Part 2 Homework

## Warning: package 'knitr' was built under R version 3.4.1

Packages used

library(AppliedPredictiveModeling)  
library(Amelia)  
library(doParallel)  
library(missForest)  
library(cwhmisc)  
library(MASS)  
library(forecast)  
library(tidyverse)  
library(pls)  
library(caret)  
library(data.table)  
library(psych)  
library(mlbench)  
library(nnet)  
library(earth)  
library(kernlab)  
library(Metrics)  
library(party)

# Chapter 6

## Problem 6.3

A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is the understand the relationship between biological measurements of the raw materials (predictors), measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1 % will boost revenue by approximately one hundred thousand dollars per batch.

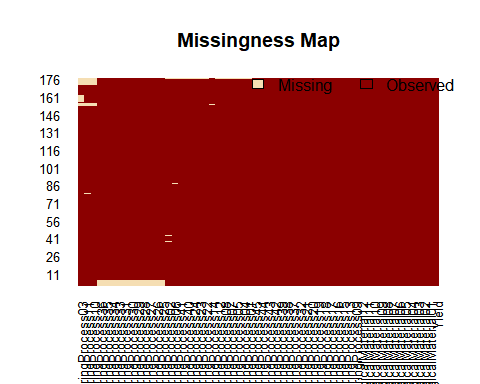
### a. Start $ and use the commands to load the data:

library(AppliedPredictiveModeling)  
data("ChemicalManufacturingProcess")

The matrix processPredictors contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. yield contains the percent yield for each run.

### b. A small percentage of cells in the predictor set contains missing values. Use an imputation function to fill in these missing values.

library(Amelia)  
missmap(ChemicalManufacturingProcess)



It appears that the missing values are quite minor and seem to be missing at random.

library(doParallel)  
library(missForest)  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100, parallelize = "forests")$ximp

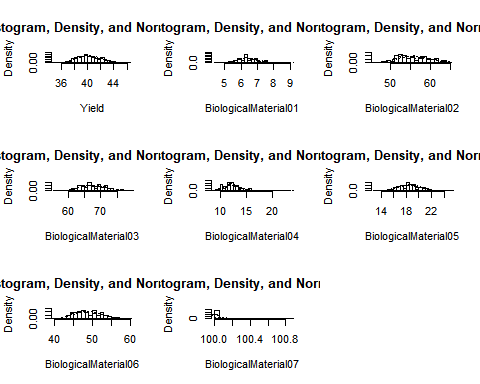
## missForest iteration 1 in progress...done!  
## missForest iteration 2 in progress...done!  
## missForest iteration 3 in progress...done!  
## missForest iteration 4 in progress...done!  
## missForest iteration 5 in progress...done!  
## missForest iteration 6 in progress...done!

data.set <- list(org.data.set = chem.imput)

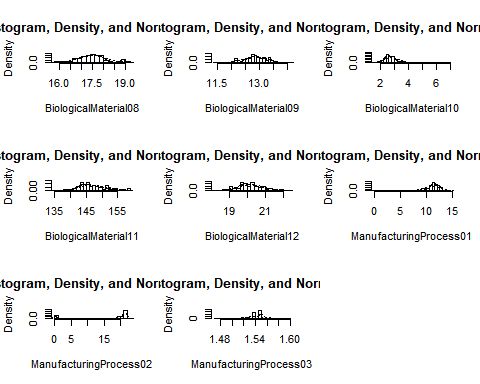
### c. Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter. What is the optimal value of the performance metric?

Lets explore the distribution of the variables

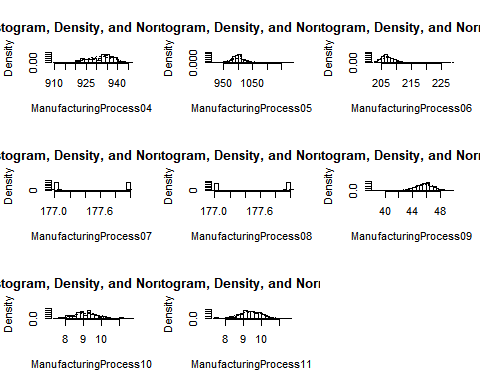
library(psych)  
  
multi.hist(data.set$org.data.set[1:8])



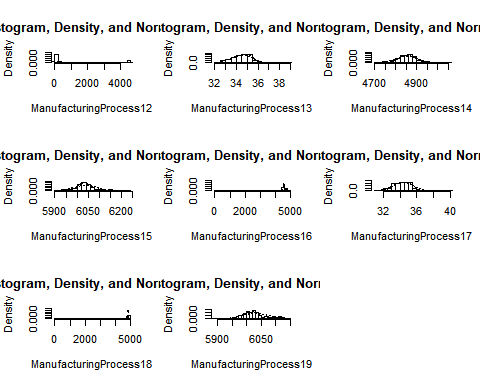
multi.hist(data.set$org.data.set[9:16])



multi.hist(data.set$org.data.set[17:24])



multi.hist(data.set$org.data.set[25:32])



The distributions of several of the variables are noticeably skewed, so we will obtain the box cox transformation lambda

variables.to.transform <- c("BiologicalMaterial04", "ManufacturingProcess01",  
 "ManufacturingProcess02","ManufacturingProcess03",  
 "ManufacturingProcess06", "ManufacturingProcess07",  
 "ManufacturingProcess08","ManufacturingProcess12",  
 "ManufacturingProcess16","ManufacturingProcess18")   
  
results <- NULL  
for (i in 1:length(variables.to.transform)){  
 results[[i]] <- list(variable = variables.to.transform[[i]],  
 lambda = BoxCox.lambda(data.set$org.data.set[,variables.to.transform[[i]]]))}  
results

## [[1]]  
## [[1]]$variable  
## [1] "BiologicalMaterial04"  
##   
## [[1]]$lambda  
## [1] -0.9999242  
##   
##   
## [[2]]  
## [[2]]$variable  
## [1] "ManufacturingProcess01"  
##   
## [[2]]$lambda  
## [1] 1.999959  
##   
##   
## [[3]]  
## [[3]]$variable  
## [1] "ManufacturingProcess02"  
##   
## [[3]]$lambda  
## [1] 1.999959  
##   
##   
## [[4]]  
## [[4]]$variable  
## [1] "ManufacturingProcess03"  
##   
## [[4]]$lambda  
## [1] -0.9999242  
##   
##   
## [[5]]  
## [[5]]$variable  
## [1] "ManufacturingProcess06"  
##   
## [[5]]$lambda  
## [1] -0.9999242  
##   
##   
## [[6]]  
## [[6]]$variable  
## [1] "ManufacturingProcess07"  
##   
## [[6]]$lambda  
## [1] -0.9999242  
##   
##   
## [[7]]  
## [[7]]$variable  
## [1] "ManufacturingProcess08"  
##   
## [[7]]$lambda  
## [1] 1.999924  
##   
##   
## [[8]]  
## [[8]]$variable  
## [1] "ManufacturingProcess12"  
##   
## [[8]]$lambda  
## [1] 4.102259e-05  
##   
##   
## [[9]]  
## [[9]]$variable  
## [1] "ManufacturingProcess16"  
##   
## [[9]]$lambda  
## [1] 1.999959  
##   
##   
## [[10]]  
## [[10]]$variable  
## [1] "ManufacturingProcess18"  
##   
## [[10]]$lambda  
## [1] 1.999959

We have our lambda transformations and we can apply them to our data set then set our test and train splices.

trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))

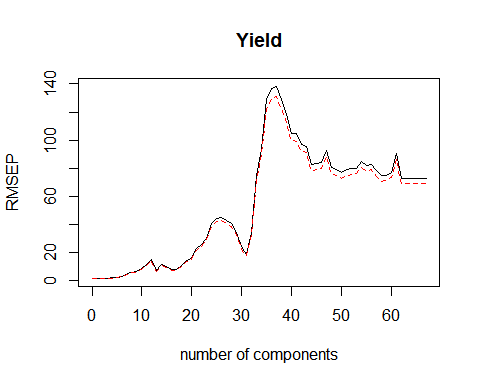
Splitting the test and training data set

set.seed(123)

data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))

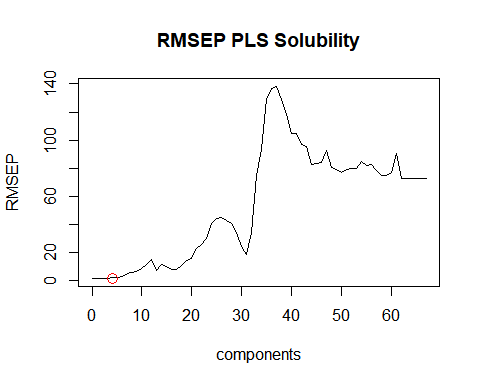
Now we can proceed with the model of choice which is the Partial Least Squares. Note that the evaluation below relied heavily on this rpubs post- <https://rpubs.com/omicsdata/pls>.

library(pls)  
chem.plsFit <- plsr(Yield ~ ., data = data.set$train, validation = "CV")  
validationplot(chem.plsFit, val.type="RMSEP")



We see a steep increase at 10 components followed by many mountains and valleys. Our intention is to select the most optimal number of components to minimize the RMSE.

pls.RMSEP <- RMSEP(chem.plsFit, estimate="CV")   
plot(pls.RMSEP, main="RMSEP PLS Solubility", xlab="components")  
min\_comp <- which.min(pls.RMSEP$val)  
points(min\_comp, min(pls.RMSEP$val), pch=1, col="red", cex=1.5)



Our optimal number of components is 4.

[1] RPubs - Partial Least Square Regression. N.p., n.d. Web. 14 Nov. 2017. - <https://rpubs.com/omicsdata/pls>

### d. Predict the response for the test set. What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

Our results from our training set

chem.plsPredict.train <- predict(chem.plsFit, data = data.set$train, ncomp = min\_comp)  
pls.eval <- data.frame(obs = data.set$train[,1], pred = chem.plsPredict.train[,1,1])  
defaultSummary(pls.eval)

## RMSE Rsquared MAE   
## 1.7109143 0.1354777 1.3530899

Results from the test set

chem.plsPredict <- predict(chem.plsFit, data = data.set$test, ncomp = min\_comp)  
pls.eval <- data.frame(obs = data.set$test[,1], pred = chem.plsPredict[,1,1])  
defaultSummary(pls.eval)

## RMSE Rsquared MAE   
## 1.920760986 0.004181154 1.592635928

While our RMSE is within range of both the training and test data set the Rsquared is significantly less for the test set suggesting a model that does not explain much of the variation. This may not be the most optimal model.

### e. Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

library(data.table)  
dt <- setDT(varImp(chem.plsFit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: BiologicalMaterial04.boxcoxtrans 9.0275564751268  
## 2: ManufacturingProcess41 1.2783915795656  
## 3: ManufacturingProcess40 1.2495325629426  
## 4: ManufacturingProcess29 0.9255757758128  
## 5: ManufacturingProcess45 0.7470772307046  
## 6: BiologicalMaterial12 0.3712700364646  
## 7: ManufacturingProcess43 0.2358243837861  
## 8: ManufacturingProcess21 0.2324369299319  
## 9: ManufacturingProcess11 0.2318275318242  
## 10: ManufacturingProcess17 0.2018184024741  
## 11: ManufacturingProcess30 0.1921440126146  
## 12: BiologicalMaterial08 0.1904666801390  
## 13: BiologicalMaterial01 0.1721553706209  
## 14: ManufacturingProcess10 0.1597670456727  
## 15: BiologicalMaterial04 0.1466271741442  
## 16: BiologicalMaterial07 0.1452188425985  
## 17: ManufacturingProcess02 0.1391134988042  
## 18: ManufacturingProcess39 0.0976417072856  
## 19: ManufacturingProcess37 0.0949233288120  
## 20: ManufacturingProcess01 0.0943979133454  
## 21: BiologicalMaterial06 0.0870274786329  
## 22: ManufacturingProcess33 0.0754325789994  
## 23: ManufacturingProcess38 0.0596514033320  
## 24: ManufacturingProcess31 0.0499205483308  
## 25: BiologicalMaterial02 0.0469961760327  
## 26: ManufacturingProcess13 0.0438348925576  
## 27: BiologicalMaterial05 0.0358341915880  
## 28: ManufacturingProcess42 0.0342554727683  
## 29: ManufacturingProcess32 0.0300427740037  
## 30: ManufacturingProcess28 0.0162017415364  
## 31: ManufacturingProcess09 0.0115538746313  
## 32: ManufacturingProcess02.boxcoxtrans 0.0099078810954  
## 33: ManufacturingProcess22 0.0092656099873  
## 34: ManufacturingProcess27 0.0090857048417  
## 35: BiologicalMaterial11 0.0070487242418  
## 36: ManufacturingProcess26 0.0067861545961  
## 37: ManufacturingProcess01.boxcoxtrans 0.0062067697607  
## 38: ManufacturingProcess18 0.0060491885372  
## 39: ManufacturingProcess25 0.0057714400202  
## 40: ManufacturingProcess14 0.0050344202237  
## 41: ManufacturingProcess19 0.0041324995282  
## 42: ManufacturingProcess04 0.0032774216273  
## 43: ManufacturingProcess06 0.0031966456038  
## 44: ManufacturingProcess20 0.0030512632510  
## 45: ManufacturingProcess24 0.0021820436061  
## 46: ManufacturingProcess16 0.0020501354365  
## 47: ManufacturingProcess05 0.0019832198005  
## 48: ManufacturingProcess08.boxcoxtrans 0.0000830265073  
## 49: ManufacturingProcess12 0.0000310896574  
## 50: ManufacturingProcess18.boxcoxtrans 0.0000014502081  
## 51: ManufacturingProcess16.boxcoxtrans 0.0000009127477  
## 52: ManufacturingProcess08 0.0000004680961  
## 53: ManufacturingProcess07.boxcoxtrans -0.0000029056796  
## 54: ManufacturingProcess12.boxcoxtrans -0.0000110032514  
## 55: ManufacturingProcess15 -0.0002103327042  
## 56: ManufacturingProcess35 -0.0073209396680  
## 57: BiologicalMaterial03 -0.0080233973745  
## 58: ManufacturingProcess23 -0.0166518935531  
## 59: BiologicalMaterial10 -0.0196413714591  
## 60: ManufacturingProcess44 -0.0200174754672  
## 61: ManufacturingProcess34 -0.0364037163238  
## 62: ManufacturingProcess07 -0.0915104339283  
## 63: ManufacturingProcess06.boxcoxtrans -0.2065524087591  
## 64: BiologicalMaterial09 -0.2437851152132  
## 65: ManufacturingProcess03 -182.5901328013508  
## 66: ManufacturingProcess36 -332.6232098390906  
## 67: ManufacturingProcess03.boxcoxtrans -432.7176722574175  
## rn Overall

We see that the Manufacturing Processes dominate the list for variable importance with a few biological materials having importance.

### f. Explore the relationships between each of the top predictors and the response. How would this information be helpful in improving yield in future runs of the manufacturing process?

The top manufacturing process is ManufacturingProcess40

data.set$org.data.set %>% dplyr::select(Yield, ManufacturingProcess40) %>% cor() %>% .[2]

## [1] -0.04576402

We see a negative correlation ManufacturingProcess40 and Yield, which suggests that ManufacturingProcess40 represents a process that can decrease the Yield. We cannot say that ManufacturingProcess40 itself decrease Yield, only that a negative relationship exists.

The top biological process is BiologicalMaterial12

data.set$org.data.set %>% dplyr::select(Yield, BiologicalMaterial12) %>% cor() %>% .[2]

## [1] 0.3674976

We actually see the opposite for the Biological indicator in that there is a positive correlation to Yield. We cannot say that BiologicalMaterial12 itself improves Yield, only that a positive relationship exists.

# Chapter 7

## Problem 7.2

Friedman (1991) introduced several benchmark data sets created by simulation. One of these simulations used the following nonlinear equation to create data:

where the ex values are random variables uniformly distributed between [0,1] (there are also 50 other non-informative variables also created in simulation). The package mlbench contains a function called mlbench.friedman1 that simulates these data:

set.seed(1234)  
trainingData <- mlbench.friedman1(200, sd = 1)  
trainingData$x <- data.frame(trainingData$x)  
#featurePlot(trainingData$x, trainingData$y)  
testData <- mlbench.friedman1(5000, sd = 1)  
testData$x <- data.frame(testData$x)

### Tune several models on these data.

Training Neural Net

nnetFit <- nnet(testData$x, testData$y,  
 size = 5,  
 decay = .01,  
 maxit = 500,  
 MaxNWts = 5 \* (ncol(testData$x) + 1) + 5 + 1)

## # weights: 61  
## initial value 1077271.618912   
## iter 10 value 1035722.717788  
## iter 20 value 1035626.108753  
## final value 1035625.899884   
## converged

results <- predict(nnetFit, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 14.36606

Training MARS

marsFit <- earth(testData$x, testData$y)  
results <- predict(marsFit, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 1.785523

Train K-Nearest Neighbors

This method at times will work and other times will not due to a known issue with the latest caret release. The error is “unable to find variable”optimismBoot“”, I am leaving the code in to show my attempt but note that this will not run appropriately.

#At one point I got the error optimismBoot not found, installing caret from github resolved the issue.   
knnTune <- train(testData$x, testData$y,   
 method = "knn",   
 preProc = c("center", "scale"),  
 trace = FALSE)  
results <- predict(knnTune, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

### Which models appear to give the best performance? Does MARS select informative predictors (those named X1-X5).

The model with the lowest performance metric that we are using is the RMSE value of 1.7855 for the MARS method.

varImp(marsFit)

## Overall  
## X4 100.00000  
## X1 78.68733  
## X2 63.36434  
## X5 44.56206  
## X3 31.99334  
## X6 0.00000  
## X7 0.00000  
## X8 0.00000  
## X9 0.00000  
## X10 0.00000

The MARS method did select variables with the most important of X1-X5 as shown in our previous table.

## Problem 7.5

Exercise 6.3 describes data for a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several nonlinear regression models.

The below is the minimum steps needed to recreate the data set from Chapter 6. Please see problem from Chapter 6 for complete breakdown for each step and reasons behind the choices.

library(AppliedPredictiveModeling)  
data("ChemicalManufacturingProcess")

library(missForest)  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100)$ximp

## missForest iteration 1 in progress...done!  
## missForest iteration 2 in progress...done!  
## missForest iteration 3 in progress...done!  
## missForest iteration 4 in progress...done!

data.set <- list(org.data.set = chem.imput)

trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))

data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))

### Model Selection

Training Neural Net

nnetFit <- nnet(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 size = 5,  
 decay = .01,  
 maxit = 500,  
 trace = FALSE)  
  
results <- predict(nnetFit,data.set$test %>% dplyr::select(-Yield)) %>% cbind(.,data.set$test$Yield) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 38.89715

Training MARS

marsFit <- earth(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield)  
results <- predict(marsFit, data.set$test %>% dplyr::select(-Yield)) %>% cbind(.,data.set$test$Yield) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)

## [1] 1.401761

Attempts to model a K-nearest neighbor was unsuccesful, the caret package appears to have a conflict with optimismboot and is unable to accomodate this data set. <https://stackoverflow.com/questions/46244763/caret-train-function-unable-to-find-variable-optimismboot>

### a. Which nonlinear regression model gives the optimal re-sampling and test set performance?

The nonlinear regression model with the best test set performance was again the MARS model with an RMSE of 1.401761.

### b. Which predictors are the most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

library(data.table)  
dt <- setDT(varImp(marsFit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

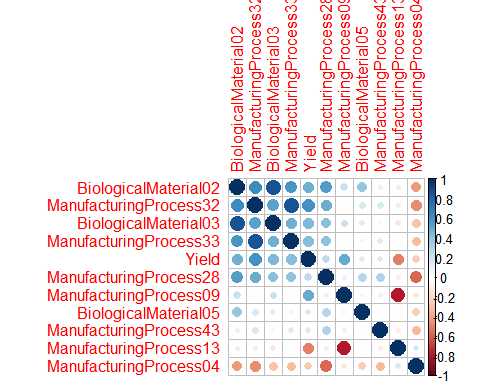
## rn Overall  
## 1: ManufacturingProcess32 100.00000  
## 2: ManufacturingProcess09 67.72174  
## 3: ManufacturingProcess13 42.33973  
## 4: ManufacturingProcess33 37.43641  
## 5: BiologicalMaterial02 37.25570  
## 6: BiologicalMaterial03 37.25570  
## 7: ManufacturingProcess28 31.32309  
## 8: ManufacturingProcess43 27.29428  
## 9: ManufacturingProcess04 19.81703  
## 10: BiologicalMaterial05 12.51240  
## 11: BiologicalMaterial01 0.00000  
## 12: BiologicalMaterial04 0.00000  
## 13: BiologicalMaterial06 0.00000  
## 14: BiologicalMaterial07 0.00000  
## 15: BiologicalMaterial08 0.00000  
## 16: BiologicalMaterial09 0.00000  
## 17: BiologicalMaterial10 0.00000  
## 18: BiologicalMaterial11 0.00000  
## 19: BiologicalMaterial12 0.00000  
## 20: ManufacturingProcess01 0.00000  
## 21: ManufacturingProcess02 0.00000  
## 22: ManufacturingProcess03 0.00000  
## 23: ManufacturingProcess05 0.00000  
## 24: ManufacturingProcess06 0.00000  
## 25: ManufacturingProcess07 0.00000  
## 26: ManufacturingProcess08 0.00000  
## 27: ManufacturingProcess10 0.00000  
## 28: ManufacturingProcess11 0.00000  
## 29: ManufacturingProcess12 0.00000  
## 30: ManufacturingProcess14 0.00000  
## 31: ManufacturingProcess15 0.00000  
## 32: ManufacturingProcess16 0.00000  
## 33: ManufacturingProcess17 0.00000  
## 34: ManufacturingProcess18 0.00000  
## 35: ManufacturingProcess19 0.00000  
## 36: ManufacturingProcess20 0.00000  
## 37: ManufacturingProcess21 0.00000  
## 38: ManufacturingProcess22 0.00000  
## 39: ManufacturingProcess23 0.00000  
## 40: ManufacturingProcess24 0.00000  
## 41: ManufacturingProcess25 0.00000  
## 42: ManufacturingProcess26 0.00000  
## 43: ManufacturingProcess27 0.00000  
## 44: ManufacturingProcess29 0.00000  
## 45: ManufacturingProcess30 0.00000  
## 46: ManufacturingProcess31 0.00000  
## 47: ManufacturingProcess34 0.00000  
## 48: ManufacturingProcess35 0.00000  
## 49: ManufacturingProcess36 0.00000  
## 50: ManufacturingProcess37 0.00000  
## 51: ManufacturingProcess38 0.00000  
## 52: ManufacturingProcess39 0.00000  
## 53: ManufacturingProcess40 0.00000  
## 54: ManufacturingProcess41 0.00000  
## 55: ManufacturingProcess42 0.00000  
## 56: ManufacturingProcess44 0.00000  
## 57: ManufacturingProcess45 0.00000  
## 58: BiologicalMaterial04.boxcoxtrans 0.00000  
## 59: ManufacturingProcess01.boxcoxtrans 0.00000  
## 60: ManufacturingProcess02.boxcoxtrans 0.00000  
## 61: ManufacturingProcess03.boxcoxtrans 0.00000  
## 62: ManufacturingProcess06.boxcoxtrans 0.00000  
## 63: ManufacturingProcess07.boxcoxtrans 0.00000  
## 64: ManufacturingProcess08.boxcoxtrans 0.00000  
## 65: ManufacturingProcess12.boxcoxtrans 0.00000  
## 66: ManufacturingProcess16.boxcoxtrans 0.00000  
## 67: ManufacturingProcess18.boxcoxtrans 0.00000  
## rn Overall

We see that the Manufacturing Processes dominate the list for variable importance with a few biological materials having importance. This is similar to our previous model however we do see a few more biological materials variables. Although we see none of the variables are previous model found important which suggests that the manufacturing processes contain a lot of information on the variance of yield.

### c. Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

subset.data.set <- data.set$trans.data.set %>%   
 dplyr::select(ManufacturingProcess32, ManufacturingProcess09,   
 ManufacturingProcess13, ManufacturingProcess33,   
 BiologicalMaterial02, BiologicalMaterial03,   
 ManufacturingProcess28, ManufacturingProcess43,   
 ManufacturingProcess04, BiologicalMaterial05, Yield)

library(corrplot)  
correlations <- cor(subset.data.set)  
corrplot(correlations, order = "FPC", diag = TRUE)



The correlation plot provides some insight to these important variables. Yield has a mostly positive correlation with many of hte processes but has the strongest negative correlation with only manufacturing processes from our top 10 variables. We also see some very high correlations with other variables in our top variable set which suggests possible issues of multicollinearity.

library(tidyverse)

# Chapter 8

## Problem 8.1

### Recreate the simulated data from Exercise 7.2:

library(mlbench)  
set.seed(200)  
simulated <- mlbench.friedman1(200, sd = 1)  
simulated <- cbind(simulated$x, simulated$y)  
simulated <- as.data.frame(simulated)  
colnames(simulated)[ncol(simulated)] <- "y"

### a. Fit a random forest model to all of the predictors, then estimate the variable importance

library(randomForest)  
library(caret)  
model1 <- randomForest(y ~., data = simulated, importance = TRUE,  
 ntree = 1000)  
rfImp <- varImp(model1, scale = FALSE)  
rfImp

## Overall  
## V1 8.66496174  
## V2 6.36504194  
## V3 0.76724347  
## V4 7.93742254  
## V5 2.10190958  
## V6 0.19154480  
## V7 0.04177869  
## V8 -0.08293296  
## V9 -0.06267951  
## V10 -0.04521494

Does the random forest model significantly use the uninformative predictors (V6-V10).

The random forest does not use the uniformative predictors. In fact, the variable importance is in line with the variable numbering.

### b. Now add an additional predictor that is highly correlated with one of the informative predictors. For Example:

simulated$duplicate1 <- simulated$V1 + rnorm(200) \* .1  
cor(simulated$duplicate1, simulated$V1)

## [1] 0.9386619

### Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1?

model1 <- randomForest(y ~., data = simulated, importance = TRUE,  
 ntree = 1000)  
rfImp <- varImp(model1, scale = FALSE)

library(data.table)  
dt <- setDT(varImp(model1, scale = FALSE), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: V4 6.59026253  
## 2: V2 6.04782043  
## 3: V1 5.34138782  
## 4: duplicate1 4.71267815  
## 5: V5 1.86532068  
## 6: V3 0.54776459  
## 7: V6 0.13680850  
## 8: V9 0.06233358  
## 9: V10 0.05506748  
## 10: V7 0.04745508  
## 11: V8 -0.01270436

We do see that the importance score has been reduced with the duplicate variable. The interaction between the two variables is reducing the signal we receive from using simple one of the variables.

simulated$duplicate2 <- simulated$V1 + rnorm(200) \* .15  
cor(simulated$duplicate2, simulated$V1)

## [1] 0.8849327

model2 <- randomForest(y ~., data = simulated, importance = TRUE,  
 ntree = 1000)  
rfImp <- varImp(model2, scale = FALSE)

library(data.table)  
dt <- setDT(varImp(model2, scale = FALSE), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: V4 7.292570850  
## 2: V2 6.562165756  
## 3: V1 4.562573722  
## 4: duplicate1 4.295437598  
## 5: V5 2.070957251  
## 6: duplicate2 1.384370802  
## 7: V3 0.517422886  
## 8: V6 0.132806443  
## 9: V7 0.072280073  
## 10: V9 -0.003797135  
## 11: V10 -0.049610851  
## 12: V8 -0.082135816

We don’t any improvements are reductions in the variable importance but it is reducing variable importance of other variables that may be meaningful for other reasons like literature based reasons.

### c. Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in Strobl et Al. (2007). Do these importance show the same pattern as the traditional random forest model?

library(party)  
fit.cforest <- cforest(y ~., data = simulated)

library(data.table)  
dt <- setDT(party::varimp(fit.cforest, conditional = TRUE) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-.)]

## rn .  
## 1: V4 5.281196888  
## 2: V2 4.373678395  
## 3: duplicate1 1.719885895  
## 4: V1 1.288320322  
## 5: V5 0.971063628  
## 6: duplicate2 0.276946982  
## 7: V3 0.021087453  
## 8: V6 0.019535747  
## 9: V7 0.011797289  
## 10: V10 0.002561214  
## 11: V9 -0.001990153  
## 12: V8 -0.005674798

dt <- setDT(party::varimp(fit.cforest, conditional = FALSE) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-.)]

## rn .  
## 1: V4 6.57619578  
## 2: V2 5.79417540  
## 3: duplicate1 4.99198355  
## 4: V1 3.74038496  
## 5: V5 1.62301336  
## 6: duplicate2 1.02429697  
## 7: V7 0.04711682  
## 8: V9 0.02985211  
## 9: V6 0.01570609  
## 10: V3 0.01140049  
## 11: V10 -0.01415559  
## 12: V8 -0.03169090

The top few variables remain at the same level of importance for both models. However, the variable importance is less using conditional inference trees via cpart from the party package.

### d. Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur?

library(Cubist)  
fit.cubist <- cubist(simulated %>% select(-y), simulated$y, committees = 4)  
fit.cubist

##   
## Call:  
## cubist.default(x = simulated %>% select(-y), y = simulated$y, committees  
## = 4)  
##   
## Number of samples: 200   
## Number of predictors: 12   
##   
## Number of committees: 4   
## Number of rules per committee: 1, 4, 2, 5

dt <- setDT(varImp(fit.cubist) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: V2 64.0  
## 2: V1 63.5  
## 3: V4 50.0  
## 4: V5 49.0  
## 5: V3 35.0  
## 6: duplicate1 25.5  
## 7: duplicate2 7.5  
## 8: V6 4.0  
## 9: V7 0.0  
## 10: V8 0.0  
## 11: V9 0.0  
## 12: V10 0.0

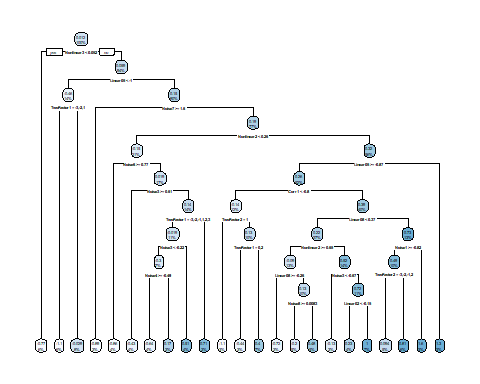
Using the cubist model we actually get very different results. In fact, the model appears to adjust for the highly correlated variables and gives us variable importance that more closely match our original data set before the added variables.

## Problem 8.2

### Use a simulation to show tree bias with different granularity.

library(mlbench)  
library(tidyverse)  
library(rpart)  
library(rpart.plot)  
library(caret)  
  
data.set <- twoClassSim(300, noiseVars = 8, corrVar = 6, corrValue = 0.8) %>%  
 mutate(TwoFactor1 = as.factor(round(TwoFactor1, 0)),  
 TwoFactor2 = as.factor(round(TwoFactor2, 0)))

r.fit <- rpart(Linear01 ~ ., data=data.set)  
rpart.plot(r.fit)



library(data.table)  
dt <- setDT(varImp(r.fit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]

## rn Overall  
## 1: TwoFactor2 1.62598157  
## 2: TwoFactor1 0.92692954  
## 3: Noise1 0.76886577  
## 4: Linear09 0.72470303  
## 5: Linear08 0.71648162  
## 6: Noise5 0.69175952  
## 7: Noise6 0.61321134  
## 8: Noise3 0.59730960  
## 9: Linear02 0.59195971  
## 10: Noise4 0.54439475  
## 11: Linear05 0.48916749  
## 12: Linear07 0.47884757  
## 13: Corr5 0.46520871  
## 14: Linear03 0.43772882  
## 15: Linear10 0.43364590  
## 16: Linear04 0.43274643  
## 17: Nonlinear2 0.38760350  
## 18: Nonlinear3 0.38327928  
## 19: Corr2 0.36396153  
## 20: Noise8 0.31716985  
## 21: Corr3 0.31250548  
## 22: Corr6 0.28603545  
## 23: Noise7 0.24357693  
## 24: Corr1 0.19694726  
## 25: Linear06 0.17564253  
## 26: Nonlinear1 0.15408339  
## 27: Noise2 0.08328166  
## 28: Corr4 0.00000000  
## 29: Class 0.00000000  
## rn Overall

We can see the factor variables are not as important as the linear variables. Trees will typically bias towards linear variables as they give more opportunties to split a tree as opposed to factor/categorical variables. However, it’s important to know if the splits make sense for the data domain or not.

## Problem 8.3

In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by gradient. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 8.24 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data. The left-had plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9:

### a. Why does the model on the right focus its importance on just the first few of predictors, whereas the model on the left spreads importance across more predictors?

A learning rate closer to 1 will make less corrections for each tree added to the model. A high bagging fraction closer to 1 indicates more of the data is used which can lead to overfitting. So as the learning rate and bagging fraction increase, the importance will be placed on fewer and fewer predictors.

### b. Which model do you think would be more predictive of other samples?

Subsequently, if the parameters increase the model performance will then decrease. Therefore, the model on the left will perform better. Although, I would be cautious about usefulness. If a model uses variables that can’t always be reliably captured than a model dependent on very few variables may be more powerful.

### c. How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24?

Interaction depth specifies the tree depth and node splits. As the tree depth increase, and more node splits occur the variable importance becomes spread across more predictors. In both models the variable importance would decrease for the top variables and increase for less important variables. If we have any highly correlated variables we may actually see a swap of importance between the two variables.

## Problem 8.7

Refer to Exercise 6.3 and 7.5 which describe a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several tree-based models:

library(missForest)  
library(AppliedPredictiveModeling)  
library(doParallel)  
data("ChemicalManufacturingProcess")  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100)$ximp

## missForest iteration 1 in progress...done!  
## missForest iteration 2 in progress...done!  
## missForest iteration 3 in progress...done!  
## missForest iteration 4 in progress...done!

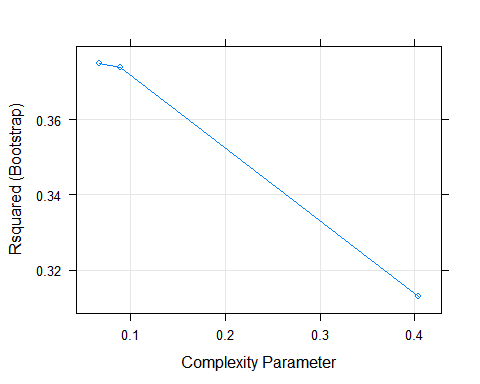
data.set <- list(org.data.set = chem.imput)

library(tidyverse)  
library(caret)  
library(forecast)  
library(MASS)  
  
trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))

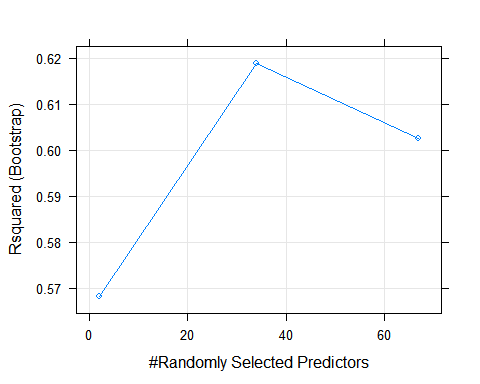
data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))

### a. Which tree-based regression model gives the optimal re-sampling and test set performance?

ctrl <- trainControl(method = "boot", number = 15)  
  
rpart.fit <- train(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 method = "rpart",  
 metric = "Rsquared",  
 trControl = ctrl)  
plot(rpart.fit)



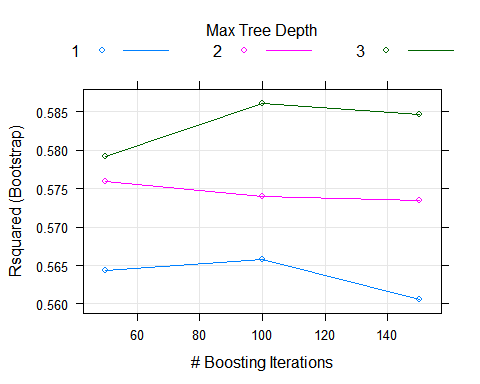
rf.fit <- train(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 method = "rf",  
 metric = "Rsquared",  
 trControl = ctrl,  
 importance = TRUE)  
plot(rf.fit)



gbm.fit <- train(data.set$train %>%   
 dplyr::select(-Yield), data.set$train$Yield,  
 method = "gbm",  
 metric = "Rsquared",  
 trControl = ctrl)

## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.9953 nan 0.1000 0.3368  
## 2 3.6156 nan 0.1000 0.3654  
## 3 3.2660 nan 0.1000 0.3245  
## 4 3.0058 nan 0.1000 0.2255  
## 5 2.8105 nan 0.1000 0.1083  
## 6 2.6106 nan 0.1000 0.1873  
## 7 2.4212 nan 0.1000 0.1752  
## 8 2.2881 nan 0.1000 0.1420  
## 9 2.1560 nan 0.1000 0.1216  
## 10 2.0312 nan 0.1000 0.0737  
## 20 1.3781 nan 0.1000 0.0350  
## 40 0.9108 nan 0.1000 -0.0024  
## 60 0.6883 nan 0.1000 -0.0018  
## 80 0.5427 nan 0.1000 0.0066  
## 100 0.4581 nan 0.1000 -0.0003  
## 120 0.3787 nan 0.1000 -0.0033  
## 140 0.3125 nan 0.1000 -0.0008  
## 150 0.2906 nan 0.1000 -0.0014  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.7652 nan 0.1000 0.5125  
## 2 3.3190 nan 0.1000 0.4032  
## 3 2.9354 nan 0.1000 0.3857  
## 4 2.6457 nan 0.1000 0.2844  
## 5 2.4099 nan 0.1000 0.2157  
## 6 2.1973 nan 0.1000 0.1525  
## 7 2.0238 nan 0.1000 0.1337  
## 8 1.8703 nan 0.1000 0.1032  
## 9 1.7619 nan 0.1000 0.0936  
## 10 1.6603 nan 0.1000 0.0695  
## 20 1.0101 nan 0.1000 0.0126  
## 40 0.5457 nan 0.1000 -0.0036  
## 60 0.3338 nan 0.1000 -0.0032  
## 80 0.2199 nan 0.1000 -0.0031  
## 100 0.1612 nan 0.1000 -0.0034  
## 120 0.1120 nan 0.1000 -0.0016  
## 140 0.0821 nan 0.1000 -0.0014  
## 150 0.0712 nan 0.1000 -0.0010  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.7196 nan 0.1000 0.5745  
## 2 3.2647 nan 0.1000 0.4460  
## 3 2.8620 nan 0.1000 0.3316  
## 4 2.6100 nan 0.1000 0.1780  
## 5 2.3983 nan 0.1000 0.2229  
## 6 2.1077 nan 0.1000 0.1487  
## 7 1.9459 nan 0.1000 0.1096  
## 8 1.8062 nan 0.1000 0.1462  
## 9 1.6552 nan 0.1000 0.1637  
## 10 1.5218 nan 0.1000 0.1078  
## 20 0.8097 nan 0.1000 0.0252  
## 40 0.3485 nan 0.1000 -0.0014  
## 60 0.1802 nan 0.1000 -0.0015  
## 80 0.1133 nan 0.1000 0.0006  
## 100 0.0721 nan 0.1000 -0.0016  
## 120 0.0485 nan 0.1000 -0.0005  
## 140 0.0341 nan 0.1000 -0.0007  
## 150 0.0288 nan 0.1000 -0.0003  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0320 nan 0.1000 0.2584  
## 2 2.7971 nan 0.1000 0.1871  
## 3 2.5582 nan 0.1000 0.1656  
## 4 2.3426 nan 0.1000 0.1416  
## 5 2.2227 nan 0.1000 0.0519  
## 6 2.0408 nan 0.1000 0.1451  
## 7 1.8892 nan 0.1000 0.1125  
## 8 1.7753 nan 0.1000 0.0586  
## 9 1.6579 nan 0.1000 0.1004  
## 10 1.5651 nan 0.1000 0.0761  
## 20 0.9843 nan 0.1000 0.0286  
## 40 0.6613 nan 0.1000 -0.0029  
## 60 0.5363 nan 0.1000 0.0027  
## 80 0.4396 nan 0.1000 -0.0043  
## 100 0.3716 nan 0.1000 -0.0068  
## 120 0.3189 nan 0.1000 -0.0006  
## 140 0.2840 nan 0.1000 -0.0057  
## 150 0.2667 nan 0.1000 -0.0043  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.8878 nan 0.1000 0.3257  
## 2 2.6199 nan 0.1000 0.2173  
## 3 2.3377 nan 0.1000 0.2220  
## 4 2.1266 nan 0.1000 0.1440  
## 5 1.9114 nan 0.1000 0.2125  
## 6 1.7575 nan 0.1000 0.1200  
## 7 1.5808 nan 0.1000 0.1115  
## 8 1.4377 nan 0.1000 0.1255  
## 9 1.3175 nan 0.1000 0.0821  
## 10 1.2262 nan 0.1000 0.0674  
## 20 0.6976 nan 0.1000 0.0197  
## 40 0.4132 nan 0.1000 -0.0074  
## 60 0.2912 nan 0.1000 -0.0020  
## 80 0.2103 nan 0.1000 -0.0019  
## 100 0.1553 nan 0.1000 -0.0050  
## 120 0.1181 nan 0.1000 -0.0010  
## 140 0.0940 nan 0.1000 -0.0011  
## 150 0.0838 nan 0.1000 -0.0006  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.9027 nan 0.1000 0.3542  
## 2 2.5708 nan 0.1000 0.3306  
## 3 2.2703 nan 0.1000 0.2969  
## 4 2.0541 nan 0.1000 0.1798  
## 5 1.8638 nan 0.1000 0.2075  
## 6 1.7101 nan 0.1000 0.0988  
## 7 1.5289 nan 0.1000 0.1333  
## 8 1.3969 nan 0.1000 0.1270  
## 9 1.2565 nan 0.1000 0.0945  
## 10 1.1529 nan 0.1000 0.0524  
## 20 0.6439 nan 0.1000 0.0077  
## 40 0.3264 nan 0.1000 -0.0025  
## 60 0.2010 nan 0.1000 -0.0014  
## 80 0.1219 nan 0.1000 -0.0002  
## 100 0.0854 nan 0.1000 -0.0018  
## 120 0.0623 nan 0.1000 0.0005  
## 140 0.0456 nan 0.1000 -0.0001  
## 150 0.0383 nan 0.1000 -0.0006  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 4.0189 nan 0.1000 0.3388  
## 2 3.6699 nan 0.1000 0.2554  
## 3 3.4544 nan 0.1000 0.1272  
## 4 3.1935 nan 0.1000 0.1714  
## 5 2.9598 nan 0.1000 0.1796  
## 6 2.8117 nan 0.1000 0.1162  
## 7 2.6131 nan 0.1000 0.1039  
## 8 2.4733 nan 0.1000 0.1235  
## 9 2.2947 nan 0.1000 0.1358  
## 10 2.1679 nan 0.1000 0.1148  
## 20 1.3963 nan 0.1000 0.0280  
## 40 0.8703 nan 0.1000 -0.0112  
## 60 0.6401 nan 0.1000 -0.0019  
## 80 0.5122 nan 0.1000 -0.0089  
## 100 0.4178 nan 0.1000 -0.0030  
## 120 0.3498 nan 0.1000 -0.0028  
## 140 0.2965 nan 0.1000 -0.0069  
## 150 0.2684 nan 0.1000 0.0018  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.8642 nan 0.1000 0.4288  
## 2 3.4464 nan 0.1000 0.3754  
## 3 3.1238 nan 0.1000 0.2614  
## 4 2.8492 nan 0.1000 0.2689  
## 5 2.5449 nan 0.1000 0.2401  
## 6 2.3687 nan 0.1000 0.2056  
## 7 2.1843 nan 0.1000 0.1163  
## 8 2.0183 nan 0.1000 0.1256  
## 9 1.8793 nan 0.1000 0.0932  
## 10 1.7751 nan 0.1000 0.0587  
## 20 1.0065 nan 0.1000 0.0335  
## 40 0.5242 nan 0.1000 0.0061  
## 60 0.3067 nan 0.1000 -0.0018  
## 80 0.2130 nan 0.1000 -0.0087  
## 100 0.1445 nan 0.1000 -0.0034  
## 120 0.1034 nan 0.1000 -0.0015  
## 140 0.0753 nan 0.1000 -0.0016  
## 150 0.0662 nan 0.1000 -0.0014  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.8703 nan 0.1000 0.4481  
## 2 3.3155 nan 0.1000 0.4246  
## 3 3.0086 nan 0.1000 0.2830  
## 4 2.6729 nan 0.1000 0.3156  
## 5 2.3874 nan 0.1000 0.2419  
## 6 2.1529 nan 0.1000 0.2202  
## 7 2.0069 nan 0.1000 0.1405  
## 8 1.8584 nan 0.1000 0.1251  
## 9 1.7155 nan 0.1000 0.1351  
## 10 1.5828 nan 0.1000 0.0989  
## 20 0.8593 nan 0.1000 0.0152  
## 40 0.3705 nan 0.1000 0.0013  
## 60 0.1924 nan 0.1000 -0.0010  
## 80 0.1145 nan 0.1000 -0.0006  
## 100 0.0714 nan 0.1000 -0.0006  
## 120 0.0505 nan 0.1000 -0.0006  
## 140 0.0368 nan 0.1000 -0.0010  
## 150 0.0299 nan 0.1000 -0.0002  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.3828 nan 0.1000 0.3376  
## 2 3.1072 nan 0.1000 0.2859  
## 3 2.8472 nan 0.1000 0.2475  
## 4 2.6775 nan 0.1000 0.1904  
## 5 2.5103 nan 0.1000 0.1548  
## 6 2.4077 nan 0.1000 0.0628  
## 7 2.2574 nan 0.1000 0.1353  
## 8 2.1555 nan 0.1000 0.0915  
## 9 2.0446 nan 0.1000 0.1060  
## 10 1.9496 nan 0.1000 0.0717  
## 20 1.3368 nan 0.1000 0.0355  
## 40 0.8942 nan 0.1000 0.0161  
## 60 0.6913 nan 0.1000 -0.0104  
## 80 0.5463 nan 0.1000 -0.0059  
## 100 0.4573 nan 0.1000 0.0010  
## 120 0.3898 nan 0.1000 -0.0013  
## 140 0.3358 nan 0.1000 -0.0045  
## 150 0.3199 nan 0.1000 -0.0009  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.2616 nan 0.1000 0.3557  
## 2 2.9271 nan 0.1000 0.3161  
## 3 2.6096 nan 0.1000 0.2288  
## 4 2.3818 nan 0.1000 0.1999  
## 5 2.1730 nan 0.1000 0.1854  
## 6 2.0471 nan 0.1000 0.1229  
## 7 1.9003 nan 0.1000 0.1002  
## 8 1.7369 nan 0.1000 0.0959  
## 9 1.6522 nan 0.1000 0.0562  
## 10 1.5335 nan 0.1000 0.0903  
## 20 0.9498 nan 0.1000 0.0183  
## 40 0.5488 nan 0.1000 -0.0009  
## 60 0.3661 nan 0.1000 -0.0024  
## 80 0.2533 nan 0.1000 -0.0041  
## 100 0.1866 nan 0.1000 -0.0004  
## 120 0.1386 nan 0.1000 -0.0009  
## 140 0.1105 nan 0.1000 -0.0010  
## 150 0.0962 nan 0.1000 -0.0007  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.3039 nan 0.1000 0.4292  
## 2 2.9932 nan 0.1000 0.2375  
## 3 2.6470 nan 0.1000 0.3065  
## 4 2.3974 nan 0.1000 0.1925  
## 5 2.1931 nan 0.1000 0.0889  
## 6 1.9844 nan 0.1000 0.1565  
## 7 1.8005 nan 0.1000 0.1305  
## 8 1.6725 nan 0.1000 0.1277  
## 9 1.5590 nan 0.1000 0.0750  
## 10 1.4457 nan 0.1000 0.0594  
## 20 0.8105 nan 0.1000 0.0233  
## 40 0.3943 nan 0.1000 -0.0034  
## 60 0.2404 nan 0.1000 -0.0062  
## 80 0.1625 nan 0.1000 -0.0022  
## 100 0.1171 nan 0.1000 -0.0011  
## 120 0.0825 nan 0.1000 -0.0028  
## 140 0.0611 nan 0.1000 -0.0019  
## 150 0.0521 nan 0.1000 0.0000  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.6418 nan 0.1000 0.2716  
## 2 2.4357 nan 0.1000 0.1873  
## 3 2.2423 nan 0.1000 0.1978  
## 4 2.0599 nan 0.1000 0.1598  
## 5 1.9162 nan 0.1000 0.1119  
## 6 1.8004 nan 0.1000 0.0954  
## 7 1.6827 nan 0.1000 0.0679  
## 8 1.5762 nan 0.1000 0.0675  
## 9 1.5260 nan 0.1000 0.0197  
## 10 1.4492 nan 0.1000 0.0368  
## 20 0.9533 nan 0.1000 0.0275  
## 40 0.5706 nan 0.1000 -0.0008  
## 60 0.4226 nan 0.1000 -0.0016  
## 80 0.3280 nan 0.1000 -0.0018  
## 100 0.2646 nan 0.1000 -0.0008  
## 120 0.2166 nan 0.1000 -0.0023  
## 140 0.1832 nan 0.1000 -0.0010  
## 150 0.1678 nan 0.1000 -0.0026  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.5844 nan 0.1000 0.3729  
## 2 2.3631 nan 0.1000 0.1328  
## 3 2.1000 nan 0.1000 0.2762  
## 4 1.8449 nan 0.1000 0.2124  
## 5 1.6525 nan 0.1000 0.1450  
## 6 1.5643 nan 0.1000 0.0724  
## 7 1.4491 nan 0.1000 0.1136  
## 8 1.3311 nan 0.1000 0.0667  
## 9 1.2133 nan 0.1000 0.0808  
## 10 1.1272 nan 0.1000 0.0705  
## 20 0.6554 nan 0.1000 0.0125  
## 40 0.3292 nan 0.1000 -0.0036  
## 60 0.1901 nan 0.1000 -0.0020  
## 80 0.1300 nan 0.1000 -0.0006  
## 100 0.0904 nan 0.1000 -0.0002  
## 120 0.0644 nan 0.1000 -0.0003  
## 140 0.0491 nan 0.1000 -0.0003  
## 150 0.0436 nan 0.1000 -0.0011  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.5654 nan 0.1000 0.2523  
## 2 2.2346 nan 0.1000 0.3227  
## 3 1.9629 nan 0.1000 0.2489  
## 4 1.7547 nan 0.1000 0.1835  
## 5 1.5441 nan 0.1000 0.1487  
## 6 1.4355 nan 0.1000 0.0993  
## 7 1.2929 nan 0.1000 0.1184  
## 8 1.1894 nan 0.1000 0.0927  
## 9 1.0726 nan 0.1000 0.0889  
## 10 0.9681 nan 0.1000 0.0717  
## 20 0.4891 nan 0.1000 0.0043  
## 40 0.2116 nan 0.1000 0.0028  
## 60 0.1163 nan 0.1000 -0.0009  
## 80 0.0781 nan 0.1000 -0.0023  
## 100 0.0554 nan 0.1000 -0.0016  
## 120 0.0385 nan 0.1000 -0.0002  
## 140 0.0282 nan 0.1000 -0.0007  
## 150 0.0241 nan 0.1000 -0.0007  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0242 nan 0.1000 0.2135  
## 2 2.8432 nan 0.1000 0.0977  
## 3 2.6243 nan 0.1000 0.1549  
## 4 2.4168 nan 0.1000 0.1772  
## 5 2.2501 nan 0.1000 0.0630  
## 6 2.1042 nan 0.1000 0.1362  
## 7 2.0048 nan 0.1000 0.0926  
## 8 1.8629 nan 0.1000 0.1046  
## 9 1.7580 nan 0.1000 0.0713  
## 10 1.6912 nan 0.1000 0.0432  
## 20 1.1176 nan 0.1000 0.0279  
## 40 0.7752 nan 0.1000 0.0020  
## 60 0.6290 nan 0.1000 -0.0008  
## 80 0.5283 nan 0.1000 -0.0020  
## 100 0.4482 nan 0.1000 -0.0059  
## 120 0.3899 nan 0.1000 -0.0021  
## 140 0.3416 nan 0.1000 -0.0034  
## 150 0.3224 nan 0.1000 -0.0095  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.9806 nan 0.1000 0.1760  
## 2 2.7093 nan 0.1000 0.2582  
## 3 2.4696 nan 0.1000 0.1619  
## 4 2.2810 nan 0.1000 0.2170  
## 5 2.0751 nan 0.1000 0.1926  
## 6 1.8912 nan 0.1000 0.1678  
## 7 1.7544 nan 0.1000 0.0921  
## 8 1.6406 nan 0.1000 0.0923  
## 9 1.5691 nan 0.1000 0.0766  
## 10 1.4545 nan 0.1000 0.0846  
## 20 0.9045 nan 0.1000 0.0143  
## 40 0.5306 nan 0.1000 -0.0064  
## 60 0.3697 nan 0.1000 -0.0074  
## 80 0.2623 nan 0.1000 -0.0059  
## 100 0.1998 nan 0.1000 -0.0056  
## 120 0.1480 nan 0.1000 -0.0015  
## 140 0.1172 nan 0.1000 -0.0007  
## 150 0.1052 nan 0.1000 -0.0009  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.9766 nan 0.1000 0.1933  
## 2 2.7017 nan 0.1000 0.2184  
## 3 2.4037 nan 0.1000 0.2390  
## 4 2.1119 nan 0.1000 0.2313  
## 5 1.9496 nan 0.1000 0.0482  
## 6 1.8128 nan 0.1000 0.1125  
## 7 1.6570 nan 0.1000 0.0881  
## 8 1.5296 nan 0.1000 0.1199  
## 9 1.4229 nan 0.1000 0.0538  
## 10 1.2983 nan 0.1000 0.0821  
## 20 0.6912 nan 0.1000 0.0222  
## 40 0.3856 nan 0.1000 -0.0129  
## 60 0.2732 nan 0.1000 -0.0041  
## 80 0.2008 nan 0.1000 0.0019  
## 100 0.1475 nan 0.1000 -0.0056  
## 120 0.1164 nan 0.1000 -0.0024  
## 140 0.0878 nan 0.1000 -0.0015  
## 150 0.0777 nan 0.1000 -0.0011  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0377 nan 0.1000 0.3057  
## 2 2.7503 nan 0.1000 0.2660  
## 3 2.6078 nan 0.1000 0.1602  
## 4 2.4508 nan 0.1000 0.1103  
## 5 2.2975 nan 0.1000 0.1606  
## 6 2.1538 nan 0.1000 0.1143  
## 7 2.0133 nan 0.1000 0.1164  
## 8 1.8989 nan 0.1000 0.0995  
## 9 1.8016 nan 0.1000 0.0583  
## 10 1.7221 nan 0.1000 0.0511  
## 20 1.2589 nan 0.1000 0.0102  
## 40 0.8399 nan 0.1000 0.0095  
## 60 0.6067 nan 0.1000 -0.0043  
## 80 0.4761 nan 0.1000 -0.0011  
## 100 0.3901 nan 0.1000 -0.0044  
## 120 0.3256 nan 0.1000 -0.0024  
## 140 0.2757 nan 0.1000 -0.0007  
## 150 0.2568 nan 0.1000 -0.0013  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0026 nan 0.1000 0.2880  
## 2 2.7139 nan 0.1000 0.2520  
## 3 2.4815 nan 0.1000 0.2001  
## 4 2.2637 nan 0.1000 0.1798  
## 5 2.1179 nan 0.1000 0.1003  
## 6 1.9756 nan 0.1000 0.1227  
## 7 1.7936 nan 0.1000 0.1157  
## 8 1.6854 nan 0.1000 0.0586  
## 9 1.5581 nan 0.1000 0.1022  
## 10 1.4508 nan 0.1000 0.0914  
## 20 0.8951 nan 0.1000 0.0103  
## 40 0.4847 nan 0.1000 -0.0026  
## 60 0.3041 nan 0.1000 -0.0064  
## 80 0.2095 nan 0.1000 -0.0045  
## 100 0.1454 nan 0.1000 -0.0005  
## 120 0.1081 nan 0.1000 0.0009  
## 140 0.0842 nan 0.1000 -0.0021  
## 150 0.0746 nan 0.1000 -0.0022  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.9002 nan 0.1000 0.3363  
## 2 2.5817 nan 0.1000 0.2931  
## 3 2.3312 nan 0.1000 0.2256  
## 4 2.1319 nan 0.1000 0.1630  
## 5 1.9557 nan 0.1000 0.1454  
## 6 1.7987 nan 0.1000 0.1229  
## 7 1.6400 nan 0.1000 0.1179  
## 8 1.5177 nan 0.1000 0.0780  
## 9 1.3799 nan 0.1000 0.0667  
## 10 1.2875 nan 0.1000 0.0550  
## 20 0.6846 nan 0.1000 0.0157  
## 40 0.3389 nan 0.1000 -0.0050  
## 60 0.1974 nan 0.1000 -0.0034  
## 80 0.1284 nan 0.1000 -0.0053  
## 100 0.0884 nan 0.1000 -0.0022  
## 120 0.0627 nan 0.1000 -0.0003  
## 140 0.0482 nan 0.1000 -0.0010  
## 150 0.0429 nan 0.1000 -0.0004  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.5617 nan 0.1000 0.2413  
## 2 2.3688 nan 0.1000 0.1972  
## 3 2.1975 nan 0.1000 0.1646  
## 4 2.0402 nan 0.1000 0.1377  
## 5 1.9330 nan 0.1000 0.0869  
## 6 1.8526 nan 0.1000 0.0686  
## 7 1.7636 nan 0.1000 0.0741  
## 8 1.6860 nan 0.1000 0.0434  
## 9 1.5848 nan 0.1000 0.0900  
## 10 1.5095 nan 0.1000 0.0422  
## 20 1.0501 nan 0.1000 0.0235  
## 40 0.7125 nan 0.1000 0.0067  
## 60 0.5713 nan 0.1000 -0.0096  
## 80 0.4649 nan 0.1000 -0.0012  
## 100 0.3851 nan 0.1000 -0.0054  
## 120 0.3364 nan 0.1000 -0.0056  
## 140 0.2924 nan 0.1000 -0.0014  
## 150 0.2673 nan 0.1000 -0.0014  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.4824 nan 0.1000 0.2691  
## 2 2.2102 nan 0.1000 0.2305  
## 3 1.9978 nan 0.1000 0.2013  
## 4 1.8170 nan 0.1000 0.0808  
## 5 1.6908 nan 0.1000 0.1457  
## 6 1.5534 nan 0.1000 0.0766  
## 7 1.4224 nan 0.1000 0.0849  
## 8 1.3574 nan 0.1000 0.0236  
## 9 1.2672 nan 0.1000 0.0743  
## 10 1.1945 nan 0.1000 0.0383  
## 20 0.7784 nan 0.1000 -0.0025  
## 40 0.4346 nan 0.1000 -0.0052  
## 60 0.2921 nan 0.1000 -0.0016  
## 80 0.2156 nan 0.1000 -0.0027  
## 100 0.1559 nan 0.1000 -0.0029  
## 120 0.1170 nan 0.1000 -0.0002  
## 140 0.0923 nan 0.1000 -0.0019  
## 150 0.0808 nan 0.1000 -0.0010  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.4409 nan 0.1000 0.2714  
## 2 2.1586 nan 0.1000 0.2266  
## 3 1.9740 nan 0.1000 0.1718  
## 4 1.7974 nan 0.1000 0.1835  
## 5 1.6535 nan 0.1000 0.0779  
## 6 1.5171 nan 0.1000 0.1416  
## 7 1.4294 nan 0.1000 0.0694  
## 8 1.3548 nan 0.1000 0.0537  
## 9 1.2685 nan 0.1000 0.0647  
## 10 1.1843 nan 0.1000 0.0525  
## 20 0.6537 nan 0.1000 0.0181  
## 40 0.3136 nan 0.1000 -0.0018  
## 60 0.1923 nan 0.1000 0.0022  
## 80 0.1307 nan 0.1000 -0.0012  
## 100 0.0942 nan 0.1000 -0.0021  
## 120 0.0675 nan 0.1000 -0.0012  
## 140 0.0533 nan 0.1000 -0.0018  
## 150 0.0465 nan 0.1000 -0.0005  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.5161 nan 0.1000 0.2605  
## 2 3.2633 nan 0.1000 0.2249  
## 3 3.0528 nan 0.1000 0.1627  
## 4 2.8918 nan 0.1000 0.1476  
## 5 2.7264 nan 0.1000 0.1600  
## 6 2.5403 nan 0.1000 0.1264  
## 7 2.3922 nan 0.1000 0.1030  
## 8 2.2579 nan 0.1000 0.0879  
## 9 2.1604 nan 0.1000 0.0876  
## 10 2.0680 nan 0.1000 0.0725  
## 20 1.5055 nan 0.1000 -0.0147  
## 40 0.9861 nan 0.1000 -0.0083  
## 60 0.7351 nan 0.1000 -0.0016  
## 80 0.5965 nan 0.1000 -0.0041  
## 100 0.4844 nan 0.1000 -0.0035  
## 120 0.4126 nan 0.1000 0.0045  
## 140 0.3552 nan 0.1000 -0.0058  
## 150 0.3365 nan 0.1000 -0.0024  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.4684 nan 0.1000 0.2066  
## 2 3.0998 nan 0.1000 0.3104  
## 3 2.8656 nan 0.1000 0.2155  
## 4 2.6344 nan 0.1000 0.1876  
## 5 2.4032 nan 0.1000 0.1202  
## 6 2.2220 nan 0.1000 0.1997  
## 7 2.0635 nan 0.1000 0.0867  
## 8 1.9297 nan 0.1000 0.1086  
## 9 1.7953 nan 0.1000 0.0971  
## 10 1.7053 nan 0.1000 0.0797  
## 20 1.1197 nan 0.1000 -0.0076  
## 40 0.6032 nan 0.1000 0.0053  
## 60 0.3818 nan 0.1000 0.0025  
## 80 0.2631 nan 0.1000 -0.0055  
## 100 0.1953 nan 0.1000 -0.0025  
## 120 0.1515 nan 0.1000 -0.0033  
## 140 0.1183 nan 0.1000 -0.0042  
## 150 0.1041 nan 0.1000 -0.0034  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.3497 nan 0.1000 0.3696  
## 2 3.0185 nan 0.1000 0.2242  
## 3 2.7160 nan 0.1000 0.2515  
## 4 2.4480 nan 0.1000 0.2339  
## 5 2.2658 nan 0.1000 0.1615  
## 6 2.0608 nan 0.1000 0.1527  
## 7 1.9181 nan 0.1000 0.0691  
## 8 1.7771 nan 0.1000 0.0821  
## 9 1.6890 nan 0.1000 0.0713  
## 10 1.5742 nan 0.1000 0.0651  
## 20 0.9083 nan 0.1000 -0.0125  
## 40 0.4202 nan 0.1000 -0.0011  
## 60 0.2539 nan 0.1000 -0.0006  
## 80 0.1642 nan 0.1000 -0.0027  
## 100 0.1157 nan 0.1000 -0.0023  
## 120 0.0841 nan 0.1000 -0.0004  
## 140 0.0622 nan 0.1000 -0.0006  
## 150 0.0529 nan 0.1000 0.0002  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.5276 nan 0.1000 0.2697  
## 2 3.2941 nan 0.1000 0.2758  
## 3 3.0156 nan 0.1000 0.2355  
## 4 2.8207 nan 0.1000 0.1522  
## 5 2.6585 nan 0.1000 0.1727  
## 6 2.4623 nan 0.1000 0.1412  
## 7 2.3475 nan 0.1000 0.0656  
## 8 2.2150 nan 0.1000 0.1011  
## 9 2.1182 nan 0.1000 0.0481  
## 10 2.0313 nan 0.1000 0.0810  
## 20 1.3979 nan 0.1000 0.0260  
## 40 0.9077 nan 0.1000 0.0049  
## 60 0.6581 nan 0.1000 0.0029  
## 80 0.5001 nan 0.1000 -0.0022  
## 100 0.3884 nan 0.1000 -0.0024  
## 120 0.3175 nan 0.1000 -0.0018  
## 140 0.2634 nan 0.1000 -0.0034  
## 150 0.2437 nan 0.1000 -0.0031  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.4793 nan 0.1000 0.2614  
## 2 3.1721 nan 0.1000 0.2793  
## 3 2.8103 nan 0.1000 0.2774  
## 4 2.5222 nan 0.1000 0.2757  
## 5 2.3017 nan 0.1000 0.1378  
## 6 2.1227 nan 0.1000 0.1530  
## 7 1.9231 nan 0.1000 0.1525  
## 8 1.7945 nan 0.1000 0.1539  
## 9 1.6586 nan 0.1000 0.0916  
## 10 1.5318 nan 0.1000 0.0596  
## 20 0.9188 nan 0.1000 0.0005  
## 40 0.4655 nan 0.1000 0.0053  
## 60 0.2749 nan 0.1000 -0.0002  
## 80 0.1836 nan 0.1000 -0.0014  
## 100 0.1349 nan 0.1000 -0.0012  
## 120 0.1000 nan 0.1000 -0.0012  
## 140 0.0722 nan 0.1000 -0.0005  
## 150 0.0629 nan 0.1000 -0.0004  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.3592 nan 0.1000 0.3605  
## 2 3.0284 nan 0.1000 0.3393  
## 3 2.7147 nan 0.1000 0.2699  
## 4 2.4307 nan 0.1000 0.2743  
## 5 2.2296 nan 0.1000 0.1930  
## 6 2.0347 nan 0.1000 0.1206  
## 7 1.8466 nan 0.1000 0.1452  
## 8 1.7200 nan 0.1000 0.1104  
## 9 1.5980 nan 0.1000 0.1311  
## 10 1.5021 nan 0.1000 0.0584  
## 20 0.7798 nan 0.1000 0.0455  
## 40 0.3393 nan 0.1000 0.0079  
## 60 0.1853 nan 0.1000 0.0001  
## 80 0.1189 nan 0.1000 -0.0029  
## 100 0.0808 nan 0.1000 -0.0011  
## 120 0.0567 nan 0.1000 0.0003  
## 140 0.0424 nan 0.1000 -0.0000  
## 150 0.0377 nan 0.1000 -0.0005  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.8693 nan 0.1000 0.1883  
## 2 2.7192 nan 0.1000 0.0513  
## 3 2.5703 nan 0.1000 0.1388  
## 4 2.4525 nan 0.1000 0.0869  
## 5 2.3282 nan 0.1000 0.1117  
## 6 2.1895 nan 0.1000 0.1356  
## 7 2.0778 nan 0.1000 0.0701  
## 8 1.9959 nan 0.1000 0.0380  
## 9 1.9480 nan 0.1000 -0.0003  
## 10 1.8629 nan 0.1000 0.0096  
## 20 1.3117 nan 0.1000 0.0363  
## 40 0.8866 nan 0.1000 0.0167  
## 60 0.6243 nan 0.1000 0.0034  
## 80 0.4833 nan 0.1000 -0.0010  
## 100 0.3935 nan 0.1000 -0.0021  
## 120 0.3249 nan 0.1000 -0.0011  
## 140 0.2654 nan 0.1000 -0.0014  
## 150 0.2444 nan 0.1000 -0.0004  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.7855 nan 0.1000 0.2597  
## 2 2.5661 nan 0.1000 0.1618  
## 3 2.3870 nan 0.1000 0.1701  
## 4 2.1973 nan 0.1000 0.0734  
## 5 2.0186 nan 0.1000 0.1612  
## 6 1.8923 nan 0.1000 0.1261  
## 7 1.7757 nan 0.1000 0.1072  
## 8 1.6618 nan 0.1000 0.0707  
## 9 1.5394 nan 0.1000 0.0721  
## 10 1.4417 nan 0.1000 0.0289  
## 20 0.8339 nan 0.1000 0.0190  
## 40 0.4187 nan 0.1000 -0.0008  
## 60 0.2646 nan 0.1000 0.0008  
## 80 0.1814 nan 0.1000 -0.0010  
## 100 0.1318 nan 0.1000 -0.0011  
## 120 0.0932 nan 0.1000 -0.0003  
## 140 0.0677 nan 0.1000 0.0002  
## 150 0.0569 nan 0.1000 0.0000  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.8481 nan 0.1000 0.1629  
## 2 2.5845 nan 0.1000 0.1704  
## 3 2.3367 nan 0.1000 0.0644  
## 4 2.0726 nan 0.1000 0.2152  
## 5 1.8770 nan 0.1000 0.1675  
## 6 1.7116 nan 0.1000 0.1739  
## 7 1.5693 nan 0.1000 0.0729  
## 8 1.4866 nan 0.1000 0.0623  
## 9 1.3829 nan 0.1000 0.0852  
## 10 1.2830 nan 0.1000 0.0695  
## 20 0.7237 nan 0.1000 0.0369  
## 40 0.3249 nan 0.1000 -0.0008  
## 60 0.1691 nan 0.1000 -0.0001  
## 80 0.1063 nan 0.1000 -0.0008  
## 100 0.0656 nan 0.1000 0.0001  
## 120 0.0475 nan 0.1000 -0.0006  
## 140 0.0326 nan 0.1000 0.0004  
## 150 0.0272 nan 0.1000 -0.0002  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.4009 nan 0.1000 0.1620  
## 2 3.1797 nan 0.1000 0.1663  
## 3 2.9721 nan 0.1000 0.1953  
## 4 2.7769 nan 0.1000 0.1491  
## 5 2.6063 nan 0.1000 0.1779  
## 6 2.4451 nan 0.1000 0.1202  
## 7 2.3397 nan 0.1000 0.1006  
## 8 2.2257 nan 0.1000 0.0798  
## 9 2.1519 nan 0.1000 -0.0094  
## 10 2.0855 nan 0.1000 0.0640  
## 20 1.4190 nan 0.1000 0.0297  
## 40 0.9075 nan 0.1000 -0.0011  
## 60 0.6677 nan 0.1000 -0.0089  
## 80 0.5240 nan 0.1000 -0.0048  
## 100 0.4242 nan 0.1000 -0.0072  
## 120 0.3496 nan 0.1000 -0.0062  
## 140 0.2950 nan 0.1000 -0.0018  
## 150 0.2799 nan 0.1000 -0.0016  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.3685 nan 0.1000 0.2818  
## 2 3.0712 nan 0.1000 0.2270  
## 3 2.7939 nan 0.1000 0.2363  
## 4 2.5995 nan 0.1000 0.1456  
## 5 2.3811 nan 0.1000 0.1627  
## 6 2.2125 nan 0.1000 0.1168  
## 7 2.0230 nan 0.1000 0.1212  
## 8 1.9041 nan 0.1000 0.1051  
## 9 1.7769 nan 0.1000 0.1266  
## 10 1.6525 nan 0.1000 0.0983  
## 20 1.0094 nan 0.1000 0.0082  
## 40 0.5729 nan 0.1000 -0.0166  
## 60 0.3495 nan 0.1000 -0.0009  
## 80 0.2504 nan 0.1000 -0.0022  
## 100 0.1905 nan 0.1000 -0.0033  
## 120 0.1468 nan 0.1000 0.0008  
## 140 0.1153 nan 0.1000 -0.0014  
## 150 0.1034 nan 0.1000 -0.0018  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.3626 nan 0.1000 0.2999  
## 2 3.0009 nan 0.1000 0.3030  
## 3 2.7203 nan 0.1000 0.2801  
## 4 2.4656 nan 0.1000 0.2589  
## 5 2.2255 nan 0.1000 0.2268  
## 6 2.0493 nan 0.1000 0.0916  
## 7 1.9089 nan 0.1000 0.0929  
## 8 1.7831 nan 0.1000 0.0896  
## 9 1.6285 nan 0.1000 0.0889  
## 10 1.5086 nan 0.1000 0.0844  
## 20 0.8659 nan 0.1000 0.0139  
## 40 0.4209 nan 0.1000 0.0077  
## 60 0.2485 nan 0.1000 0.0005  
## 80 0.1527 nan 0.1000 -0.0002  
## 100 0.1033 nan 0.1000 -0.0025  
## 120 0.0754 nan 0.1000 0.0008  
## 140 0.0550 nan 0.1000 -0.0013  
## 150 0.0485 nan 0.1000 -0.0015  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.5172 nan 0.1000 0.2696  
## 2 2.3783 nan 0.1000 0.1264  
## 3 2.1892 nan 0.1000 0.1314  
## 4 2.0604 nan 0.1000 0.1060  
## 5 1.9460 nan 0.1000 0.1117  
## 6 1.8638 nan 0.1000 0.0323  
## 7 1.7625 nan 0.1000 0.0839  
## 8 1.6962 nan 0.1000 0.0316  
## 9 1.6280 nan 0.1000 0.0480  
## 10 1.5399 nan 0.1000 0.0340  
## 20 1.1565 nan 0.1000 0.0125  
## 40 0.8044 nan 0.1000 0.0110  
## 60 0.6112 nan 0.1000 -0.0128  
## 80 0.4838 nan 0.1000 -0.0022  
## 100 0.3930 nan 0.1000 -0.0034  
## 120 0.3292 nan 0.1000 -0.0018  
## 140 0.2779 nan 0.1000 -0.0017  
## 150 0.2554 nan 0.1000 -0.0028  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.5114 nan 0.1000 0.2453  
## 2 2.2343 nan 0.1000 0.2157  
## 3 2.0285 nan 0.1000 0.1966  
## 4 1.9057 nan 0.1000 0.1263  
## 5 1.8075 nan 0.1000 0.0626  
## 6 1.6999 nan 0.1000 0.0728  
## 7 1.5970 nan 0.1000 0.0893  
## 8 1.4938 nan 0.1000 0.0822  
## 9 1.3858 nan 0.1000 0.0847  
## 10 1.2905 nan 0.1000 0.0547  
## 20 0.7935 nan 0.1000 0.0090  
## 40 0.4685 nan 0.1000 -0.0101  
## 60 0.2895 nan 0.1000 -0.0062  
## 80 0.1926 nan 0.1000 -0.0035  
## 100 0.1319 nan 0.1000 -0.0047  
## 120 0.0947 nan 0.1000 -0.0018  
## 140 0.0696 nan 0.1000 -0.0002  
## 150 0.0610 nan 0.1000 -0.0018  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 2.5077 nan 0.1000 0.2256  
## 2 2.2694 nan 0.1000 0.1997  
## 3 2.0709 nan 0.1000 0.1892  
## 4 1.8884 nan 0.1000 0.1589  
## 5 1.7600 nan 0.1000 0.0985  
## 6 1.6460 nan 0.1000 0.0899  
## 7 1.5263 nan 0.1000 0.0871  
## 8 1.4083 nan 0.1000 0.1000  
## 9 1.3321 nan 0.1000 0.0470  
## 10 1.2505 nan 0.1000 0.0384  
## 20 0.6869 nan 0.1000 0.0007  
## 40 0.3258 nan 0.1000 -0.0103  
## 60 0.1883 nan 0.1000 -0.0036  
## 80 0.1069 nan 0.1000 -0.0004  
## 100 0.0644 nan 0.1000 -0.0006  
## 120 0.0399 nan 0.1000 -0.0006  
## 140 0.0274 nan 0.1000 -0.0005  
## 150 0.0228 nan 0.1000 -0.0004  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.5717 nan 0.1000 0.3640  
## 2 3.2205 nan 0.1000 0.3471  
## 3 2.9046 nan 0.1000 0.2293  
## 4 2.6835 nan 0.1000 0.2030  
## 5 2.4973 nan 0.1000 0.2040  
## 6 2.3086 nan 0.1000 0.1867  
## 7 2.1831 nan 0.1000 0.0903  
## 8 2.0431 nan 0.1000 0.0720  
## 9 1.9431 nan 0.1000 0.0697  
## 10 1.8374 nan 0.1000 0.0634  
## 20 1.2728 nan 0.1000 0.0225  
## 40 0.8892 nan 0.1000 0.0048  
## 60 0.7171 nan 0.1000 -0.0023  
## 80 0.5956 nan 0.1000 -0.0051  
## 100 0.5177 nan 0.1000 -0.0072  
## 120 0.4467 nan 0.1000 -0.0080  
## 140 0.3922 nan 0.1000 -0.0034  
## 150 0.3741 nan 0.1000 -0.0009  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.4692 nan 0.1000 0.4238  
## 2 3.0652 nan 0.1000 0.3537  
## 3 2.7342 nan 0.1000 0.3110  
## 4 2.4539 nan 0.1000 0.2720  
## 5 2.2087 nan 0.1000 0.2638  
## 6 1.9856 nan 0.1000 0.1576  
## 7 1.8144 nan 0.1000 0.1352  
## 8 1.6923 nan 0.1000 0.1031  
## 9 1.5805 nan 0.1000 0.0133  
## 10 1.4820 nan 0.1000 0.0592  
## 20 0.9020 nan 0.1000 0.0251  
## 40 0.5710 nan 0.1000 -0.0064  
## 60 0.3818 nan 0.1000 0.0013  
## 80 0.2844 nan 0.1000 0.0001  
## 100 0.2010 nan 0.1000 0.0004  
## 120 0.1576 nan 0.1000 -0.0020  
## 140 0.1144 nan 0.1000 -0.0015  
## 150 0.1045 nan 0.1000 -0.0007  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.4620 nan 0.1000 0.4304  
## 2 3.0468 nan 0.1000 0.4185  
## 3 2.7469 nan 0.1000 0.1694  
## 4 2.4526 nan 0.1000 0.2742  
## 5 2.1944 nan 0.1000 0.1786  
## 6 1.9694 nan 0.1000 0.1992  
## 7 1.7796 nan 0.1000 0.1171  
## 8 1.6396 nan 0.1000 0.1337  
## 9 1.5225 nan 0.1000 0.0920  
## 10 1.3957 nan 0.1000 0.0639  
## 20 0.8012 nan 0.1000 0.0066  
## 40 0.4186 nan 0.1000 -0.0098  
## 60 0.2750 nan 0.1000 -0.0053  
## 80 0.2012 nan 0.1000 -0.0014  
## 100 0.1403 nan 0.1000 -0.0022  
## 120 0.0985 nan 0.1000 -0.0012  
## 140 0.0711 nan 0.1000 -0.0020  
## 150 0.0602 nan 0.1000 -0.0008  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.1495 nan 0.1000 0.1544  
## 2 2.8870 nan 0.1000 0.1870  
## 3 2.6211 nan 0.1000 0.1604  
## 4 2.4366 nan 0.1000 0.1108  
## 5 2.3080 nan 0.1000 0.0954  
## 6 2.1843 nan 0.1000 0.1292  
## 7 2.0485 nan 0.1000 0.1102  
## 8 1.9580 nan 0.1000 0.0712  
## 9 1.8633 nan 0.1000 0.0820  
## 10 1.7852 nan 0.1000 0.0704  
## 20 1.2383 nan 0.1000 0.0158  
## 40 0.8548 nan 0.1000 -0.0053  
## 60 0.6906 nan 0.1000 -0.0019  
## 80 0.5623 nan 0.1000 -0.0111  
## 100 0.4614 nan 0.1000 0.0002  
## 120 0.3808 nan 0.1000 -0.0007  
## 140 0.3205 nan 0.1000 0.0015  
## 150 0.2927 nan 0.1000 -0.0028  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0491 nan 0.1000 0.3049  
## 2 2.7564 nan 0.1000 0.3136  
## 3 2.5119 nan 0.1000 0.2531  
## 4 2.3128 nan 0.1000 0.1773  
## 5 2.1009 nan 0.1000 0.1509  
## 6 1.9437 nan 0.1000 0.1736  
## 7 1.7906 nan 0.1000 0.1480  
## 8 1.6514 nan 0.1000 0.1168  
## 9 1.5418 nan 0.1000 0.0693  
## 10 1.4454 nan 0.1000 0.0745  
## 20 0.8686 nan 0.1000 0.0280  
## 40 0.4757 nan 0.1000 0.0004  
## 60 0.3165 nan 0.1000 -0.0005  
## 80 0.2158 nan 0.1000 -0.0015  
## 100 0.1547 nan 0.1000 -0.0020  
## 120 0.1203 nan 0.1000 -0.0056  
## 140 0.0894 nan 0.1000 -0.0002  
## 150 0.0797 nan 0.1000 -0.0014  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0039 nan 0.1000 0.3066  
## 2 2.6289 nan 0.1000 0.2985  
## 3 2.3959 nan 0.1000 0.2322  
## 4 2.1669 nan 0.1000 0.2007  
## 5 1.9341 nan 0.1000 0.1282  
## 6 1.7539 nan 0.1000 0.1235  
## 7 1.6086 nan 0.1000 0.0934  
## 8 1.4762 nan 0.1000 0.0790  
## 9 1.3644 nan 0.1000 0.0275  
## 10 1.2822 nan 0.1000 0.0571  
## 20 0.7344 nan 0.1000 -0.0049  
## 40 0.3535 nan 0.1000 -0.0067  
## 60 0.1977 nan 0.1000 -0.0028  
## 80 0.1295 nan 0.1000 -0.0012  
## 100 0.0901 nan 0.1000 -0.0001  
## 120 0.0579 nan 0.1000 -0.0009  
## 140 0.0395 nan 0.1000 -0.0006  
## 150 0.0325 nan 0.1000 -0.0002  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 3.0812 nan 0.1000 0.3265  
## 2 2.7358 nan 0.1000 0.2213  
## 3 2.4736 nan 0.1000 0.2279  
## 4 2.2640 nan 0.1000 0.1816  
## 5 2.0698 nan 0.1000 0.1617  
## 6 1.8866 nan 0.1000 0.1125  
## 7 1.7093 nan 0.1000 0.1259  
## 8 1.6250 nan 0.1000 0.0529  
## 9 1.5190 nan 0.1000 0.0830  
## 10 1.4228 nan 0.1000 0.0500  
## 20 0.9278 nan 0.1000 0.0100  
## 40 0.5500 nan 0.1000 -0.0055  
## 60 0.3775 nan 0.1000 -0.0016  
## 80 0.2551 nan 0.1000 -0.0052  
## 100 0.1763 nan 0.1000 -0.0005

plot(gbm.fit)



### The best performing model is the random forest with an value at 61.8.

rf.fit$finalModel

##   
## Call:  
## randomForest(x = x, y = y, mtry = param$mtry, importance = TRUE)   
## Type of random forest: regression  
## Number of trees: 500  
## No. of variables tried at each split: 34  
##   
## Mean of squared residuals: 1.359705  
## % Var explained: 61.12

### b. Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models?

varImp(rf.fit, scale = FALSE)

## rf variable importance  
##   
## only 20 most important variables shown (out of 67)  
##   
## Overall  
## ManufacturingProcess32 25.016  
## ManufacturingProcess31 10.789  
## ManufacturingProcess17 9.345  
## ManufacturingProcess13 9.003  
## BiologicalMaterial03 8.801  
## BiologicalMaterial06 8.518  
## BiologicalMaterial12 6.293  
## ManufacturingProcess11 6.107  
## ManufacturingProcess36 5.762  
## ManufacturingProcess09 5.365  
## BiologicalMaterial05 5.286  
## BiologicalMaterial02 5.109  
## ManufacturingProcess43 5.104  
## ManufacturingProcess25 5.018  
## BiologicalMaterial11 5.005  
## ManufacturingProcess30 4.926  
## ManufacturingProcess28 4.519  
## BiologicalMaterial09 4.381  
## ManufacturingProcess39 4.354  
## BiologicalMaterial04 4.336

ManufacturingProcess32 is the most important followed by ManufacturingProcess17, BiologicalMaterial03, and ManufacturingProcess13 in that order. The Manufacturing Processes dominate the list and hold the highest importance compared to other variables.

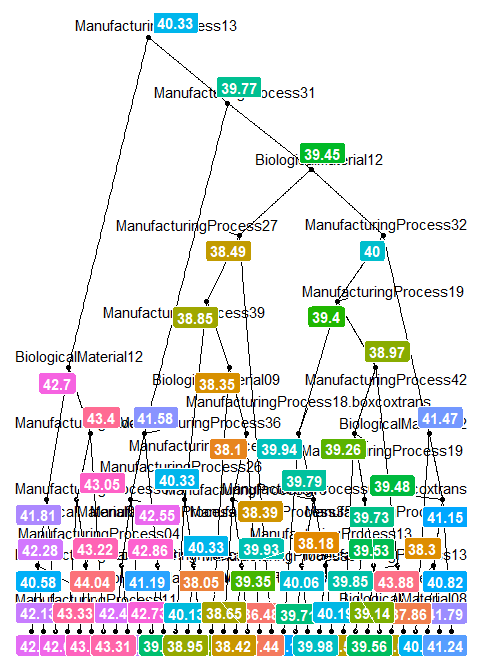
The top 10 lists between the three methods is almost completely different. However, the non-linear method and the forest models both have ManufacturingProcess32 as the most important variable.

### c. Plot the optimal single tree with the distribution yields in the terminal nodes. Does this view of the data provide additional knowledge about the biological or process predictors and their relationship with yield?

Random Forest does not natively allow for a plot of this type. I am assuming that the author envisioned another model making the being the best model as my model cannot be used in this way.

Here is a plot of the tree and interestingly we see that the biological process 12 is the first split and then we our manufacturing nodes splitting the tree and the next level.

#https://shiring.github.io/machine\_learning/2017/03/16/rf\_plot\_ggraph  
  
library(dplyr)  
library(ggraph)  
library(igraph)  
  
tree\_func <- function(final\_model,   
 tree\_num) {  
   
 # get tree by index  
 tree <- randomForest::getTree(final\_model,   
 k = tree\_num,   
 labelVar = TRUE) %>%  
 tibble::rownames\_to\_column() %>%  
 # make leaf split points to NA, so the 0s won't get plotted  
 mutate(`split point` = ifelse(is.na(prediction), `split point`, NA))  
   
 # prepare data frame for graph  
 graph\_frame <- data.frame(from = rep(tree$rowname, 2),  
 to = c(tree$`left daughter`, tree$`right daughter`))  
   
 # convert to graph and delete the last node that we don't want to plot  
 graph <- graph\_from\_data\_frame(graph\_frame) %>%  
 delete\_vertices("0")  
   
 # set node labels  
 V(graph)$node\_label <- gsub("\_", " ", as.character(tree$`split var`))  
 V(graph)$leaf\_label <- as.character(round(tree$prediction,2))  
 V(graph)$split <- as.character(round(tree$`split point`, digits = 2))  
   
 # plot  
 plot <- ggraph(graph, 'dendrogram') +   
 theme\_bw() +  
 geom\_edge\_link() +  
 geom\_node\_point() +  
 geom\_node\_text(aes(label = node\_label), na.rm = TRUE, repel = TRUE) +  
 geom\_node\_label(aes(label = split), vjust = 2.5, na.rm = TRUE, fill = "white") +  
 geom\_node\_label(aes(label = leaf\_label, fill = leaf\_label), na.rm = TRUE,   
 repel = TRUE, colour = "white", fontface = "bold", show.legend = FALSE) +  
 theme(panel.grid.minor = element\_blank(),  
 panel.grid.major = element\_blank(),  
 panel.background = element\_blank(),  
 plot.background = element\_rect(fill = "white"),  
 panel.border = element\_blank(),  
 axis.line = element\_blank(),  
 axis.text.x = element\_blank(),  
 axis.text.y = element\_blank(),  
 axis.ticks = element\_blank(),  
 axis.title.x = element\_blank(),  
 axis.title.y = element\_blank(),  
 plot.title = element\_text(size = 18))  
   
 print(plot)  
}  
  
tree\_num <- max(which(rf.fit$finalModel$forest$ndbigtree == min(rf.fit$finalModel$forest$ndbigtree)))  
tree\_func(final\_model = rf.fit$finalModel, tree\_num = tree\_num)



# Recommender Systems

Imagine 10000 receipts sitting on your table. Each receipt represents a transaction with items that were purchased. The receipt is a representation of stuff that went into a customer’s basket - and therefore ‘Market Basket Analysis’.

That is exactly what the Groceries Data Set contains: a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a transaction and each column in a row represents an item.

Here is the dataset = GroceryDataSet.csv (comma separated file)

You assignment is to use R to mine the data for association rules. You should report support, confidence and lift and your top 10 rules by lift.

### Loading the Data

df <- read.transactions("https://raw.githubusercontent.com/ChristopheHunt/MSDA---Coursework/master/Data%20624/Homework%2011/GroceryDataSet.csv", sep = ",", format = c("basket"))

Let’s look at the first 5 transactions to understand how the data is being loaded.

inspect(df[1:5])

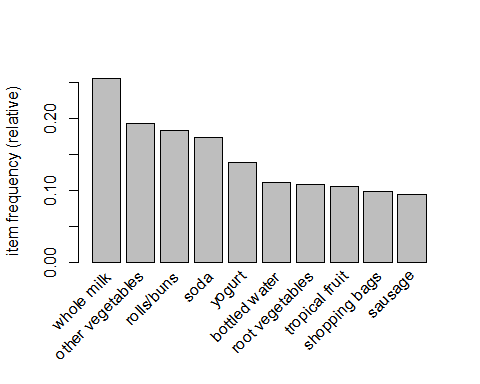
## items   
## [1] {citrus fruit,   
## margarine,   
## ready soups,   
## semi-finished bread}   
## [2] {coffee,   
## tropical fruit,   
## yogurt}   
## [3] {whole milk}   
## [4] {cream cheese,   
## meat spreads,   
## pip fruit,   
## yogurt}   
## [5] {condensed milk,   
## long life bakery product,  
## other vegetables,   
## whole milk}

Now we can review common occurences. We see that vgetables and milk are at the top of the list, which is reassuring.

summary(df)

## transactions as itemMatrix in sparse format with  
## 9835 rows (elements/itemsets/transactions) and  
## 169 columns (items) and a density of 0.02609146   
##   
## most frequent items:  
## whole milk other vegetables rolls/buns soda   
## 2513 1903 1809 1715   
## yogurt (Other)   
## 1372 34055   
##   
## element (itemset/transaction) length distribution:  
## sizes  
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15   
## 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55   
## 16 17 18 19 20 21 22 23 24 26 27 28 29 32   
## 46 29 14 14 9 11 4 6 1 1 1 1 3 1   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 2.000 3.000 4.409 6.000 32.000   
##   
## includes extended item information - examples:  
## labels  
## 1 abrasive cleaner  
## 2 artif. sweetener  
## 3 baby cosmetics

itemFrequencyPlot(df, topN = 10)



Further we can run the apriori function to determine the associated rules. Because we have so many different choices to purchase we will set a lower support. 100 out 9835 gives us a support value of 0.01016777, we will use an arbitrary confidence of .5.

rules <- apriori(df, parameter = list(support = 0.01016777, confidence = 0.5))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support  
## 0.5 0.1 1 none FALSE TRUE 5 0.01016777  
## minlen maxlen target ext  
## 1 10 rules FALSE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 100   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
## sorting and recoding items ... [88 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 done [0.00s].  
## writing ... [14 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

inspect(sort(rules, by = "lift")[1:10])

## lhs rhs support confidence lift count  
## [1] {citrus fruit,   
## root vegetables} => {other vegetables} 0.01037112 0.5862069 3.029608 102  
## [2] {root vegetables,   
## tropical fruit} => {other vegetables} 0.01230300 0.5845411 3.020999 121  
## [3] {rolls/buns,   
## root vegetables} => {other vegetables} 0.01220132 0.5020921 2.594890 120  
## [4] {root vegetables,   
## yogurt} => {other vegetables} 0.01291307 0.5000000 2.584078 127  
## [5] {butter,   
## other vegetables} => {whole milk} 0.01148958 0.5736041 2.244885 113  
## [6] {root vegetables,   
## tropical fruit} => {whole milk} 0.01199797 0.5700483 2.230969 118  
## [7] {root vegetables,   
## yogurt} => {whole milk} 0.01453991 0.5629921 2.203354 143  
## [8] {domestic eggs,   
## other vegetables} => {whole milk} 0.01230300 0.5525114 2.162336 121  
## [9] {whipped/sour cream,   
## yogurt} => {whole milk} 0.01087951 0.5245098 2.052747 107  
## [10] {rolls/buns,   
## root vegetables} => {whole milk} 0.01270971 0.5230126 2.046888 125

Reporting our top rules by lift we see interesting results. It appears that buying vegetables and fruits suggests that similar purchases of other vegetables are possible. We also see those buying butter, eggs, and other animal products are more apt to buy whole milk. The exercise is informative of the power in this type of analysis.

# Appendix with all code

All code used previous in below section

library(knitr)  
opts\_chunk$set(cache = TRUE, message=FALSE, warning=FALSE)  
library(AppliedPredictiveModeling)  
library(Amelia)  
library(doParallel)  
library(missForest)  
library(cwhmisc)  
library(MASS)  
library(forecast)  
library(tidyverse)  
library(pls)  
library(caret)  
library(data.table)  
library(psych)  
library(mlbench)  
library(nnet)  
library(earth)  
library(kernlab)  
library(Metrics)  
library(party)  
library(AppliedPredictiveModeling)  
library(Amelia)  
library(doParallel)  
library(missForest)  
library(cwhmisc)  
library(MASS)  
library(forecast)  
library(tidyverse)  
library(pls)  
library(caret)  
library(data.table)  
library(psych)  
library(mlbench)  
library(nnet)  
library(earth)  
library(kernlab)  
library(Metrics)  
library(party)  
library(AppliedPredictiveModeling)  
data("ChemicalManufacturingProcess")  
library(Amelia)  
missmap(ChemicalManufacturingProcess)  
library(doParallel)  
library(missForest)  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100, parallelize = "forests")$ximp  
data.set <- list(org.data.set = chem.imput)  
library(psych)  
  
multi.hist(data.set$org.data.set[1:8])  
  
multi.hist(data.set$org.data.set[9:16])  
  
multi.hist(data.set$org.data.set[17:24])  
  
multi.hist(data.set$org.data.set[25:32])  
variables.to.transform <- c("BiologicalMaterial04", "ManufacturingProcess01",  
 "ManufacturingProcess02","ManufacturingProcess03",  
 "ManufacturingProcess06", "ManufacturingProcess07",  
 "ManufacturingProcess08","ManufacturingProcess12",  
 "ManufacturingProcess16","ManufacturingProcess18")   
  
results <- NULL  
for (i in 1:length(variables.to.transform)){  
 results[[i]] <- list(variable = variables.to.transform[[i]],  
 lambda = BoxCox.lambda(data.set$org.data.set[,variables.to.transform[[i]]]))}  
results  
trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))  
set.seed(123)  
data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))  
library(pls)  
chem.plsFit <- plsr(Yield ~ ., data = data.set$train, validation = "CV")  
validationplot(chem.plsFit, val.type="RMSEP")  
pls.RMSEP <- RMSEP(chem.plsFit, estimate="CV")   
plot(pls.RMSEP, main="RMSEP PLS Solubility", xlab="components")  
min\_comp <- which.min(pls.RMSEP$val)  
points(min\_comp, min(pls.RMSEP$val), pch=1, col="red", cex=1.5)  
chem.plsPredict.train <- predict(chem.plsFit, data = data.set$train, ncomp = min\_comp)  
pls.eval <- data.frame(obs = data.set$train[,1], pred = chem.plsPredict.train[,1,1])  
defaultSummary(pls.eval)  
chem.plsPredict <- predict(chem.plsFit, data = data.set$test, ncomp = min\_comp)  
pls.eval <- data.frame(obs = data.set$test[,1], pred = chem.plsPredict[,1,1])  
defaultSummary(pls.eval)  
library(data.table)  
dt <- setDT(varImp(chem.plsFit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]  
data.set$org.data.set %>% dplyr::select(Yield, ManufacturingProcess40) %>% cor() %>% .[2]  
data.set$org.data.set %>% dplyr::select(Yield, BiologicalMaterial12) %>% cor() %>% .[2]  
set.seed(1234)  
trainingData <- mlbench.friedman1(200, sd = 1)  
trainingData$x <- data.frame(trainingData$x)  
#featurePlot(trainingData$x, trainingData$y)  
testData <- mlbench.friedman1(5000, sd = 1)  
testData$x <- data.frame(testData$x)  
nnetFit <- nnet(testData$x, testData$y,  
 size = 5,  
 decay = .01,  
 maxit = 500,  
 MaxNWts = 5 \* (ncol(testData$x) + 1) + 5 + 1)  
  
results <- predict(nnetFit, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)  
marsFit <- earth(testData$x, testData$y)  
results <- predict(marsFit, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)  
## #At one point I got the error optimismBoot not found, installing caret from github resolved the issue.  
## knnTune <- train(testData$x, testData$y,  
## method = "knn",  
## preProc = c("center", "scale"),  
## trace = FALSE)  
## results <- predict(knnTune, trainingData$x) %>% cbind(.,trainingData$y) %>% as.data.frame()  
## colnames(results) <- c("predicted", "actual")  
## rmse(results$actual, results$predicted)  
varImp(marsFit)  
library(AppliedPredictiveModeling)  
data("ChemicalManufacturingProcess")  
library(missForest)  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100)$ximp  
data.set <- list(org.data.set = chem.imput)  
trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))  
data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))  
nnetFit <- nnet(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 size = 5,  
 decay = .01,  
 maxit = 500,  
 trace = FALSE)  
  
results <- predict(nnetFit,data.set$test %>% dplyr::select(-Yield)) %>% cbind(.,data.set$test$Yield) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)  
marsFit <- earth(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield)  
results <- predict(marsFit, data.set$test %>% dplyr::select(-Yield)) %>% cbind(.,data.set$test$Yield) %>% as.data.frame()  
colnames(results) <- c("predicted", "actual")  
rmse(results$actual, results$predicted)  
library(data.table)  
dt <- setDT(varImp(marsFit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]  
subset.data.set <- data.set$trans.data.set %>%   
 dplyr::select(ManufacturingProcess32, ManufacturingProcess09,   
 ManufacturingProcess13, ManufacturingProcess33,   
 BiologicalMaterial02, BiologicalMaterial03,   
 ManufacturingProcess28, ManufacturingProcess43,   
 ManufacturingProcess04, BiologicalMaterial05, Yield)  
library(corrplot)  
correlations <- cor(subset.data.set)  
corrplot(correlations, order = "FPC", diag = TRUE)  
library(tidyverse)  
library(mlbench)  
set.seed(200)  
simulated <- mlbench.friedman1(200, sd = 1)  
simulated <- cbind(simulated$x, simulated$y)  
simulated <- as.data.frame(simulated)  
colnames(simulated)[ncol(simulated)] <- "y"  
library(randomForest)  
library(caret)  
model1 <- randomForest(y ~., data = simulated, importance = TRUE,  
 ntree = 1000)  
rfImp <- varImp(model1, scale = FALSE)  
rfImp  
simulated$duplicate1 <- simulated$V1 + rnorm(200) \* .1  
cor(simulated$duplicate1, simulated$V1)  
model1 <- randomForest(y ~., data = simulated, importance = TRUE,  
 ntree = 1000)  
rfImp <- varImp(model1, scale = FALSE)  
library(data.table)  
dt <- setDT(varImp(model1, scale = FALSE), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]  
simulated$duplicate2 <- simulated$V1 + rnorm(200) \* .15  
cor(simulated$duplicate2, simulated$V1)  
model2 <- randomForest(y ~., data = simulated, importance = TRUE,  
 ntree = 1000)  
rfImp <- varImp(model2, scale = FALSE)  
library(data.table)  
dt <- setDT(varImp(model2, scale = FALSE), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]  
library(party)  
fit.cforest <- cforest(y ~., data = simulated)  
library(data.table)  
dt <- setDT(party::varimp(fit.cforest, conditional = TRUE) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-.)]  
dt <- setDT(party::varimp(fit.cforest, conditional = FALSE) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-.)]  
library(Cubist)  
fit.cubist <- cubist(simulated %>% select(-y), simulated$y, committees = 4)  
fit.cubist  
dt <- setDT(varImp(fit.cubist) %>% data.frame(), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]  
library(mlbench)  
library(tidyverse)  
library(rpart)  
library(rpart.plot)  
library(caret)  
  
data.set <- twoClassSim(300, noiseVars = 8, corrVar = 6, corrValue = 0.8) %>%  
 mutate(TwoFactor1 = as.factor(round(TwoFactor1, 0)),  
 TwoFactor2 = as.factor(round(TwoFactor2, 0)))  
r.fit <- rpart(Linear01 ~ ., data=data.set)  
rpart.plot(r.fit)  
library(data.table)  
dt <- setDT(varImp(r.fit), keep.rownames = TRUE)[]  
options(scipen=99)  
dt[order(-Overall)]  
library(missForest)  
library(AppliedPredictiveModeling)  
library(doParallel)  
data("ChemicalManufacturingProcess")  
  
set.seed(1234)  
  
registerDoParallel(cores=3)  
chem.imput <- missForest(ChemicalManufacturingProcess, ntree = 100)$ximp  
data.set <- list(org.data.set = chem.imput)  
library(tidyverse)  
library(caret)  
library(forecast)  
library(MASS)  
  
trans.data.set <- data.set$org.data.set %>%   
 mutate(BiologicalMaterial04.boxcoxtrans =   
 BoxCox(BiologicalMaterial04, -0.99992424816297),  
 ManufacturingProcess01.boxcoxtrans =   
 BoxCox(ManufacturingProcess01, 1.99995900720725),  
 ManufacturingProcess02.boxcoxtrans =   
 BoxCox(ManufacturingProcess02, 1.99995900720725),  
 ManufacturingProcess03.boxcoxtrans =   
 BoxCox(ManufacturingProcess03, -0.99992424816297),  
 ManufacturingProcess06.boxcoxtrans =   
 BoxCox(ManufacturingProcess06, -0.99992424816297),  
 ManufacturingProcess07.boxcoxtrans =   
 BoxCox(ManufacturingProcess07,-0.99992424816297),  
 ManufacturingProcess08.boxcoxtrans =   
 BoxCox(ManufacturingProcess08, 1.99992424816297),  
 ManufacturingProcess12.boxcoxtrans =   
 BoxCox(ManufacturingProcess12, 0.0000410225926326142),  
 ManufacturingProcess16.boxcoxtrans =   
 BoxCox(ManufacturingProcess16, 1.99995900720725),  
 ManufacturingProcess18.boxcoxtrans =   
 BoxCox(ManufacturingProcess18, 1.99995900720725))  
data.set <- c(data.set, list(trans.data.set = trans.data.set)) %>%   
 c(., list(train.test.split = floor(0.75 \* nrow(chem.imput)))) %>%   
 c(., train.index = list(sample(seq\_len(nrow(chem.imput)), size = .$train.test.split))) %>%  
 c(., train = list(as.data.frame(trans.data.set[.$train.index,])), test = list(as.data.frame(trans.data.set[-.$train.index,])))  
ctrl <- trainControl(method = "boot", number = 15)  
  
rpart.fit <- train(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 method = "rpart",  
 metric = "Rsquared",  
 trControl = ctrl)  
plot(rpart.fit)  
rf.fit <- train(data.set$train %>% dplyr::select(-Yield), data.set$train$Yield,  
 method = "rf",  
 metric = "Rsquared",  
 trControl = ctrl,  
 importance = TRUE)  
plot(rf.fit)  
gbm.fit <- train(data.set$train %>%   
 dplyr::select(-Yield), data.set$train$Yield,  
 method = "gbm",  
 metric = "Rsquared",  
 trControl = ctrl)  
  
plot(gbm.fit)  
rf.fit$finalModel  
varImp(rf.fit, scale = FALSE)  
#https://shiring.github.io/machine\_learning/2017/03/16/rf\_plot\_ggraph  
  
library(dplyr)  
library(ggraph)  
library(igraph)  
  
tree\_func <- function(final\_model,   
 tree\_num) {  
   
 # get tree by index  
 tree <- randomForest::getTree(final\_model,   
 k = tree\_num,   
 labelVar = TRUE) %>%  
 tibble::rownames\_to\_column() %>%  
 # make leaf split points to NA, so the 0s won't get plotted  
 mutate(`split point` = ifelse(is.na(prediction), `split point`, NA))  
   
 # prepare data frame for graph  
 graph\_frame <- data.frame(from = rep(tree$rowname, 2),  
 to = c(tree$`left daughter`, tree$`right daughter`))  
   
 # convert to graph and delete the last node that we don't want to plot  
 graph <- graph\_from\_data\_frame(graph\_frame) %>%  
 delete\_vertices("0")  
   
 # set node labels  
 V(graph)$node\_label <- gsub("\_", " ", as.character(tree$`split var`))  
 V(graph)$leaf\_label <- as.character(round(tree$prediction,2))  
 V(graph)$split <- as.character(round(tree$`split point`, digits = 2))  
   
 # plot  
 plot <- ggraph(graph, 'dendrogram') +   
 theme\_bw() +  
 geom\_edge\_link() +  
 geom\_node\_point() +  
 geom\_node\_text(aes(label = node\_label), na.rm = TRUE, repel = TRUE) +  
 geom\_node\_label(aes(label = split), vjust = 2.5, na.rm = TRUE, fill = "white") +  
 geom\_node\_label(aes(label = leaf\_label, fill = leaf\_label), na.rm = TRUE,   
 repel = TRUE, colour = "white", fontface = "bold", show.legend = FALSE) +  
 theme(panel.grid.minor = element\_blank(),  
 panel.grid.major = element\_blank(),  
 panel.background = element\_blank(),  
 plot.background = element\_rect(fill = "white"),  
 panel.border = element\_blank(),  
 axis.line = element\_blank(),  
 axis.text.x = element\_blank(),  
 axis.text.y = element\_blank(),  
 axis.ticks = element\_blank(),  
 axis.title.x = element\_blank(),  
 axis.title.y = element\_blank(),  
 plot.title = element\_text(size = 18))  
   
 print(plot)  
}  
  
tree\_num <- max(which(rf.fit$finalModel$forest$ndbigtree == min(rf.fit$finalModel$forest$ndbigtree)))  
tree\_func(final\_model = rf.fit$finalModel, tree\_num = tree\_num)  
library(arules)  
df <- read.transactions("https://raw.githubusercontent.com/ChristopheHunt/MSDA---Coursework/master/Data%20624/Homework%2011/GroceryDataSet.csv", sep = ",", format = c("basket"))  
inspect(df[1:5])  
summary(df)  
itemFrequencyPlot(df, topN = 10)  
rules <- apriori(df, parameter = list(support = 0.01016777, confidence = 0.5))  
inspect(sort(rules, by = "lift")[1:10])  
## NA