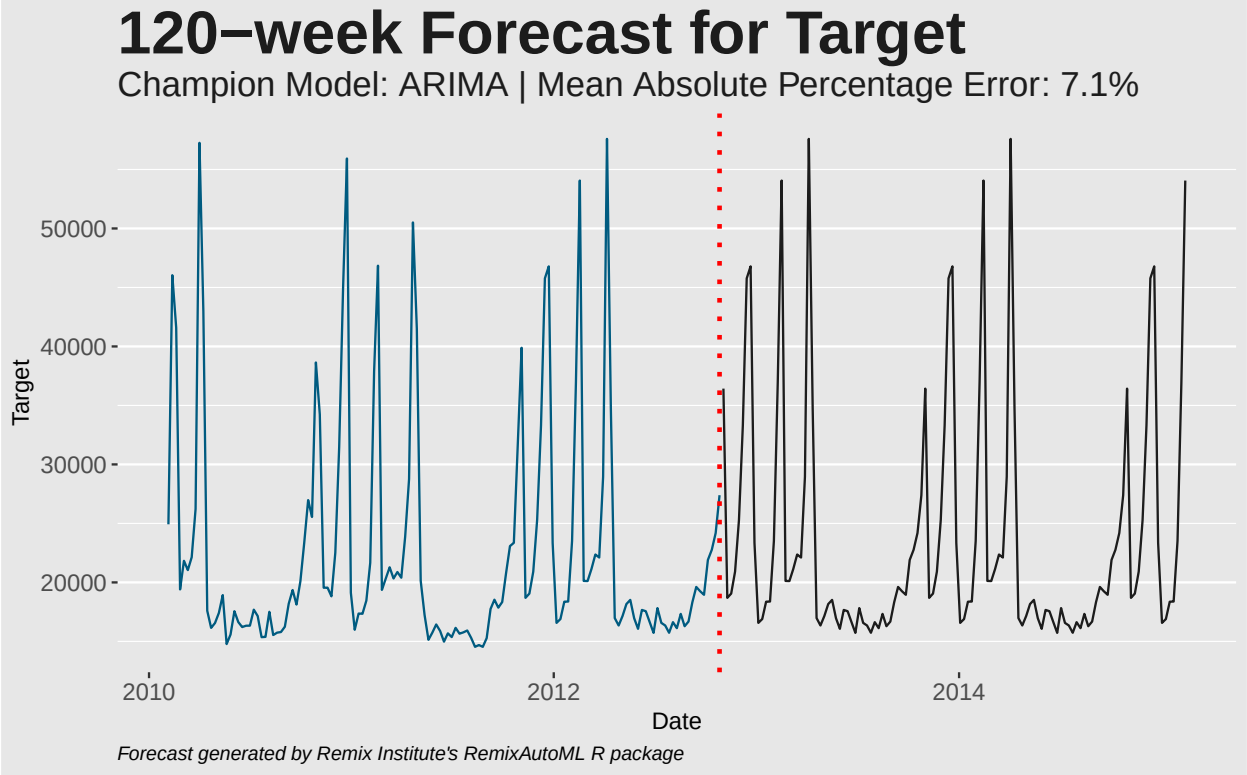
Find more demos at **https://www.remyxcourses.com/course?courseid=intro-to-remixautoml-in-r**

```r
library(RemixAutoML)

# Load Walmart data
data <- data.table::fread(file = "./WalmartStore1Dept1.csv")

# Names of data columns
names(data)
#> [1] "Date"         "Weekly_Sales"

# Build models and generate forecasts
output <- RemixAutoML::AutoTS(data,
                              TargetName     = "Weekly_Sales",
                              DateName       = "Date",
                              FCPeriods      = 120,
                              HoldOutPeriods = 12,
                              TimeUnit       = "week",
                              Lags           = 5,
                              SLags          = 1,
                              NumCores       = 4,
                              SkipModels     = NULL,
                              StepWise       = TRUE,
                              TSClean        = TRUE)
#> [1] "DSHW FITTING"
#> [1] "ARFIMA FITTING"
#> [1] "ARIMA FITTING"
#> [1] "ETS FITTING"
#> [1] "TBATS FITTING"
#> [1] "TSLM FITTING"
#> [1] "NNet FITTING"
#> [1] "NNet Iteration: 1"
#> [1] "NNet Iteration: 2"
#> [1] "NNet Iteration: 3"
#> [1] "NNet Iteration: 4"
#> [1] "NNet Iteration: 5"
#> [1] "NNet 2 Iteration: 1"
#> [1] "NNet 2 Iteration: 2"
#> [1] "NNet 2 Iteration: 3"
#> [1] "NNet 2 Iteration: 4"
#> [1] "NNet 2 Iteration: 5"
#> [1] "NNet 3 Iteration: 1"
#> [1] "NNet 3 Iteration: 2"
#> [1] "NNet 3 Iteration: 3"
#> [1] "NNet 3 Iteration: 4"
#> [1] "NNet 3 Iteration: 5"
#> [1] "NNet 4 Iteration: 1"
#> [1] "NNet 4 Iteration: 2"
#> [1] "NNet 4 Iteration: 3"
#> [1] "NNet 4 Iteration: 4"
#> [1] "NNet 4 Iteration: 5"
#> [1] "FIND WINNER"
#> [1] "GENERATE FORECASTS"
#> [1] "FULL DATA ARIMA FITTING"
```

```r
# Print the evaluation metric and model makeup
knitr::kable(output$EvaluationMetrics)
```

ModelName	MeanResid	MeanPercError	MAPE	ID
ARIMA	609.55	0.04692	0.07130	1
TBATS_TSC	1574.01	0.09270	0.11628	2
TBATS	688.19	0.03370	0.12458	3
TBATS_ModelFreq	688.19	0.03370	0.12458	4
TBATS_ModelFreqTSC	688.19	0.03370	0.12458	5
ARFIMA_TSC	431.74	0.02090	0.12866	6
ARIMA_ModelFreq	-1759.54	-0.08468	0.13118	7
ARIMA_ModelFreqTSC	-1759.54	-0.08468	0.13118	8
NN	2446.88	0.13449	0.14165	9
ARIMA_TSC	2121.11	0.14593	0.14820	10
ARFIMA	-2074.51	-0.09640	0.14925	11
ARFIMA_ModelFreq	-2074.51	-0.09640	0.14925	12
ARFIMA_ModelFreqTSC	-2074.51	-0.09640	0.14925	13
TSLM_ModelFreqTSC	-1306.83	-0.05191	0.17996	14
TSLM_TSC	2526.50	0.17461	0.18285	15
ETS	3271.82	0.19669	0.20523	16
ETS_ModelFreq	3271.88	0.19670	0.20523	17
ETS_ModelFreqTSC	3271.88	0.19670	0.20523	18
DSHW_ModelFreq	3634.13	0.22601	0.23281	19
DSHW_ModelFreqTSC	3634.13	0.22601	0.23281	20
NN_ModelFreq	3804.26	0.23667	0.23787	21
NN_ModelFreqTSC	3840.37	0.23990	0.24292	22
NN_TSC	19899.98	19898.97667	19898.97667	23

```r
summary(output$TimeSeriesModel)
#> Series: dataTSTrain
#> ARIMA(0,0,1)(0,1,0)[52]
#> Box Cox transformation: lambda= TRUE
#>
#> Coefficients:
#>          ma1
#>       0.6695
#> s.e.  0.0719
#>
#> sigma^2 estimated as 52502546:  log likelihood=-937.74
#> AIC=1879.49   AICc=1879.62   BIC=1884.51
#>
#> Training set error measures:
#>                     ME     RMSE      MAE       MPE     MAPE      MASE
#> Training set -8.656056 5748.353 2431.659 -1.819206 9.799424 0.5835215
#>                    ACF1
#> Training set 0.05599928
```



Automated Unsupervised Learning Functions

The suite of functions in this category currently handle optimized row-clustering and anomaly detection. For the row-clustering, we utilize H2O's Generalized Low Rank Model and their KMeans algorithm, with hyper-parameter tuning for both. The function automatically adds the clusters to your data and can save the models for scoring new data with the **AutoH2OScoring** function. We have a few others currently in development and will release those when they are complete. The anomaly detection functions we have currently are for time series applications. We have a control chart methodology version that lets you build upper and lower confidence bounds by up to two grouping variables along with a time series modeling version. The clustering function and the control chart method function update your data set that you feed in with new columns that store the clusterID or anomaly information. The time series function updates your data, supplies you with the final time series model built, and a data.table that only contains anomalies.

Functions include:

- `GenTSAnomVars()`
- `ResidualOutliers()`
- `AutoKMeans()`

Demo of ResidualOutliers()

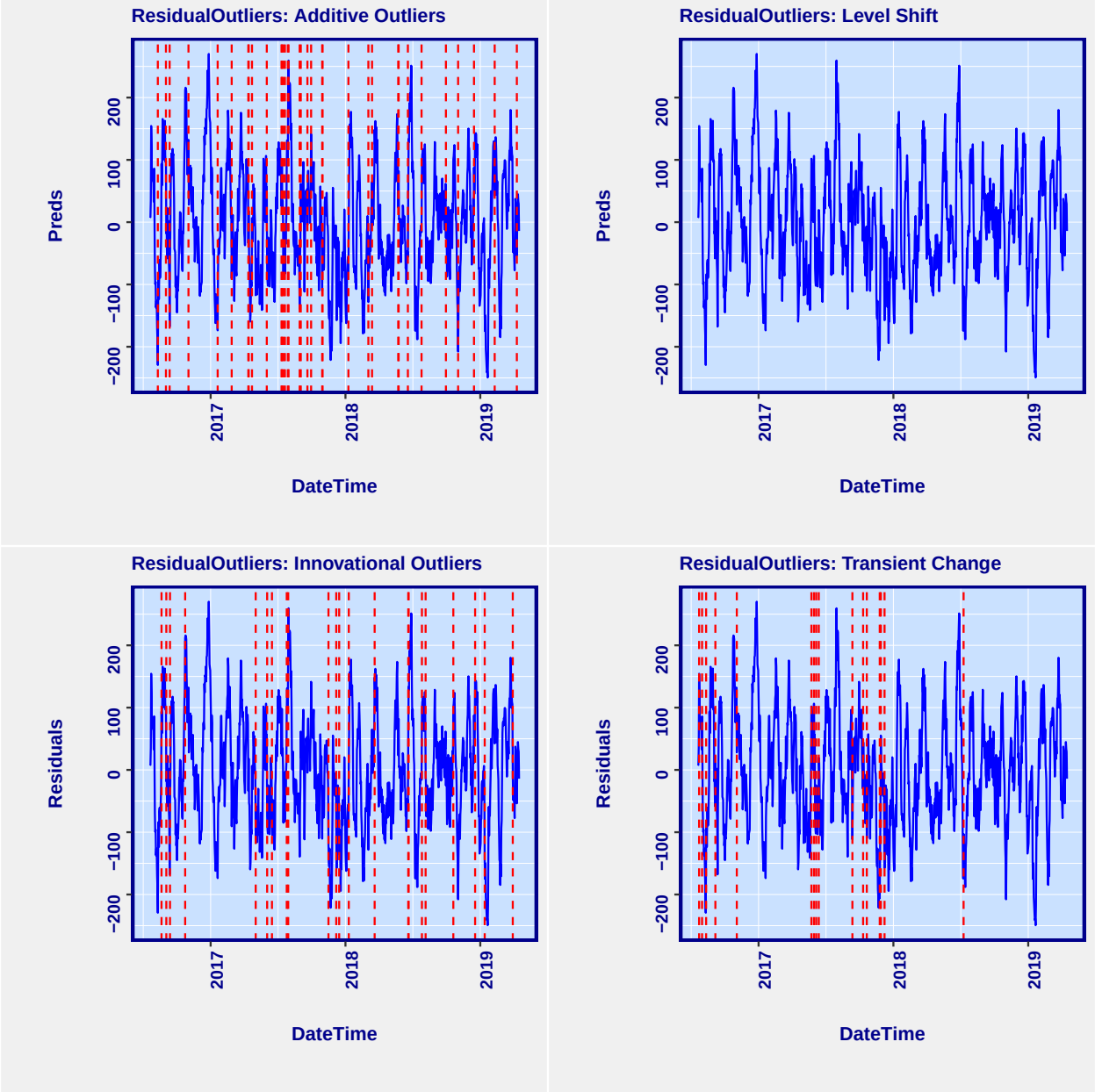
Find more demos at **https://www.remyxcourses.com/course?courseid=intro-to-remixautoml-in-r**

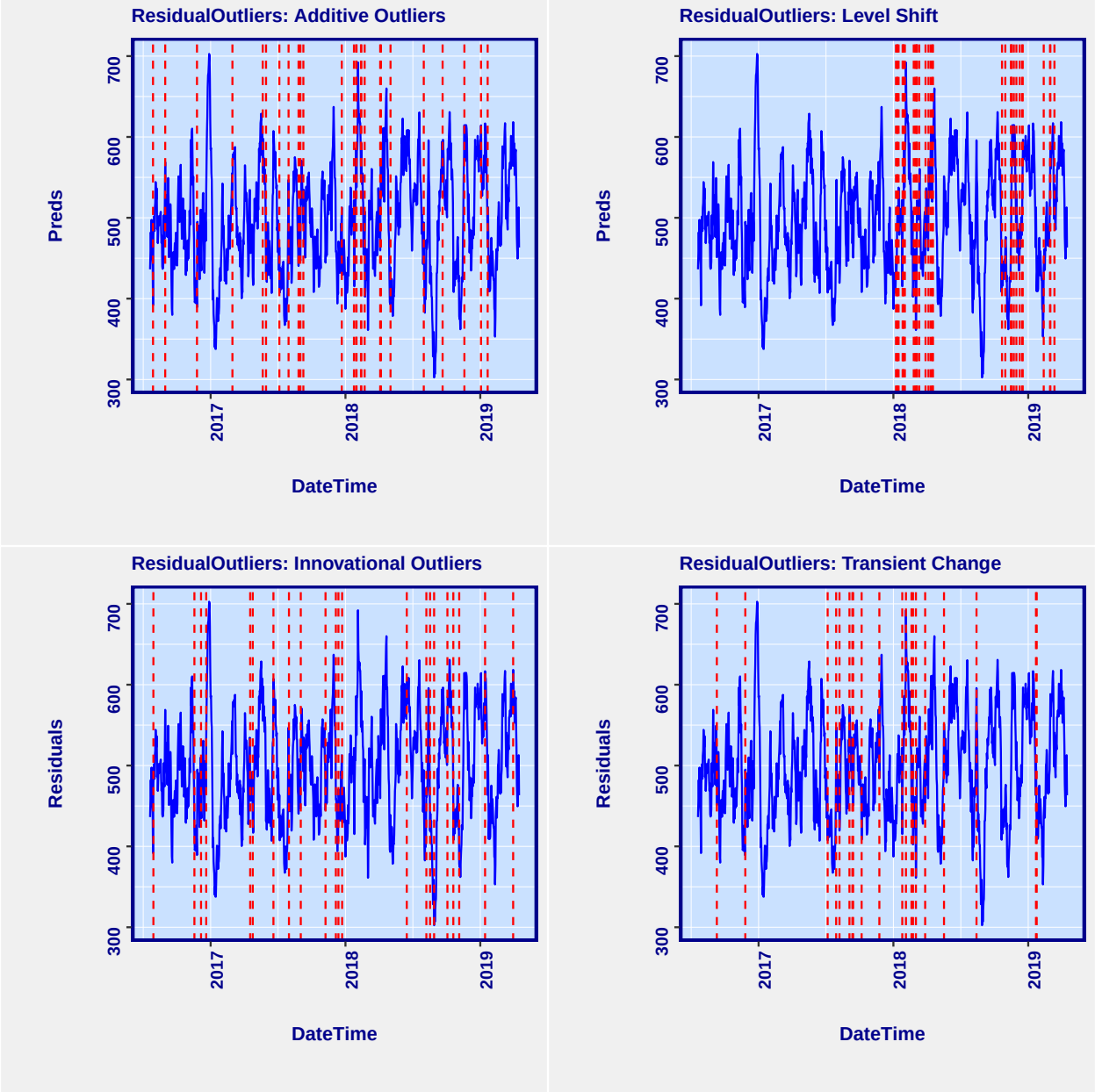
```
# Run on (Target - Predicted)
library(RemixAutoML)
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
```

```
                              Target = as.numeric(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)]
data[, Predicted := as.numeric(stats::filter(rnorm(1000,
                                                   mean = 50,
                                                   sd = 20),
                                             filter=rep(1,10),
                                             circular=TRUE))]

# Run function and collect results
stuff <- ResidualOutliers(data = data,
                          DateColName = "DateTime",
                          TargetColName = "Target",
                          PredictedColName = "Predicted",
                          TimeUnit = "day",
                          maxN = 5,
                          tstat = 2)
data     <- stuff$FullData
model    <- stuff$ARIMA_MODEL
outliers <- data[type != "<NA>"]

# Create Plots
p1 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes(y = Preds),
                     color = "blue") +
  RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "AO", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "AO"][["DateTime"]]),
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Additive Outliers")

p2 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes( y = Residuals),
                     color = "blue") +
 RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "IO", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "IO"][["DateTime"]]),
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Innovational Outliers")

p3 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes(y = Preds),
                     color = "blue") +
  RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "LS", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "LS"][["DateTime"]]),
```

```
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Level Shift")

p4 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes( y = Residuals),
                     color = "blue") +
  RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "TC", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "TC"][["DateTime"]]),
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Transient Change")

# Print plots
RemixAutoML::multiplot(plotlist = list(p1,p2,p3,p4), cols = 2)

# Run on Target data
data <- data.table::data.table(DateTime = as.Date(Sys.time()),
                               Target = as.numeric(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)]
data[, Predicted := as.numeric(stats::filter(rnorm(1000,
                                                   mean = 50,
                                                   sd = 20),
                                             filter=rep(1,10),
                                             circular=TRUE))]

# Run function and collect results
stuff    <- ResidualOutliers(data = data,
                             DateColName = "DateTime",
                             TargetColName = "Target",
                             PredictedColName = NULL,
                             TimeUnit = "day",
                             maxN = 5,
                             tstat = 2)
data     <- stuff$FullData
model    <- stuff$ARIMA_MODEL
outliers <- data[type != "<NA>"]

# Create Plots
p11 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes(y = Preds),
                     color = "blue") +
  RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "AO", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "AO"][["DateTime"]]),
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Additive Outliers")
```

```
p22 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes( y = Residuals),
                     color = "blue") +
  RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "IO", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "IO"][["DateTime"]]),
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Innovational Outliers")

p33 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes(y = Preds),
                     color = "blue") +
  RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "LS", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "LS"][["DateTime"]]),
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Level Shift")

p44 <- ggplot2::ggplot(data, ggplot2::aes(x = DateTime)) +
  ggplot2::geom_line(ggplot2::aes( y = Residuals),
                     color = "blue") +
  RemixAutoML::ChartTheme(Size = 10) +
  ggplot2::geom_vline(data = outliers[type == "TC", "DateTime"],
                      ggplot2::aes(xintercept = outliers[
                        type == "TC"][["DateTime"]]),
                      linetype = 8, colour = "red") +
  ggplot2::ggtitle("ResidualOutliers: Transient Change")

# Print plots
RemixAutoML::multiplot(plotlist = list(p11,p22,p33,p44), cols = 2)
```


ResidualOutliers: Additive Outliers
Preds
DateTime
ResidualOutliers: Level Shift
Preds
DateTime
ResidualOutliers: Innovational Outliers
Residuals
DateTime
ResidualOutliers: Transient Change
Residuals
DateTime



Automated Model Evaluation, Feature Interpretation, and Cost Sensitive Optimization Functions

The model evaluation graphs are calibration plots or calibration boxplots. The calibration plots are used for regression (expected value and quantile regession), classification, and multinomial modeling problems. The calibration boxplots are used for regression (expected value and quantile regression). These graphs display both the actual target values and the predicted values, grouped by the number of bins that you specify. The calibration boxplots are useful to understand not only the model bias but also the model variance, across the range of predicted values.

Functions include:

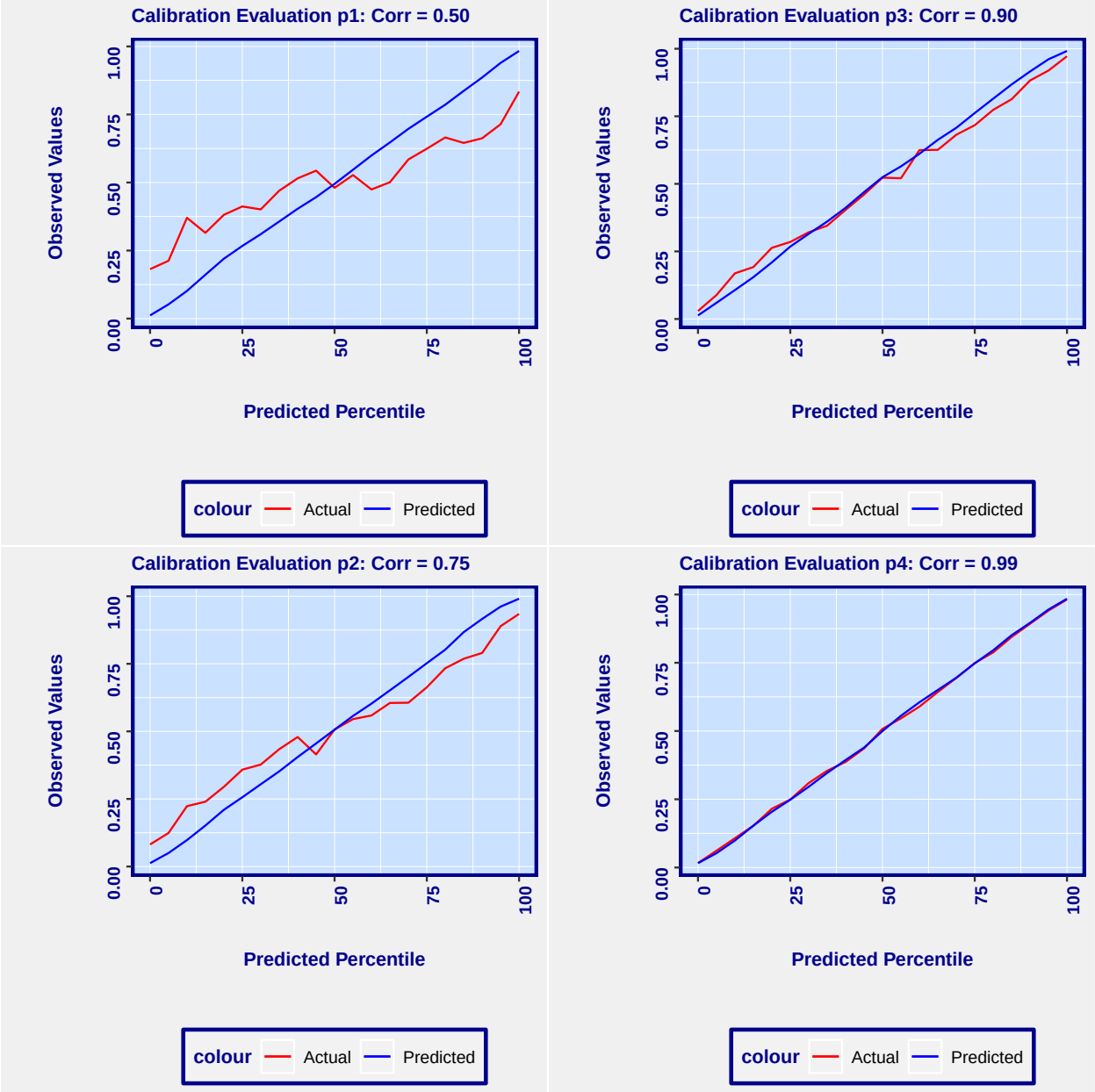
- EvalPlot()

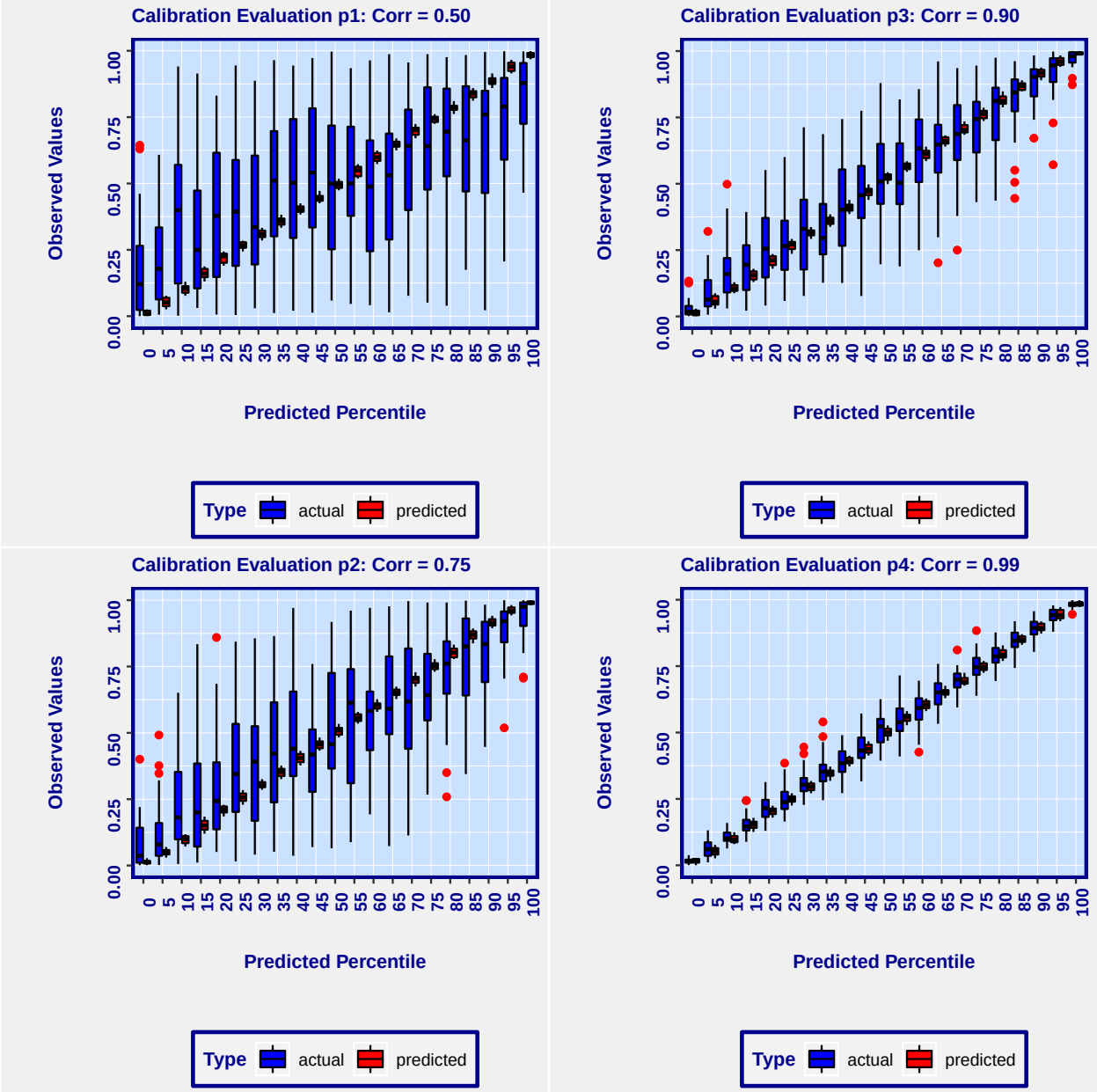
Demo of EvalPlot()

Find more demos at **https://www.remyxcourses.com/course?courseid=intro-to-remixautoml-in-r**

```
library(RemixAutoML)

# Data generator function
dataGen <- function(Correlation = 0.95) {
  Validation <- data.table::data.table(target = runif(1000))
  Validation[, x1 := qnorm(target)]
  Validation[, x2 := runif(1000)]
  Validation[, predict := pnorm(Correlation * x1 +
                                  sqrt(1 - Correlation ^2) * qnorm(x2))]
  return(Validation)
}

# Store data sets
data1 <- dataGen(Correlation = 0.50)
data2 <- dataGen(Correlation = 0.75)
data3 <- dataGen(Correlation = 0.90)
data4 <- dataGen(Correlation = 0.99)

# Generate EvalPlots (calibration)
p1 <- RemixAutoML::EvalPlot(data = data1,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "calibration",
                            PercentileBucket = 0.05,
                            aggrfun = function(x) mean(x,
                                                       na.rm = TRUE))
p1 <- p1 + ggplot2::ggtitle("Calibration Evaluation p1: Corr = 0.50") +
  RemixAutoML::ChartTheme(Size = 10)

p2 <- RemixAutoML::EvalPlot(data = data2,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "calibration",
                            PercentileBucket = 0.05,
                            aggrfun = function(x) mean(x,
                                                       na.rm = TRUE))
p2 <- p2 + ggplot2::ggtitle("Calibration Evaluation p2: Corr = 0.75") +
  RemixAutoML::ChartTheme(Size = 10)

p3 <- RemixAutoML::EvalPlot(data = data3,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "calibration",
                            PercentileBucket = 0.05,
                            aggrfun = function(x) mean(x,
                                                       na.rm = TRUE))
p3 <- p3 + ggplot2::ggtitle("Calibration Evaluation p3: Corr = 0.90") +
```

```
  RemixAutoML::ChartTheme(Size = 10)

p4 <- RemixAutoML::EvalPlot(data = data4,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "calibration",
                            PercentileBucket = 0.05,
                            aggrfun = function(x) mean(x,
                                                       na.rm = TRUE))
p4 <- p4 + ggplot2::ggtitle("Calibration Evaluation p4: Corr = 0.99") +
  RemixAutoML::ChartTheme(Size = 10)
RemixAutoML::multiplot(plotlist = list(p1,p2,p3,p4), cols = 2)

# Generate EvalPlots (boxplots)
p1 <- RemixAutoML::EvalPlot(data = data1,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "boxplot",
                            PercentileBucket = 0.05)
p1 <- p1 + ggplot2::ggtitle("Calibration Evaluation p1: Corr = 0.50") +
  RemixAutoML::ChartTheme(Size = 10)

p2 <- RemixAutoML::EvalPlot(data = data2,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "boxplot",
                            PercentileBucket = 0.05)
p2 <- p2 + ggplot2::ggtitle("Calibration Evaluation p2: Corr = 0.75") +
  RemixAutoML::ChartTheme(Size = 10)

p3 <- RemixAutoML::EvalPlot(data = data3,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "boxplot",
                            PercentileBucket = 0.05)
p3 <- p3 + ggplot2::ggtitle("Calibration Evaluation p3: Corr = 0.90") +
  RemixAutoML::ChartTheme(Size = 10)

p4 <- RemixAutoML::EvalPlot(data = data4,
                            PredictionColName = "predict",
                            TargetColName = "target",
                            GraphType = "boxplot",
                            PercentileBucket = 0.05)
p4 <- p4 + ggplot2::ggtitle("Calibration Evaluation p4: Corr = 0.99") +
  RemixAutoML::ChartTheme(Size = 10)

RemixAutoML::multiplot(plotlist = list(p1,p2,p3,p4), cols = 2)
```


Calibration Evaluation p1: Corr = 0.50
Observed Values
0.00
0.25
0.50
0.75
1.00
0
25
50
75
100
Predicted Percentile
colour
Actual
Predicted
Calibration Evaluation p3: Corr = 0.90
Observed Values
0.00
0.25
0.50
0.75
1.00
0
25
50
75
100
Predicted Percentile
colour
Actual
Predicted
Calibration Evaluation p2: Corr = 0.75
Observed Values
0.00
0.25
0.50
0.75
1.00
0
25
50
75
100
Predicted Percentile
colour
Actual
Predicted
Calibration Evaluation p4: Corr = 0.99
Observed Values
0.00
0.25
0.50
0.75
1.00
0
25
50
75
100
Predicted Percentile
colour
Actual
Predicted



The feature interpretation function graphs are very similar in nature to the model evaluation graphs. They display partial dependence calibration line plots, partial dependence calibration boxplots, and partial dependence calibration bar plots (for factor variables with the ability to limit the number of factors shown with the remainder grouped into "other"). The line graph version is for numerical features and have the ability to aggregate by quantile for quantile regression.

Functions include:

- `ParDepCalPlots()`

The cost sensitive optimizaition functions provide the user the ability to generate utility-optimized thresholds for classification tasks. There are two of these functions: one for generating a single threshold based on the values supplied to your cost confusion matrix outcomes and the second one provides two thresholds, where

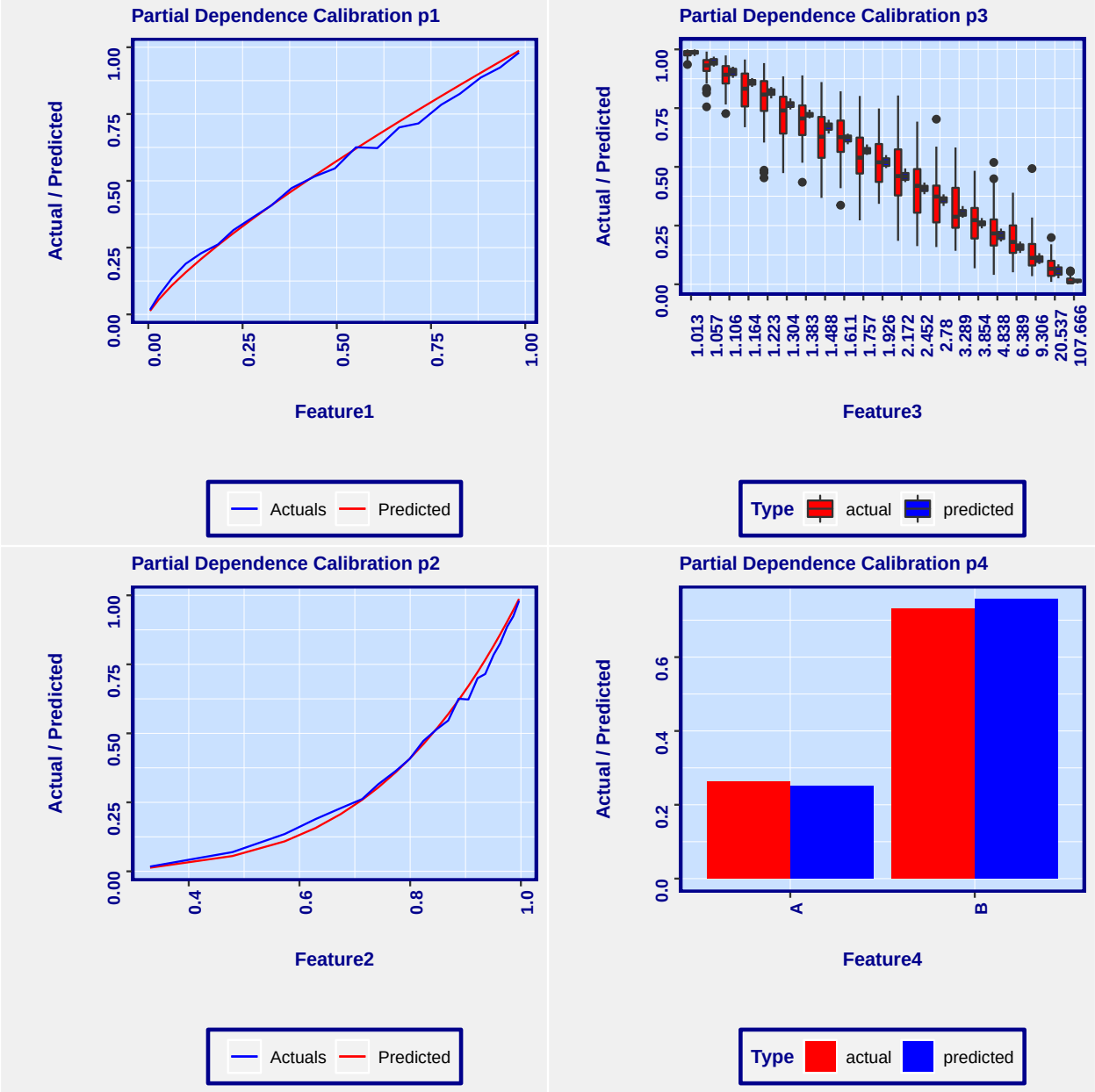
your final predicted classification could be (0|1) and "do something else". With the latter function, you would also need to supply a cost to the "do something else" option.

Demo of ParDepCalPlots() for calibration, boxplots, and barplots

Find more demos at **https://www.remyxcourses.com/course?courseid=intro-to-remixautoml-in-r**

```
library(RemixAutoML)

# Data generator function
dataGen <- function(Correlation = 0.95) {
  Validation <- data.table::data.table(target = runif(1000))
  Validation[, x1 := qnorm(target)]
  Validation[, x2 := runif(1000)]
  Validation[, predict := pnorm(Correlation * x1 +
                                  sqrt(1 - Correlation ^2) * qnorm(x2))]
  Validation[, Feature1 := (pnorm(Correlation * x1 +
                                    sqrt(1 - Correlation ^2) * qnorm(x2)))^1.25]
  Validation[, Feature2 := (pnorm(Correlation * x1 +
                                    sqrt(1 - Correlation ^2) * qnorm(x2)))^0.25]
  Validation[, Feature3 := (pnorm(Correlation * x1 +
                                    sqrt(1 - Correlation ^2) * qnorm(x2)))^(-1)]
  Validation[, Feature4 := pnorm(Correlation * x1 +
                                    sqrt(1 - Correlation ^2) * qnorm(x2))]
  Validation[, Feature4 := ifelse(Feature4 < 0.5, "A",
                                  ifelse(Feature4 < 1, "B",
                                         ifelse(Feature4 < 1.5, "C","D")))]
  return(Validation)
}

# Store data sets
data1 <- dataGen(Correlation = 0.95)

# Generate EvalPlots (calibration)
p1 <- RemixAutoML::ParDepCalPlots(data = data1,
                                  PredictionColName = "predict",
                                  TargetColName = "target",
                                  IndepVar = "Feature1",
                                  GraphType = "calibration",
                                  PercentileBucket = 0.05,
                                  Function = function(x) mean(x,
                                                              na.rm = TRUE),
                                  FactLevels = 10)
p1 <- p1 + ggplot2::ggtitle("Partial Dependence Calibration p1") +
  RemixAutoML::ChartTheme(Size = 10)

p2 <- RemixAutoML::ParDepCalPlots(data = data1,
                                  PredictionColName = "predict",
                                  TargetColName = "target",
                                  IndepVar = "Feature2",
                                  GraphType = "calibration",
                                  PercentileBucket = 0.05,
                                  Function = function(x) mean(x,
```

```
                                                          na.rm = TRUE),
                                    FactLevels = 10)
p2 <- p2 + ggplot2::ggtitle("Partial Dependence Calibration p2") +
  RemixAutoML::ChartTheme(Size = 10)

p3 <- RemixAutoML::ParDepCalPlots(data = data1,
                                    PredictionColName = "predict",
                                    TargetColName = "target",
                                    IndepVar = "Feature3",
                                    GraphType = "boxplot",
                                    PercentileBucket = 0.05,
                                    Function = function(x) mean(x,
                                                          na.rm = TRUE),
                                    FactLevels = 10)
p3 <- p3 + ggplot2::ggtitle("Partial Dependence Calibration p3") +
  RemixAutoML::ChartTheme(Size = 10)

p4 <- RemixAutoML::ParDepCalPlots(data = data1,
                                    PredictionColName = "predict",
                                    TargetColName = "target",
                                    IndepVar = "Feature4",
                                    GraphType = "calibration",
                                    PercentileBucket = 0.05,
                                    Function = function(x) mean(x,
                                                          na.rm = TRUE),
                                    FactLevels = 10)
p4 <- p4 + ggplot2::ggtitle("Partial Dependence Calibration p4") +
  RemixAutoML::ChartTheme(Size = 10)
RemixAutoML::multiplot(plotlist = list(p1,p2,p3,p4), cols = 2)
```



Functions include:

- `threshOptim()`
- `RedYellowGreen()`

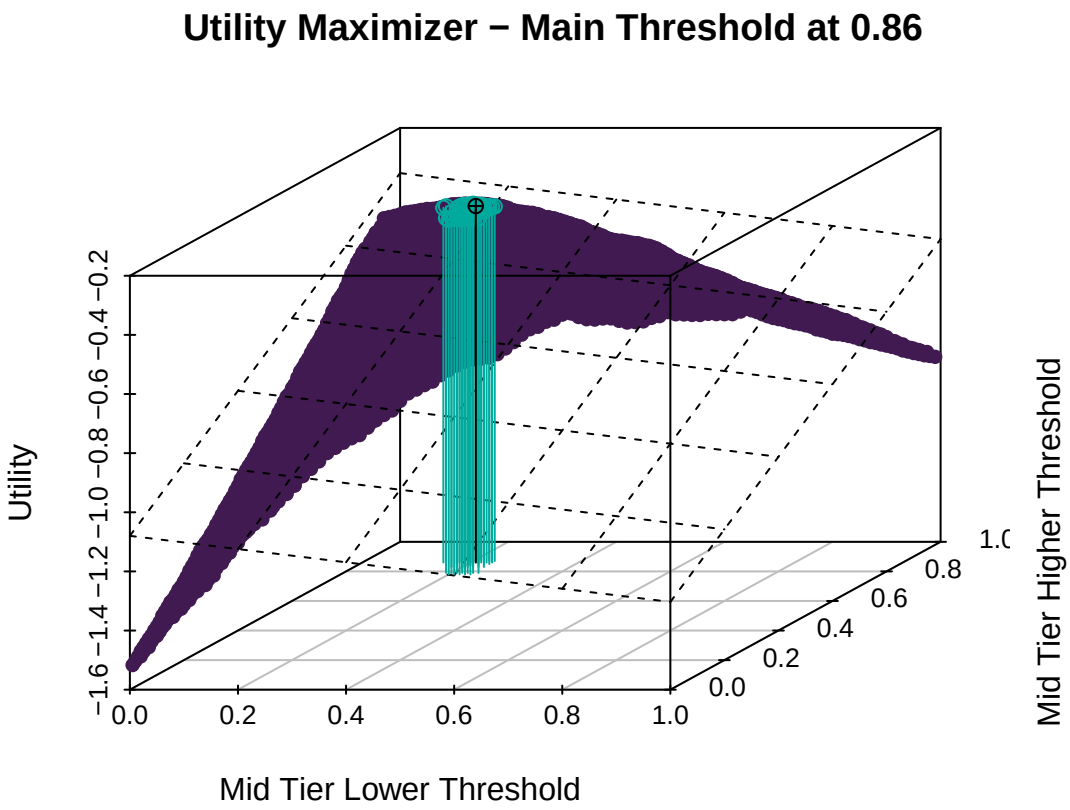
RedYellowGreen Output (simulated data)

Find more demos at **https://www.remyxcourses.com/course?courseid=intro-to-remixautoml-in-r**

```
library(RemixAutoML)
Correl <- 0.70
  aa <- data.table::data.table(target = runif(1000))
```

```
aa[, x1 := qnorm(target)]
aa[, x2 := runif(1000)]
aa[, predict := pnorm(Correl * x1 +
                      sqrt(1 - Correl ^2) *
                      qnorm(x2))]
aa[, target := as.numeric(ifelse(target < 0.5, 0, 1))]
data <- RemixAutoML::RedYellowGreen(
  aa,
  PredictColNumber  = 4,
  ActualColNumber   = 1,
  TruePositiveCost  = 0,
  TrueNegativeCost  = 0,
  FalsePositiveCost = -3,
  FalseNegativeCost = -2,
  MidTierCost       = -0.5,
  Cores             = 1
)

knitr::kable(data[order(-Utility)][1:10])
```

TPP	TNP	FPP	FNP	MTDN	MTC	Threshold	MTLT	MTHT	Utility
0	0	-3	-2	TRUE	-0.5	0.86	0.21	0.86	-0.3942968
0	0	-3	-2	TRUE	-0.5	0.85	0.21	0.85	-0.3945818
0	0	-3	-2	TRUE	-0.5	0.86	0.20	0.86	-0.3947133
0	0	-3	-2	TRUE	-0.5	0.85	0.20	0.85	-0.3952171
0	0	-3	-2	TRUE	-0.5	0.86	0.19	0.86	-0.3953234
0	0	-3	-2	TRUE	-0.5	0.79	0.24	0.79	-0.3953488
0	0	-3	-2	TRUE	-0.5	0.86	0.18	0.86	-0.3959413
0	0	-3	-2	TRUE	-0.5	0.79	0.23	0.79	-0.3960370
0	0	-3	-2	TRUE	-0.5	0.85	0.19	0.85	-0.3960502
0	0	-3	-2	TRUE	-0.5	0.87	0.21	0.87	-0.3961848



Automated Feature Engineering Functions

This suite of functions are what will take your models to the next level. The core functions are the generalized distributed lag and rolling statistics functions. I have four of them.

Functions include:

- `GDL_Feature_Engineering()`
- `DT_GDL_Feature_Engineering()`
- `FAST_GDL_Feature_Engineering()`
- `Scoring_GDL_Feature_Engineering()`

The first three are used for building out lags and rolling statistics from target variables (numeric type; including classification models (0|1) and multinomial models with a little bit of work) and numeric features over your entire data set (no aggregation is done) with the option for creating the rolling statistics on the main variable or the lag1 version of the main variable. You can also compute time between records (by group) and add their lags and rolling statistics as well (really useful for transactional data). They can be generated using a single grouping variable (for multiple grouping variables you can concatenate them) and you can feed in a list of grouping variables to generate them by. The first function (**GDL_**) has the largest variety of rolling statics options but runs the slowest. The second function (**DT_GDL_**) runs the fastest but only generates moving averages. The third function (**FAST_GDL_**) is used for cases where you don't

need to generate the features across the entire data set. Suppose you have a limited number of target variable instance but a rich history of data. You can use the FAST_GDL_ version to create lags and rolling statistics for N number of records previous to each target instance (i.e. not the entire historical data set). The fourth function (**Scoring_GDL_**) is for use in a production setting where you need to generate single instances of the feature set quickly. You basically feed in the same arguments as you used for the other versions and out the other end is the same set of features, identically named.

DT_GDL_Feature_Engineering and Scoring_GDL_Feature_Engineering Demo (simulated data)

Find more demos at **https://www.remyxcourses.com/course?courseid=intro-to-remixautoml-in-r**

```
library(RemixAutoML)

# Build data for feature engineering for modeling
N <- 25116
ModelData <-
  data.table::data.table(GroupVariable = sample(
    x = c(letters,LETTERS,paste0(letters, letters),
          paste0(LETTERS, LETTERS),
          paste0(letters, LETTERS),
          paste0(LETTERS, letters))),
    DateTime = base::as.Date(Sys.time()),
    Target = stats::filter(rnorm(N,
                                 mean = 50,
                                 sd = 20),
                           filter = rep(1, 10),
                           circular = TRUE))
ModelData[, temp := seq(1:161), by = "GroupVariable"][
       , DateTime := DateTime - temp][
         , temp := NULL]
ModelData <- ModelData[order(DateTime)]
ModelData <- RemixAutoML::DT_GDL_Feature_Engineering(
  ModelData,
  lags           = c(seq(1, 5, 1)),
  periods        = c(3, 5, 10, 15, 20, 25),
  statsNames     = c("MA"),
  targets        = c("Target"),
  groupingVars   = "GroupVariable",
  sortDateName   = "DateTime",
  timeDiffTarget = c("Time_Gap"),
  timeAgg        = c("days"),
  WindowingLag   = 1,
  Type           = "Lag",
  Timer          = FALSE,
  SkipCols       = FALSE,
  SimpleImpute   = TRUE)
#> [1] 22

# Build data for feature engineering for scoring
N <- 25116
ScoringData <-
  data.table::data.table(GroupVariable = sample(
```

```
    x = c(letters,LETTERS,paste0(letters, letters),
          paste0(LETTERS, LETTERS),
          paste0(letters, LETTERS),
          paste0(LETTERS, letters))),
    DateTime = base::as.Date(Sys.time()),
    Target = stats::filter(rnorm(N,
                                 mean = 50,
                                 sd = 20),
                           filter = rep(1, 10),
                           circular = TRUE))
ScoringData[, temp := seq(1:161),
      by = "GroupVariable"][, DateTime := DateTime - temp]
ScoringData <- ScoringData[order(DateTime)]

# Use WindowingLag = 1 to build moving averages off of the lag1 Target Variable to eliminate forward le
ScoringData <- RemixAutoML::Scoring_GDL_Feature_Engineering(
  ScoringData,
  lags           = c(seq(1, 5, 1)),
  periods        = c(3, 5, 10, 15, 20, 25),
  statsFUNs      = c(function(x) mean(x, na.rm = TRUE)),
  statsNames     = c("MA"),
  targets        = c("Target"),
  groupingVars   = c("GroupVariable"),
  sortDateName   = c("DateTime"),
  timeDiffTarget = c("Time_Gap"),
  timeAgg        = "days",
  WindowingLag   = 1,
  Type           = "Lag",
  Timer          = FALSE,
  SkipCols       = FALSE,
  SimpleImpute   = TRUE,
  AscRowByGroup  = "temp",
  RecordsKeep    = 1
)

# View some of new features
knitr::kable(ModelData[order(GroupVariable,-DateTime)][1:10,c(3,4,14)])
```

Target	GroupVariable_LAG_1_Target	GroupVariableMA_3_GroupVariable_LAG_1_Target
459.8698	424.6604	520.7788
424.6604	672.6465	544.2602
672.6465	465.0294	494.1031
465.0294	495.1048	469.1118
495.1048	522.1751	453.3144
522.1751	390.0555	415.2033
390.0555	447.7126	472.1582
447.7126	407.8417	482.5355
407.8417	560.9203	496.4438
560.9203	478.8445	513.6590

```
# Ensure names equal
knitr::kable(
  data.table::as.data.table(
    cbind(ModelData_Names = sort(names(ModelData)),
          ScoringData_Names = sort(names(ScoringData[, temp := NULL]))))
```

ModelData_Names	ScoringData_Names
DateTime	DateTime
GroupVariable	GroupVariable
GroupVariable_LAG_1_Target	GroupVariable_LAG_1_Target
GroupVariable_LAG_2_Target	GroupVariable_LAG_2_Target
GroupVariable_LAG_3_Target	GroupVariable_LAG_3_Target
GroupVariable_LAG_4_Target	GroupVariable_LAG_4_Target
GroupVariable_LAG_5_Target	GroupVariable_LAG_5_Target
GroupVariableMA_10_GroupVariable_LAG_1_Target	GroupVariableMA_10_GroupVariable_LAG_1_Target
GroupVariableMA_10_GroupVariableTime_Gap1	GroupVariableMA_10_GroupVariableTime_Gap1
GroupVariableMA_15_GroupVariable_LAG_1_Target	GroupVariableMA_15_GroupVariable_LAG_1_Target
GroupVariableMA_15_GroupVariableTime_Gap1	GroupVariableMA_15_GroupVariableTime_Gap1
GroupVariableMA_20_GroupVariable_LAG_1_Target	GroupVariableMA_20_GroupVariable_LAG_1_Target
GroupVariableMA_20_GroupVariableTime_Gap1	GroupVariableMA_20_GroupVariableTime_Gap1
GroupVariableMA_25_GroupVariable_LAG_1_Target	GroupVariableMA_25_GroupVariable_LAG_1_Target
GroupVariableMA_25_GroupVariableTime_Gap1	GroupVariableMA_25_GroupVariableTime_Gap1
GroupVariableMA_3_GroupVariable_LAG_1_Target	GroupVariableMA_3_GroupVariable_LAG_1_Target
GroupVariableMA_3_GroupVariableTime_Gap1	GroupVariableMA_3_GroupVariableTime_Gap1
GroupVariableMA_5_GroupVariable_LAG_1_Target	GroupVariableMA_5_GroupVariable_LAG_1_Target
GroupVariableMA_5_GroupVariableTime_Gap1	GroupVariableMA_5_GroupVariableTime_Gap1
GroupVariableTime_Gap1	GroupVariableTime_Gap1
GroupVariableTime_Gap2	GroupVariableTime_Gap2
GroupVariableTime_Gap3	GroupVariableTime_Gap3
GroupVariableTime_Gap4	GroupVariableTime_Gap4
GroupVariableTime_Gap5	GroupVariableTime_Gap5
Target	Target

Automated Feature Engineering Functions (Continued)

Functions include:

- `AutoWord2VecModeler()`
- `ModelDataPrep()`
- `DummifyDT()`

The **AutoWord2VecModeler** function converts your text features into numerical vector representations. You supply the function with your data set and all the text column names you want converted, and out the other end you have a data set with all the features merged on. The models can be saved to file and metadata saves their paths for scoring purposes in a production setting. The models built are based on H2O's word2vec algorithm and has done an execellent job at extracting high quality information out of those text columns. The **ModelDataPrep** function is used to prepare your data for modeling with the **AutoH2OModeler** function. It will convert character columns to factors, replace inf values to NA, and impute missing values (both numeric and factor based on supplied values). The **DummifyDT** function will turn your character (or factor) columns into dummy variable columns. You can specify one-hot encoding or not in which you will get N+1 columns for one-hot or N columns otherwise.

Miscellaneous Functions

Functions include:

- `AutoWordFreq()`
- `AutoH2OTextPrepScoring()`
- `RecomDataCreate()`
- `ChartTheme()`
- `RemixTheme()`
- `multiplot()`
- `PrintObjectsSize()`
- `percRank()`

The **AutoWordFreq** function will go through a process of cleaning your text column, doing some other text operations, and ouput a table with word frequencies and a word cloud plot. The **AutoH2OTextPrepScoring** will automatically prepare your text data for scoring. This function is run internally in the **AutoH2OScoring** function but you can utilize it outside for other purposes. The **ChartTheme** and **RemixTheme** functions will turn your ggplots into nicely formatted and colored charts, worthy of presentation. The **multiplot** function are for those who have had a terrible time plotting multiple graphs onto a single image. The **PrintObjectsSize** function is more of a debugging function for inspecting the size of variables in your environment (useful in looping functions). The **percRank** is simply a function to compute the percentile rank of every value in a column of data. **RecomDataCreate** will turn your transactional data set into a binary ratings matrix fast.