Machine Learning Prototyping with R

KRANNERT SCHOOL OF MANAGEMENT
PURDUE UNIVERSITY

Matthew A. Lanham, PhD, CAP
Department of Management/Quantitative Methods

Outline

- Machine Learning
- Prototyping with R/RStudio
- **caret** package
- Example script using caret functions
- Model assessment

Machine Learning

Machine learning is viewed as the development of algorithms that transform data into actionable information, or intelligence.

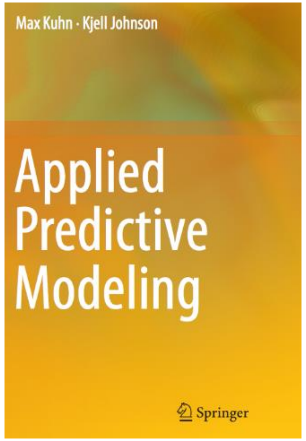
Usually spoken about from the perspective of predictive modeling, but has direct ties to optimization and heuristics

Today, I'll talk about it specifically from the binary classification perspective

caret Package

The **C**lassification **A**nd **RE**gression **T**raining **caret** package was developed by Max Kuhn as a tool to streamline the predictive modeling process for R users.

Max currently works for RStudio. His book published in 2013 and has many nice examples



Additional information here: http://topepo.github.io/caret/index.html

Most of the functionality can be broken down into 5 domains

Data splitting	**Pre-processing**	**Feature selection**	**Model tuning using resampling**	**Variable importance estimation**

Key functions

- **createDataPartition()** function allows one to easily partition their data into training and test sets that are distributed (or imbalanced) similar to one another.
- **upSample(), downSample()** allows you to up or down sample your training data
- **trainControl()** is where you specify how you want to design your run
- **train()** is the bread and butter of what makes the caret package so great. It's a wrapper for essentially every (not all) regression or classification modeling technique
- **predict()** allows you to generate your predictions from a training set or testing set
- **confusionMatrix()** provides a convenient function to assess classification model performance

Business Case

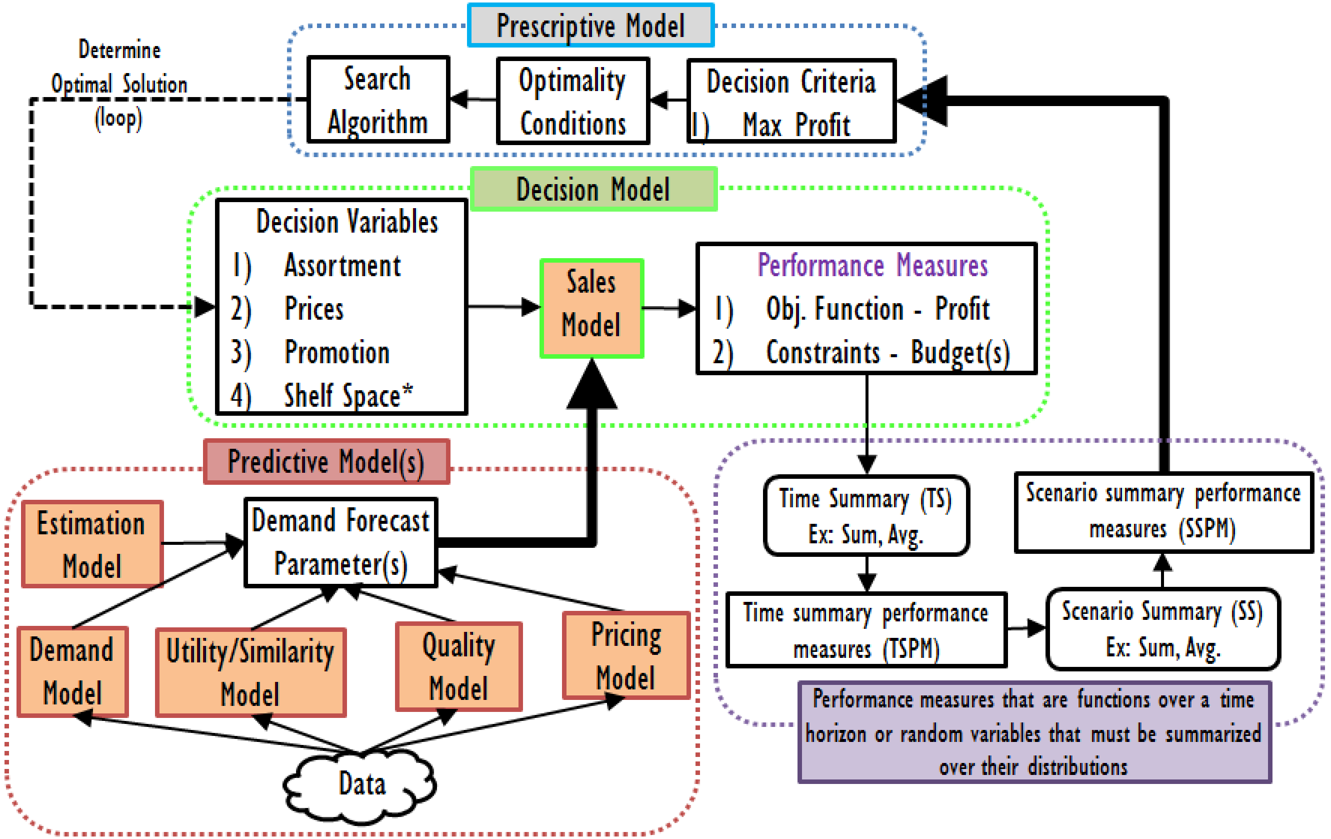
Business Problem

- Service parts are known to exhibit very sparse demand patterns.
- There are often many potential parts one might need, but can never have all of them on hand.
- They must determine the best assortment among the large set while satisfying various constraints (e.g. classical knapsack problem).

Analytics Problem

- Demand is so sparse that each part might be required (or sold) once or twice 90% of the time during an entire planning horizon (here one year).
- Traditional forecasting approaches for predicting highly sparse demand (e.g. Poisson regression, time-series ETS techniques) are often very poor
- Demand estimation could be formulated as a binary classification problem (e.g. 1s = sellers vs 0s = non-sellers) where predicted probabilities provide a rank among competing parts

An Assortment Framework


Prescriptive Model
Determine Optimal Solution (loop)
Search Algorithm
Optimality Conditions
Decision Criteria
1) Max Profit
Decision Model
Decision Variables
1) Assortment
2) Prices
3) Promotion
4) Shelf Space*
Sales Model
Performance Measures
1) Obj. Function - Profit
2) Constraints - Budget(s)
Predictive Model(s)
Estimation Model
Demand Forecast Parameter(s)
Demand Model
Utility/Similarity Model
Quality Model
Pricing Model
Data
Time Summary (TS) Ex: Sum, Avg.
Scenario summary performance measures (SSPM)
Time summary performance measures (TSPM)
Scenario Summary (SS) Ex: Sum, Avg.
Performance measures that are functions over a time horizon or random variables that must be summarized over their distributions

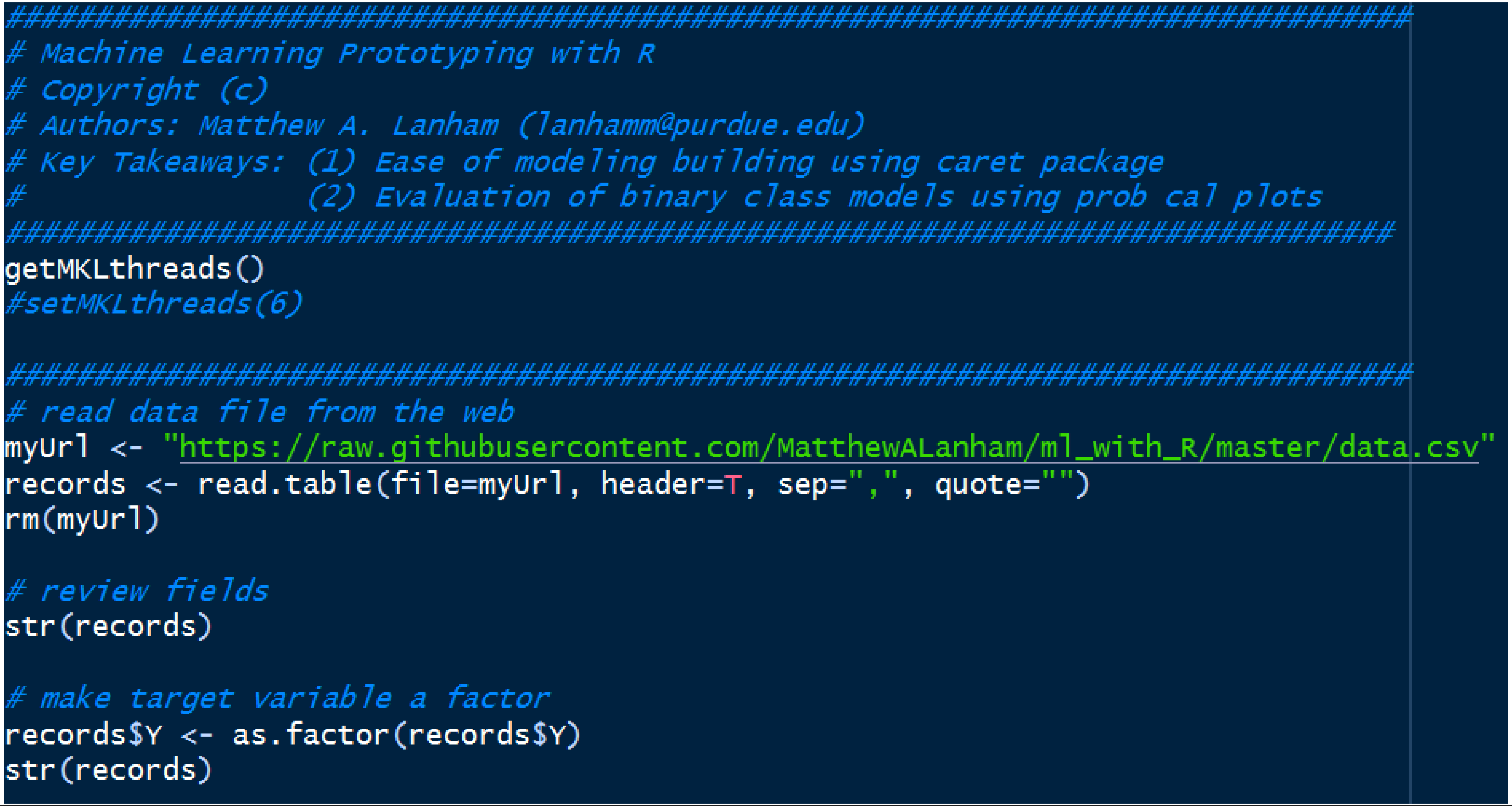
Study focus

- When part data sets are modeled together they are often imbalanced (some more than others)
- Some claim rebalancing your training data so as to achieve a 50/50 balance on the training set will allow the machine to learn without bias toward one class or another (often the majority class).
- However, machine learning theory states that a testing/holdout set should always be representative of the training set
- This catch-22 is what makes this area of study so interesting and challenging for researchers today.

Research Questions

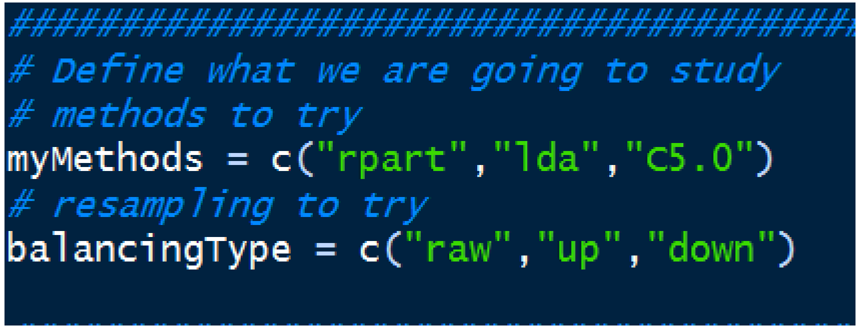
- For various levels of class imbalance which combination of rebalance-machine approach works "best"?
- Are the statistical performance measures (e.g. AUC, accuracy) and business performance measures (e.g. probability calibration plots) in alignment?

Load in data



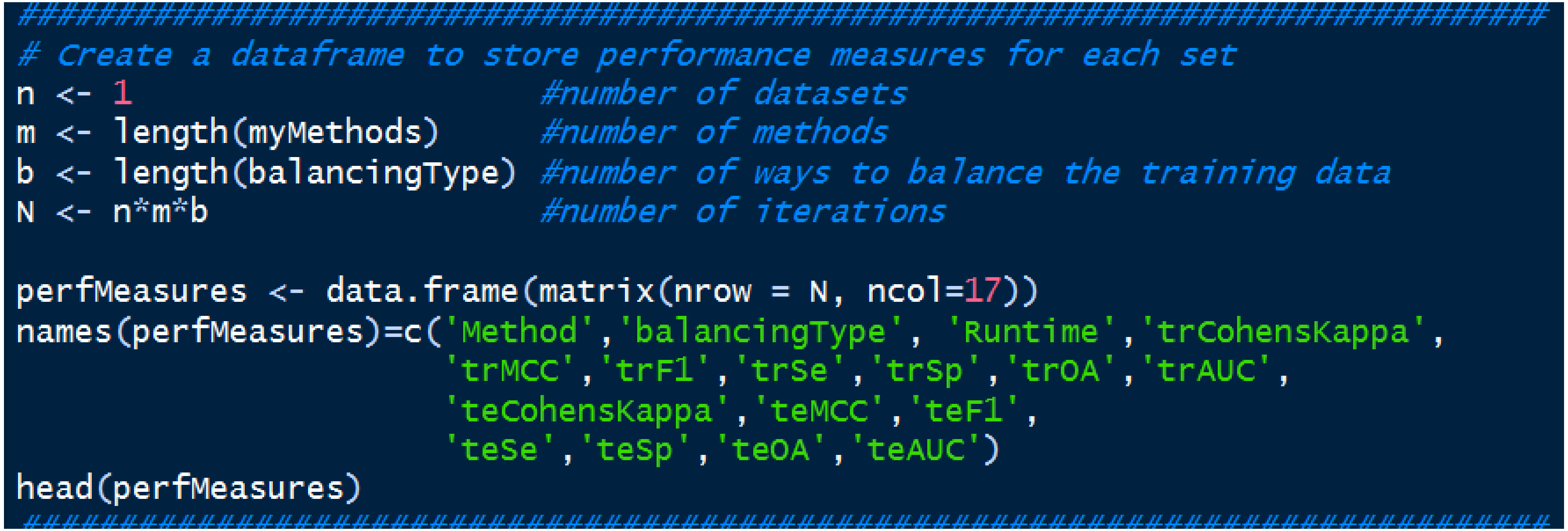
```
#################################################################################
# Machine Learning Prototyping with R
# Copyright (c)
# Authors: Matthew A. Lanham (lanhamm@purdue.edu)
# Key Takeaways: (1) Ease of modeling building using caret package
#                (2) Evaluation of binary class models using prob cal plots
################################################################################
getMKLthreads()
#setMKLthreads(6)

#################################################################################
# read data file from the web
myUrl <- "https://raw.githubusercontent.com/MatthewALanham/ml_with_R/master/data.csv"
records <- read.table(file=myUrl, header=T, sep=",", quote="")
rm(myUrl)

# review fields
str(records)

# make target variable a factor
records$Y <- as.factor(records$Y)
str(records)
```

Specify your models & resampling approaches



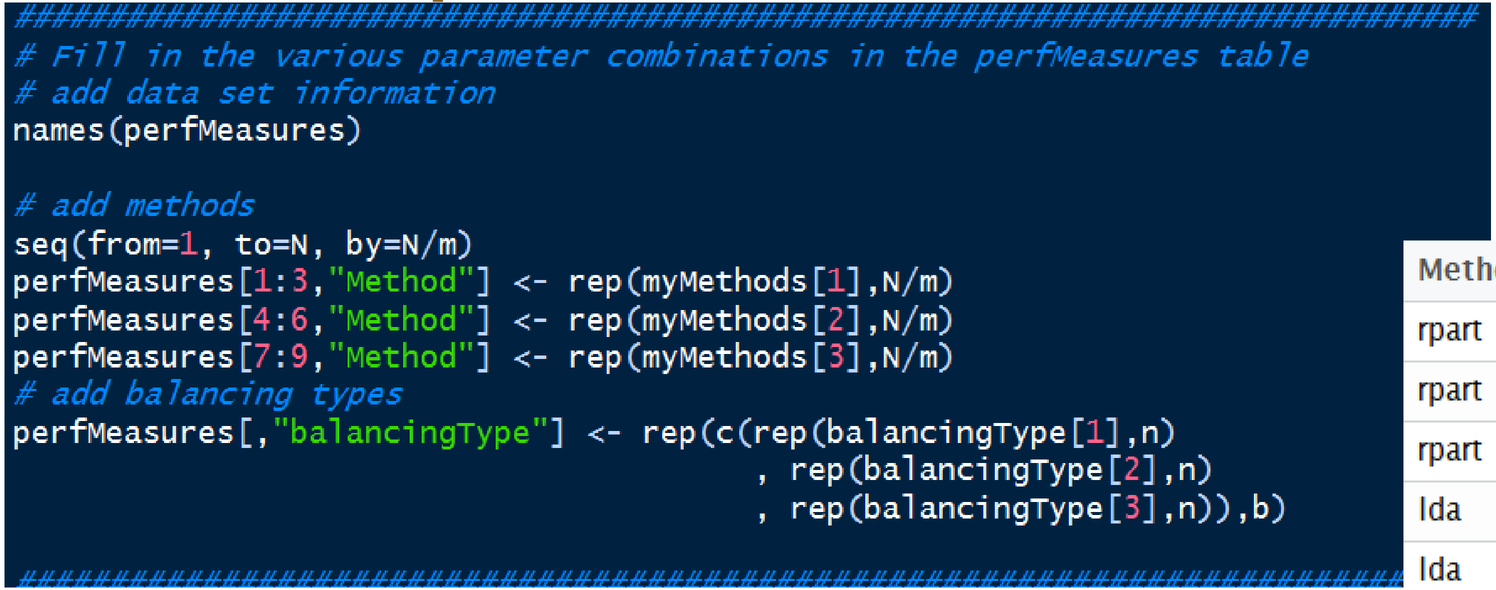
```
#######################################
# Define what we are going to study
# methods to try
myMethods = c("rpart","lda","C5.0")
# resampling to try
balancingType = c("raw","up","down")
```

Create a table to store your stats

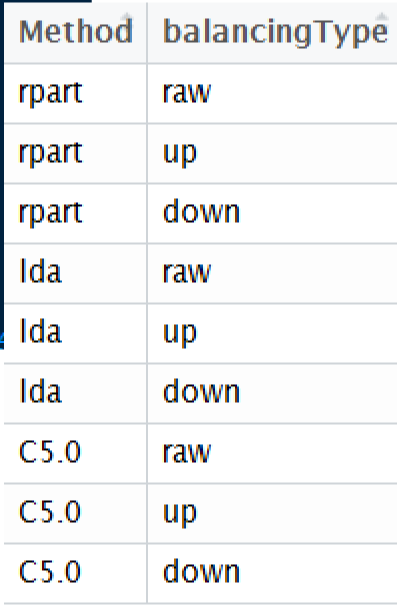


```
###############################################################################
# Create a dataframe to store performance measures for each set
n <- 1                     #number of datasets
m <- length(myMethods)     #number of methods
b <- length(balancingType) #number of ways to balance the training data
N <- n*m*b                 #number of iterations

perfMeasures <- data.frame(matrix(nrow = N, ncol=17))
names(perfMeasures)=c('Method','balancingType', 'Runtime','trCohensKappa',
                      'trMCC','trF1','trSe','trSp','trOA','trAUC',
                      'teCohensKappa','teMCC','teF1',
                      'teSe','teSp','teOA','teAUC')
head(perfMeasures)
###############################################################################
```

Fill in the experimental runs

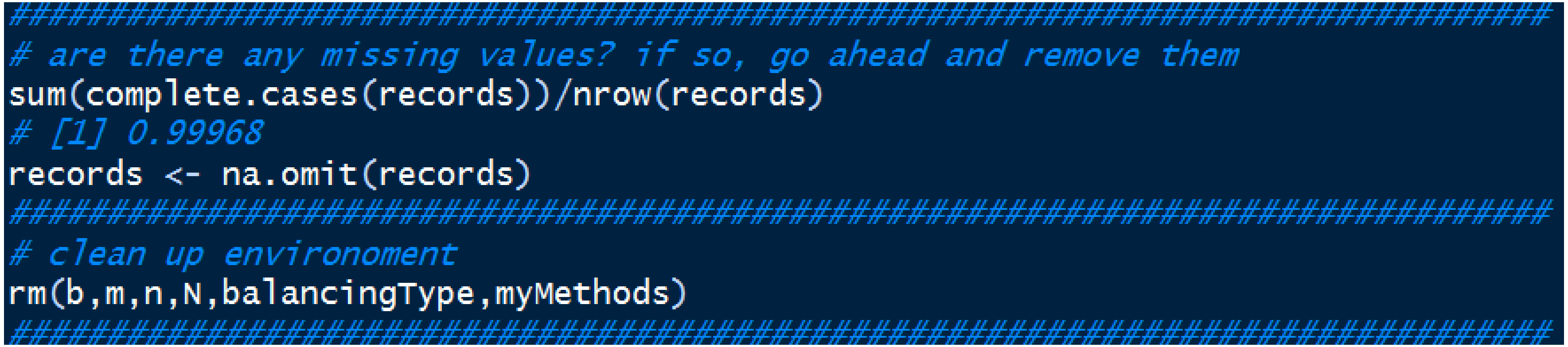


```r
################################################################################
# Fill in the various parameter combinations in the perfMeasures table
# add data set information
names(perfMeasures)

# add methods
seq(from=1, to=N, by=N/m)
perfMeasures[1:3,"Method"] <- rep(myMethods[1],N/m)
perfMeasures[4:6,"Method"] <- rep(myMethods[2],N/m)
perfMeasures[7:9,"Method"] <- rep(myMethods[3],N/m)
# add balancing types
perfMeasures[,"balancingType"] <- rep(c(rep(balancingType[1],n)
                                        , rep(balancingType[2],n)
                                        , rep(balancingType[3],n)),b)

################################################################################
```



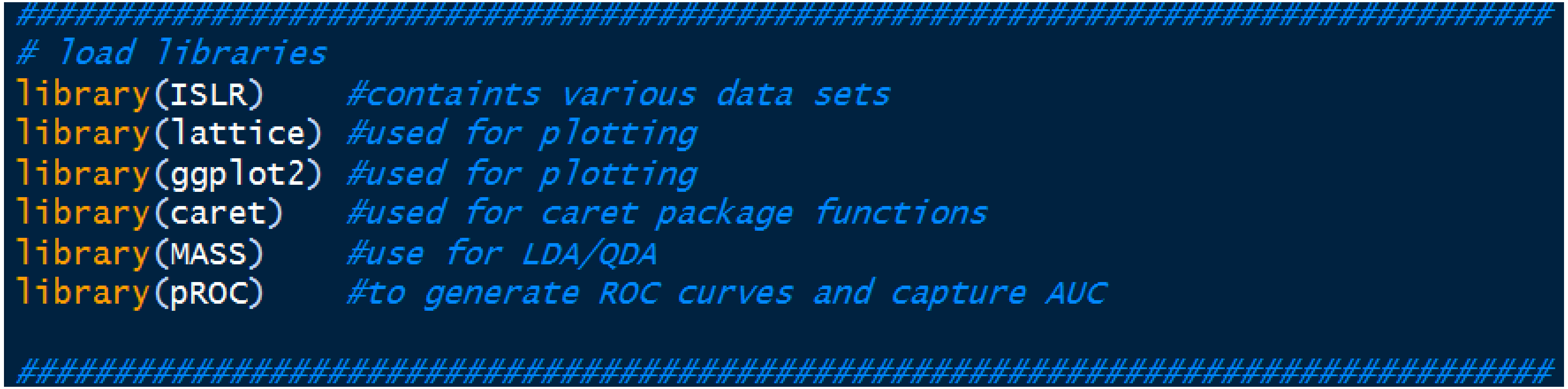
Method	balancingType
rpart	raw
rpart	up
rpart	down
lda	raw
lda	up
lda	down
C5.0	raw
C5.0	up
C5.0	down

Example of removing missing values



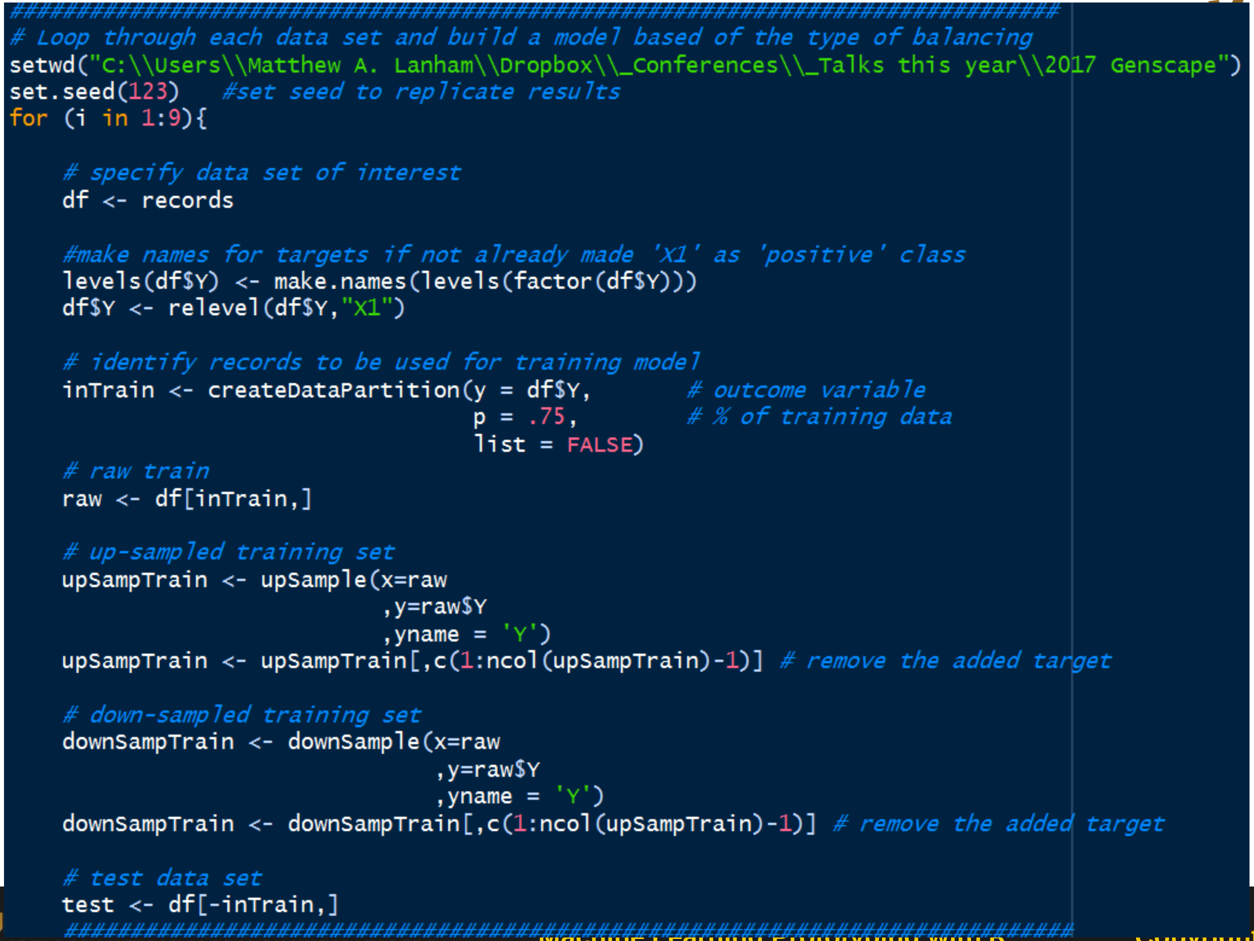
```r
################################################################################
# are there any missing values? if so, go ahead and remove them
sum(complete.cases(records))/nrow(records)
# [1] 0.99968
records <- na.omit(records)
################################################################################
# clean up environoment
rm(b,m,n,N,balancingType,myMethods)
################################################################################
```

Use libraries that have various useful functions



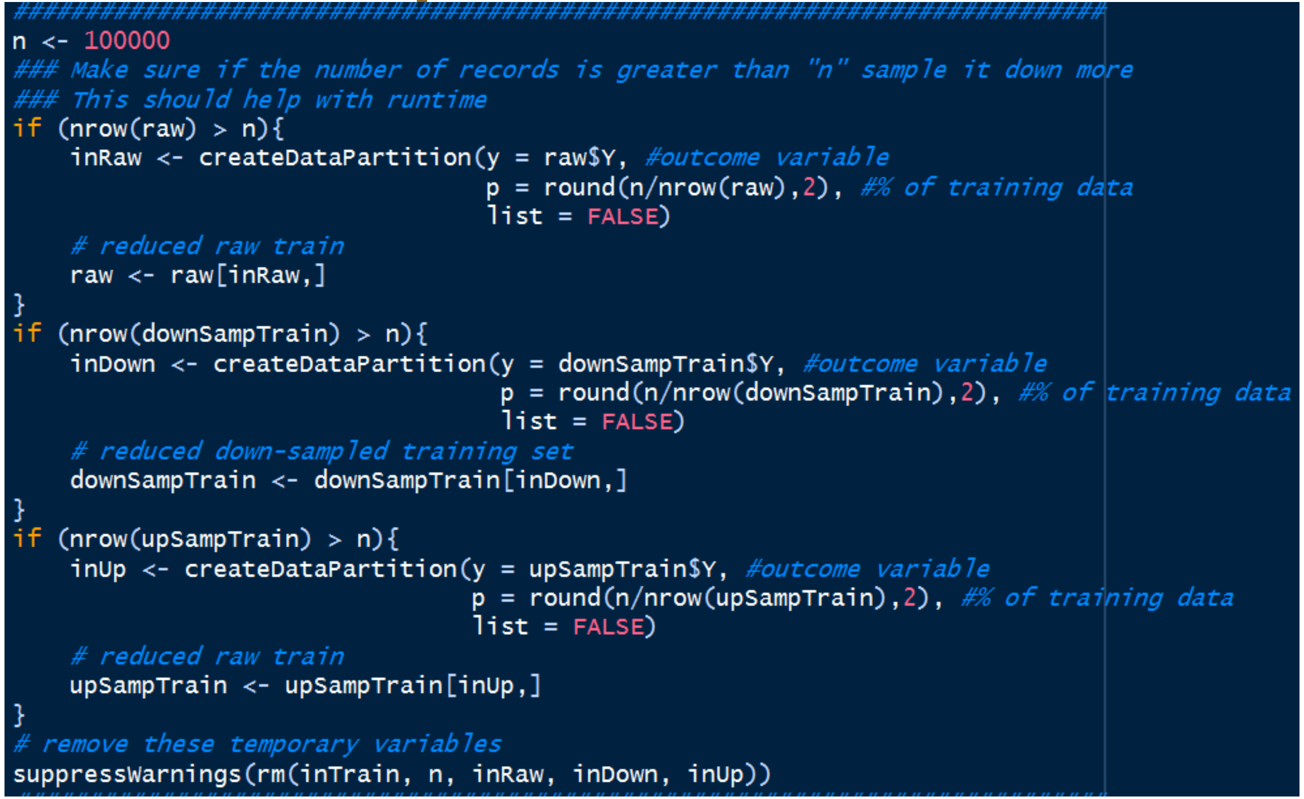
```
##########################################################################################
# load libraries
library(ISLR)     #containts various data sets
library(lattice) #used for plotting
library(ggplot2) #used for plotting
library(caret)   #used for caret package functions
library(MASS)    #use for LDA/QDA
library(pROC)    #to generate ROC curves and capture AUC

##########################################################################################
```

Partition data into different training sets



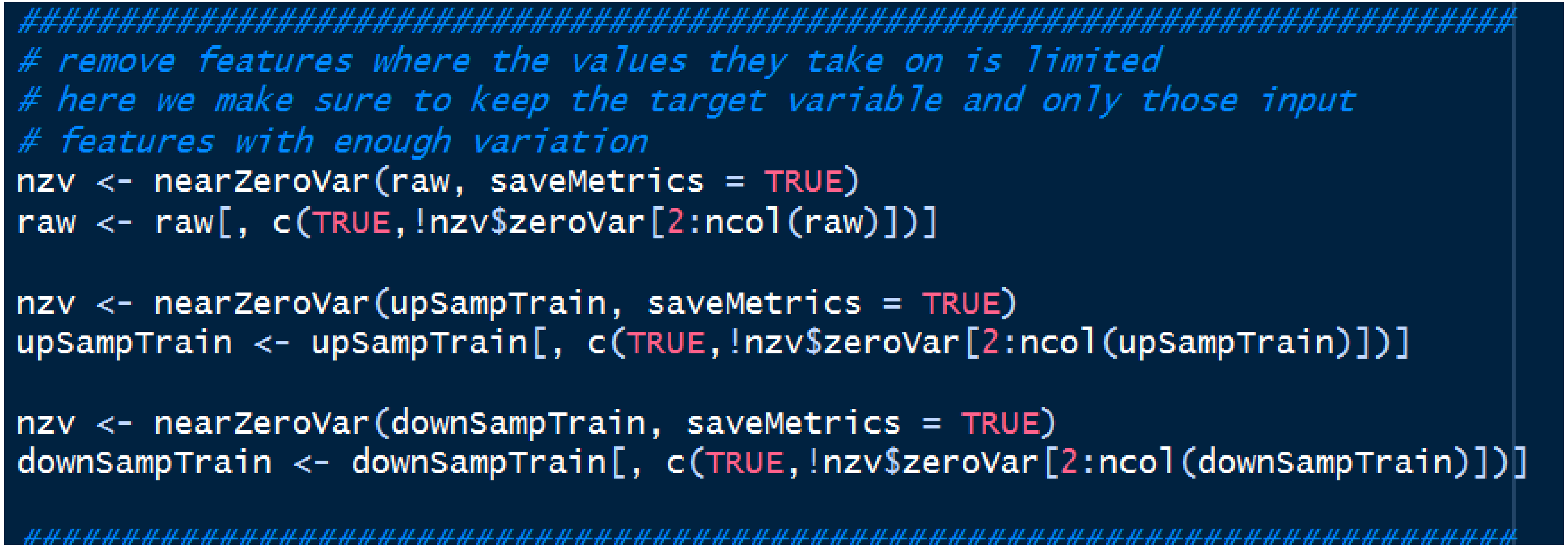
```
################################################################################
# Loop through each data set and build a model based of the type of balancing
setwd("C:\\Users\\Matthew A. Lanham\\Dropbox\\_Conferences\\_Talks this year\\2017 Genscape")
set.seed(123)   #set seed to replicate results
for (i in 1:9){

    # specify data set of interest
    df <- records

    #make names for targets if not already made 'X1' as 'positive' class
    levels(df$Y) <- make.names(levels(factor(df$Y)))
    df$Y <- relevel(df$Y,"X1")

    # identify records to be used for training model
    inTrain <- createDataPartition(y = df$Y,       # outcome variable
                                   p = .75,        # % of training data
                                   list = FALSE)
    # raw train
    raw <- df[inTrain,]

    # up-sampled training set
    upSampTrain <- upSample(x=raw
                            ,y=raw$Y
                            ,yname = 'Y')
    upSampTrain <- upSampTrain[,c(1:ncol(upSampTrain)-1)] # remove the added target

    # down-sampled training set
    downSampTrain <- downSample(x=raw
                                ,y=raw$Y
                                ,yname = 'Y')
    downSampTrain <- downSampTrain[,c(1:ncol(upSampTrain)-1)] # remove the added target

    # test data set
    test <- df[-inTrain,]
    ##############################################################################
```

Depending on the machine or algorithm using, I sometimes will cap the number of obs



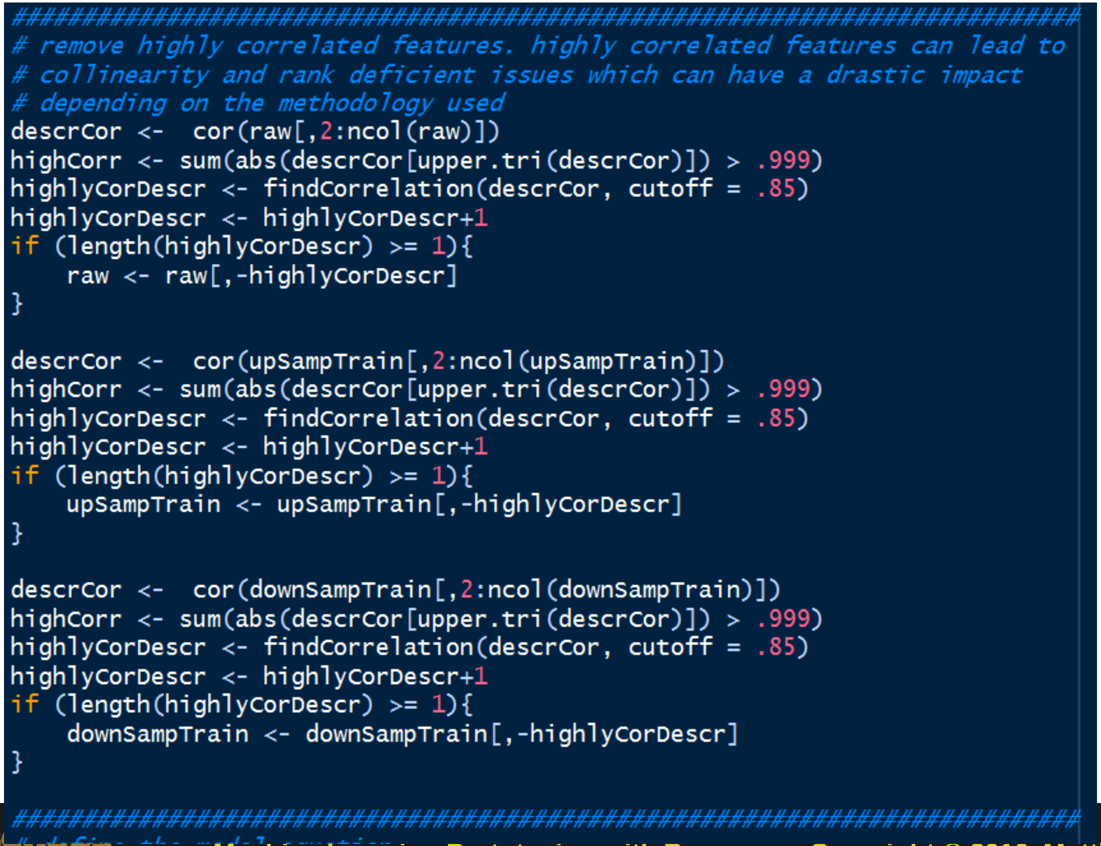
```
#############################################################################
n <- 100000
### Make sure if the number of records is greater than "n" sample it down more
### This should help with runtime
if (nrow(raw) > n){
    inRaw <- createDataPartition(y = raw$Y, #outcome variable
                                 p = round(n/nrow(raw),2), #% of training data
                                 list = FALSE)
    # reduced raw train
    raw <- raw[inRaw,]
}
if (nrow(downSampTrain) > n){
    inDown <- createDataPartition(y = downSampTrain$Y, #outcome variable
                                  p = round(n/nrow(downSampTrain),2), #% of training data
                                  list = FALSE)
    # reduced down-sampled training set
    downSampTrain <- downSampTrain[inDown,]
}
if (nrow(upSampTrain) > n){
    inUp <- createDataPartition(y = upSampTrain$Y, #outcome variable
                                p = round(n/nrow(upSampTrain),2), #% of training data
                                list = FALSE)
    # reduced raw train
    upSampTrain <- upSampTrain[inUp,]
}
# remove these temporary variables
suppressWarnings(rm(inTrain, n, inRaw, inDown, inUp))
```

Removing features that can cause learning issues



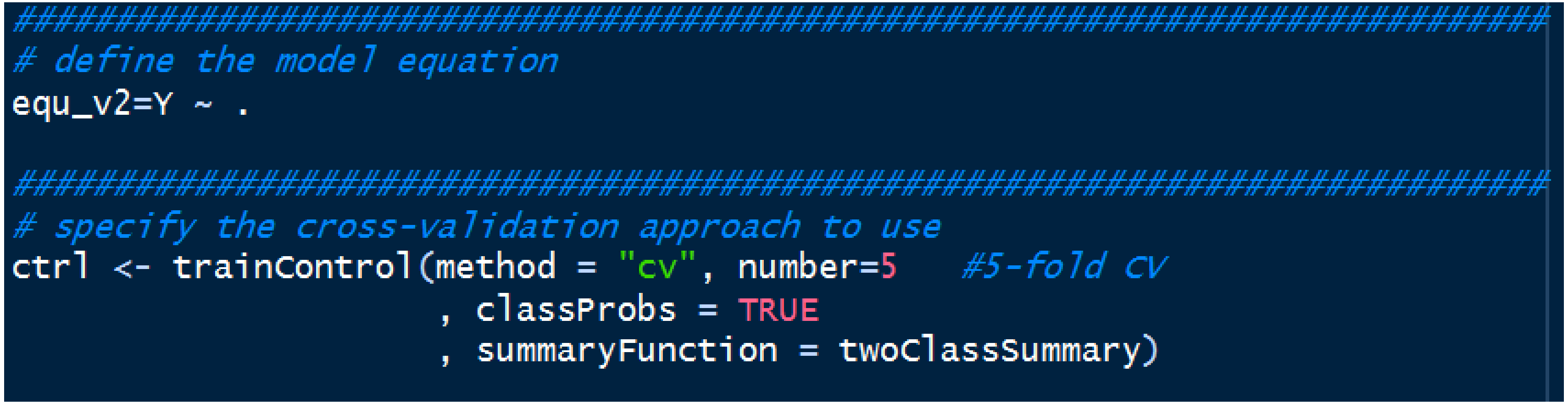
```
############################################################################
# remove features where the values they take on is limited
# here we make sure to keep the target variable and only those input
# features with enough variation
nzv <- nearZeroVar(raw, saveMetrics = TRUE)
raw <- raw[, c(TRUE,!nzv$zeroVar[2:ncol(raw)])]

nzv <- nearZeroVar(upSampTrain, saveMetrics = TRUE)
upSampTrain <- upSampTrain[, c(TRUE,!nzv$zeroVar[2:ncol(upSampTrain)])]

nzv <- nearZeroVar(downSampTrain, saveMetrics = TRUE)
downSampTrain <- downSampTrain[, c(TRUE,!nzv$zeroVar[2:ncol(downSampTrain)])]

############################################################################
```

Removing highly correlated features can also help algorithms learn



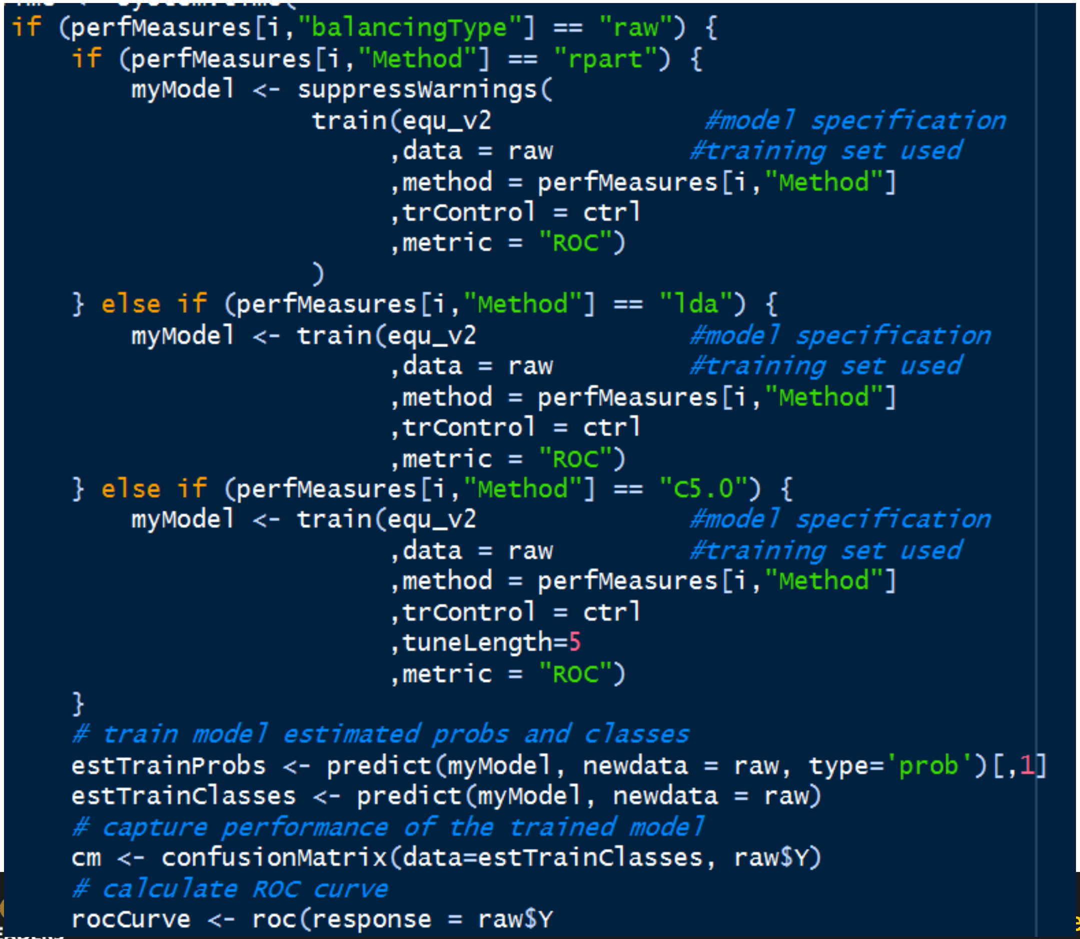
```
###########################################################################
# remove highly correlated features. highly correlated features can lead to
# collinearity and rank deficient issues which can have a drastic impact
# depending on the methodology used
descrCor <-  cor(raw[,2:ncol(raw)])
highCorr <- sum(abs(descrCor[upper.tri(descrCor)]) > .999)
highlyCorDescr <- findCorrelation(descrCor, cutoff = .85)
highlyCorDescr <- highlyCorDescr+1
if (length(highlyCorDescr) >= 1){
    raw <- raw[,-highlyCorDescr]
}

descrCor <-  cor(upSampTrain[,2:ncol(upSampTrain)])
highCorr <- sum(abs(descrCor[upper.tri(descrCor)]) > .999)
highlyCorDescr <- findCorrelation(descrCor, cutoff = .85)
highlyCorDescr <- highlyCorDescr+1
if (length(highlyCorDescr) >= 1){
    upSampTrain <- upSampTrain[,-highlyCorDescr]
}

descrCor <-  cor(downSampTrain[,2:ncol(downSampTrain)])
highCorr <- sum(abs(descrCor[upper.tri(descrCor)]) > .999)
highlyCorDescr <- findCorrelation(descrCor, cutoff = .85)
highlyCorDescr <- highlyCorDescr+1
if (length(highlyCorDescr) >= 1){
    downSampTrain <- downSampTrain[,-highlyCorDescr]
}

###########################################################################
```

trainControl() is where you specify your design



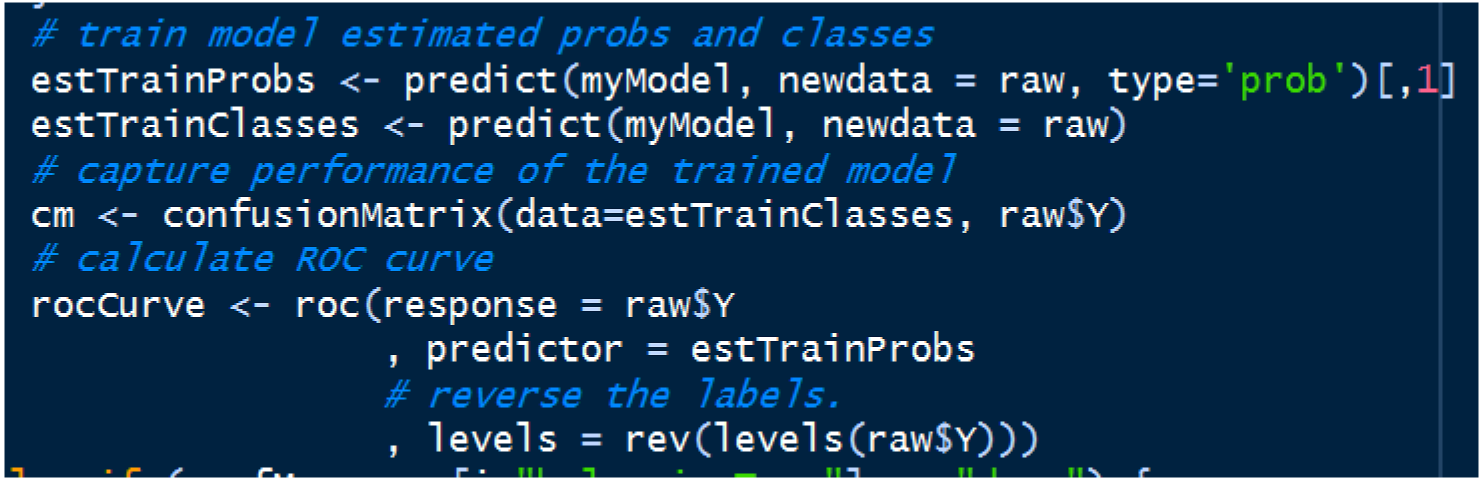
```r
#############################################################################
# define the model equation
equ_v2=Y ~ .

#############################################################################
# specify the cross-validation approach to use
ctrl <- trainControl(method = "cv", number=5   #5-fold cv
                     , classProbs = TRUE
                     , summaryFunction = twoClassSummary)
```

train () is where the magic happens



```
if (perfMeasures[i,"balancingType"] == "raw") {
    if (perfMeasures[i,"Method"] == "rpart") {
        myModel <- suppressWarnings(
                    train(equ_v2                        #model specification
                         ,data = raw                   #training set used
                         ,method = perfMeasures[i,"Method"]
                         ,trControl = ctrl
                         ,metric = "ROC")
                    )
    } else if (perfMeasures[i,"Method"] == "lda") {
        myModel <- train(equ_v2                        #model specification
                         ,data = raw                   #training set used
                         ,method = perfMeasures[i,"Method"]
                         ,trControl = ctrl
                         ,metric = "ROC")
    } else if (perfMeasures[i,"Method"] == "C5.0") {
        myModel <- train(equ_v2                        #model specification
                         ,data = raw                   #training set used
                         ,method = perfMeasures[i,"Method"]
                         ,trControl = ctrl
                         ,tuneLength=5
                         ,metric = "ROC")
    }
    # train model estimated probs and classes
    estTrainProbs <- predict(myModel, newdata = raw, type='prob')[,1]
    estTrainClasses <- predict(myModel, newdata = raw)
    # capture performance of the trained model
    cm <- confusionMatrix(data=estTrainClasses, raw$Y)
    # calculate ROC curve
    rocCurve <- roc(response = raw$Y
```

predict() allows you to predict on for the training set or test set

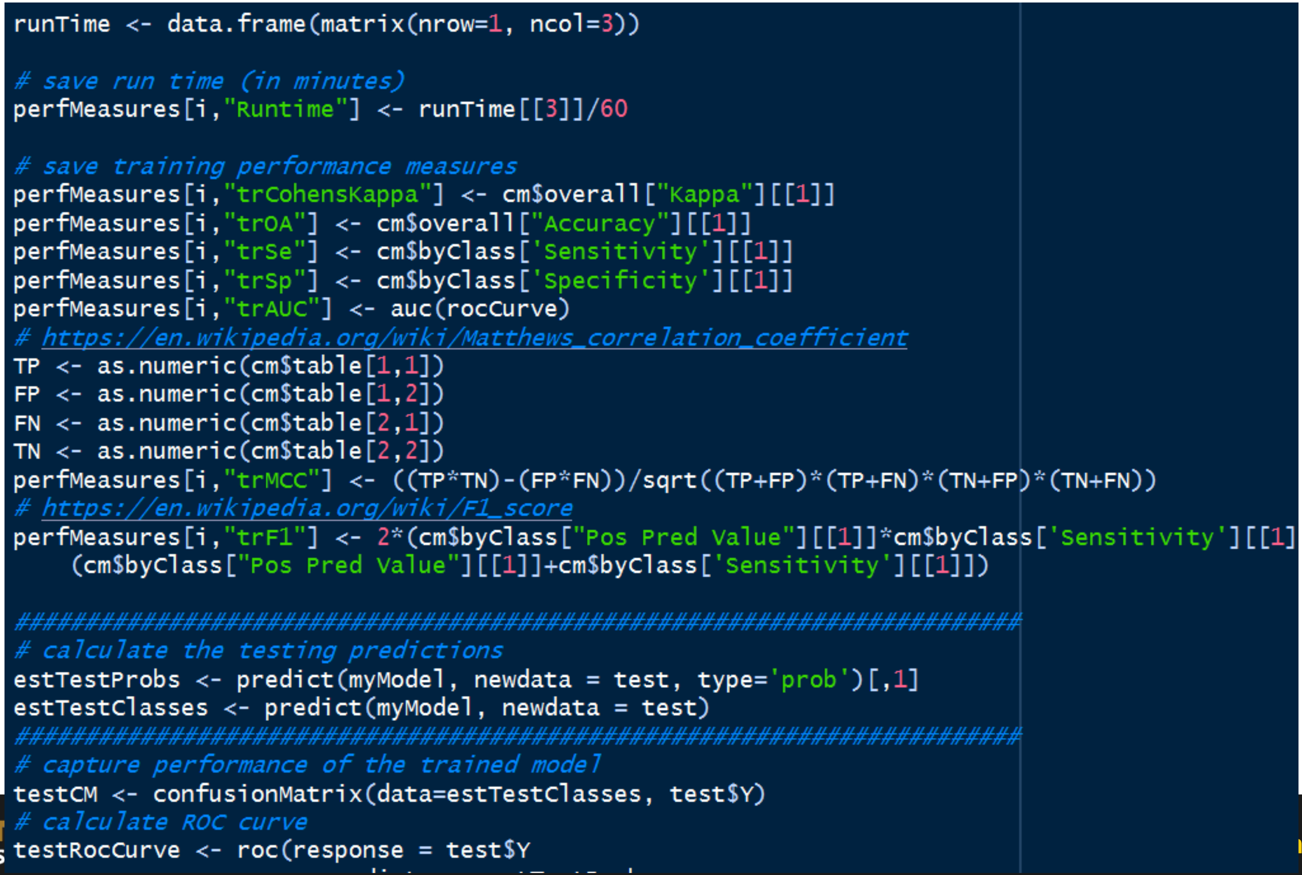


```
# train model estimated probs and classes
estTrainProbs <- predict(myModel, newdata = raw, type='prob')[,1]
estTrainClasses <- predict(myModel, newdata = raw)
# capture performance of the trained model
cm <- confusionMatrix(data=estTrainClasses, raw$Y)
# calculate ROC curve
rocCurve <- roc(response = raw$Y
                , predictor = estTrainProbs
                # reverse the labels.
                , levels = rev(levels(raw$Y)))
```

confusionMatrix() calculates the most common stats for classification problems

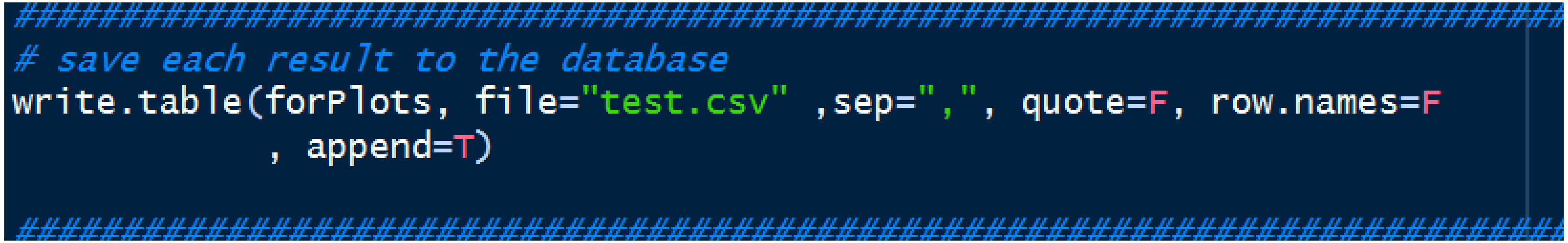
roc() provides you an ROC curve from where the AUC statistic can be obtained

After model learns, capture runtime and test set performance

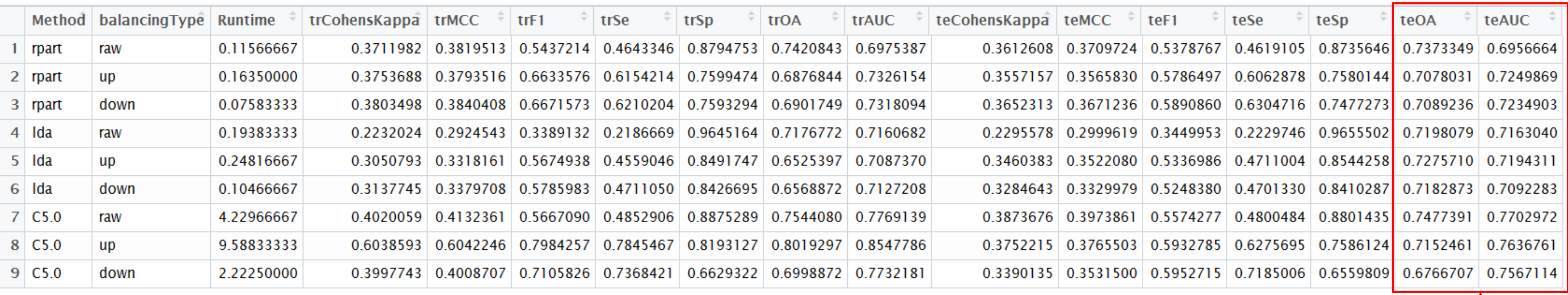


```
runTime <- data.frame(matrix(nrow=1, ncol=3))

# save run time (in minutes)
perfMeasures[i,"Runtime"] <- runTime[[3]]/60

# save training performance measures
perfMeasures[i,"trCohensKappa"] <- cm$overall["Kappa"][[1]]
perfMeasures[i,"trOA"] <- cm$overall["Accuracy"][[1]]
perfMeasures[i,"trSe"] <- cm$byClass['Sensitivity'][[1]]
perfMeasures[i,"trSp"] <- cm$byClass['Specificity'][[1]]
perfMeasures[i,"trAUC"] <- auc(rocCurve)
# https://en.wikipedia.org/wiki/Matthews_correlation_coefficient
TP <- as.numeric(cm$table[1,1])
FP <- as.numeric(cm$table[1,2])
FN <- as.numeric(cm$table[2,1])
TN <- as.numeric(cm$table[2,2])
perfMeasures[i,"trMCC"] <- ((TP*TN)-(FP*FN))/sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
# https://en.wikipedia.org/wiki/F1_score
perfMeasures[i,"trF1"] <- 2*(cm$byClass["Pos Pred Value"][[1]]*cm$byClass['Sensitivity'][[1]
    (cm$byClass["Pos Pred Value"][[1]]+cm$byClass['Sensitivity'][[1]])

########################################################################
# calculate the testing predictions
estTestProbs <- predict(myModel, newdata = test, type='prob')[,1]
estTestClasses <- predict(myModel, newdata = test)
########################################################################
# capture performance of the trained model
testCM <- confusionMatrix(data=estTestClasses, test$Y)
# calculate ROC curve
testRocCurve <- roc(response = test$Y
```

Save predicted probabilities on test set by writing to DB or file



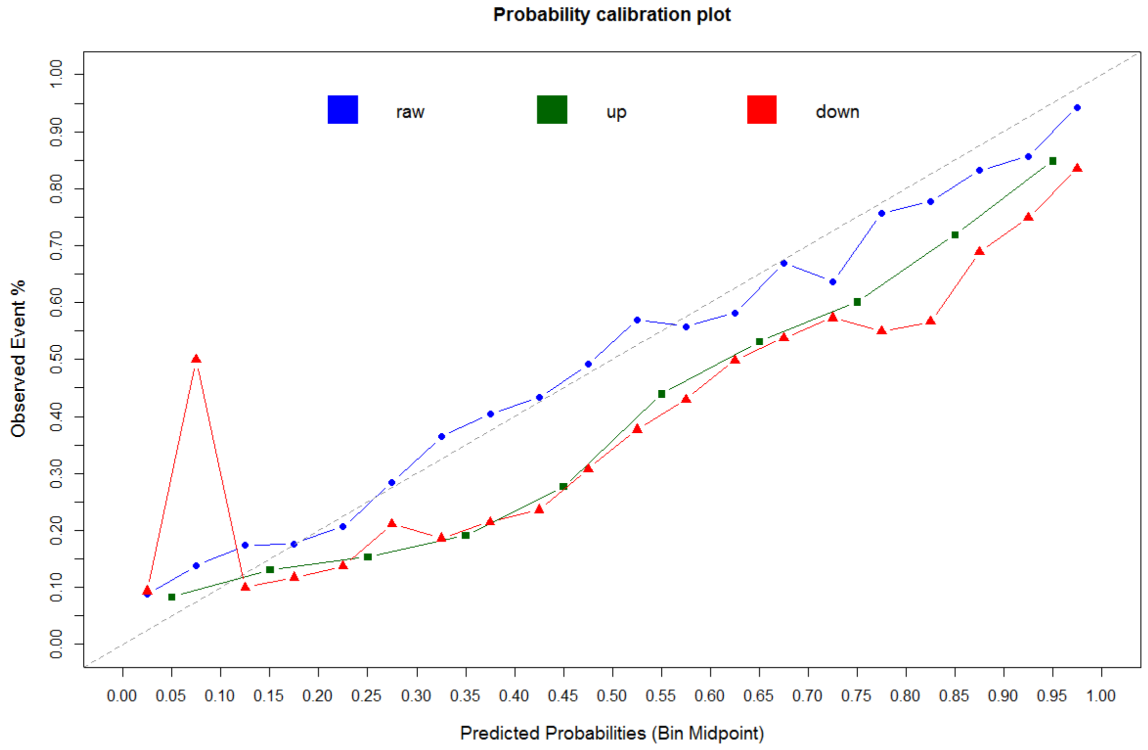
```
################################################################################
# save each result to the database
write.table(forPlots, file="test.csv" ,sep=",", quote=F, row.names=F
            , append=T)

################################################################################
```



	Method	balancingType	Runtime	trCohensKappa	trMCC	trF1	trSe	trSp	trOA	trAUC	teCohensKappa	teMCC	teF1	teSe	teSp	teOA	teAUC
1	rpart	raw	0.11566667	0.3711982	0.3819513	0.5437214	0.4643346	0.8794753	0.7420843	0.6975387	0.3612608	0.3709724	0.5378767	0.4619105	0.8735646	0.7373349	0.6956664
2	rpart	up	0.16350000	0.3753688	0.3793516	0.6633576	0.6154214	0.7599474	0.6876844	0.7326154	0.3557157	0.3565830	0.5786497	0.6062878	0.7580144	0.7078031	0.7249869
3	rpart	down	0.07583333	0.3803498	0.3840408	0.6671573	0.6210204	0.7593294	0.6901749	0.7318094	0.3652313	0.3671236	0.5890860	0.6304716	0.7477273	0.7089236	0.7234903
4	lda	raw	0.19383333	0.2232024	0.2924543	0.3389132	0.2186669	0.9645164	0.7176772	0.7160682	0.2295578	0.2999619	0.3449953	0.2229746	0.9655502	0.7198079	0.7163040
5	lda	up	0.24816667	0.3050793	0.3318161	0.5674938	0.4559046	0.8491747	0.6525397	0.7087370	0.3460383	0.3522080	0.5336986	0.4711004	0.8544258	0.7275710	0.7194311
6	lda	down	0.10466667	0.3137745	0.3379708	0.5785983	0.4711050	0.8426695	0.6568872	0.7127208	0.3284643	0.3329979	0.5248380	0.4701330	0.8410287	0.7182873	0.7092283
7	C5.0	raw	4.22966667	0.4020059	0.4132361	0.5667090	0.4852906	0.8875289	0.7544080	0.7769139	0.3873676	0.3973861	0.5574277	0.4800484	0.8801435	0.7477391	0.7702972
8	C5.0	up	9.58833333	0.6038593	0.6042246	0.7984257	0.7845467	0.8193127	0.8019297	0.8547786	0.3752215	0.3765503	0.5932785	0.6275695	0.7586124	0.7152461	0.7636761
9	C5.0	down	2.22250000	0.3997743	0.4008707	0.7105826	0.7368421	0.6629322	0.6998872	0.7732181	0.3390135	0.3531500	0.5952715	0.7185006	0.6559809	0.6766707	0.7567114

The performance among models and balance type might be very similar. However, the probabilities might be significantly different

Potentially better performance assessment



You should look at the probability calibration plots to see how each method performs with respect to each balancing approach

Here we see across all methods, not rebalancing performs best across all probability bins

You might break this out my method (i.e. LDA vs. C5.0)

Take Aways

1. The R caret package provides some outstanding functions to build nearly any kind of regression or classification-type model
2. Often models will lead to similar test set performance based on statistical performance measures (e.g. AUC)
3. Consider how those performance measures support the problem at hand. Should you use other ways to evaluate those models?
4. I recommend using
   - Probability calibration plots for binary classification-type problems
   - Cost-based confusion matrices to better gauge performance with regard to the business problem

Student projects this past year

Fall 2016

- Probability to sell calibration for unbalanced data sets – Morgan Wu (MS BAIM) *
- Improving machine scheduled maintenance policies using text analytics – Felipe Camacho (MBA) *
- Developing an HR analytics framework to predict turnover – Akhilesh Karaumanchi (MS BAIM, pursuing doctoral students in HR post-BAIM) *
- Identifying fraudulent bank activity using text analytics – Jeffery Mei (MBA), Lukia Chen (MS Accounting), Shiva Biradar (MS BAIM)
- Developing scalable optimization models using SolverStudio, an Excel front-end for optimization – Prerana Ghosh (MS BAIM)

Spring 2017

- Field research on predicting turnover and policy modifications – Akhilesh Karaumanchi (MS BAIM) *
- Retailer incentive policy for furniture manufacturer – Anupama Bharathula (MS BAIM) *
- Estimating the value of analytical benchmarking – Amar Iyengar (MS BAIM) *

*** Indicates collaborating with industry partner**