Final Draft

# Overview

### Abstract

The purpose of this report is to address the new regulations and to better understand the manufacturing process within ABC Beverage. We will examine all factors involved in the production process and will attempt to identify the factors that will help us properly predict the PH levels as well us understand the influence of the various factors on the overall process.

### Data used

This report is using the historical data collected from approximately 2572 samples which should be sufficient for the analysis.

### Brief overview of the process

We will first cleanup the data, by filling in or imputing the missing data, use various transformation methods in order to normalize the data to address issues such as outlier data points and other normalization related issues.

Next we will run various models in order to identify the factors that are important to reaching out goal and once we have those we will use various models in order to estimate the PH levels and come up with a best suiting method that we feel will be best to predict the data we are looking for.

We will include documented R code within the report so that it is easy to follow our research. Should you have any questions regarding the process or the code, feel free to reach out to our department.

# Data Exploration

We will get started by loading our historical data into a data frame and loading the necessary libraries.

library(missForest)  
library(corrgram)  
library(caret)  
library(psych)  
library(knitr)  
  
dfBevMod <- read.csv("https://github.com/ChristopheHunt/CUNY-DATA624/raw/master/data/StudentData.csv", header = TRUE)  
dfBevPred <- read.csv("https://raw.githubusercontent.com/ChristopheHunt/CUNY-DATA624/master/data/StudentEvaluation-%20TO%20PREDICT.csv", header =TRUE)

We will examine the training dataset we want to see how many predictor variables we are dealing with and if we are missing any data. We see several variables with missing data,

dim(dfBevMod)

## [1] 2571 33

It appears we have a total of 32 predictor variables and a target variable. Next we will check for any missing data.

colSums(is.na(dfBevMod))

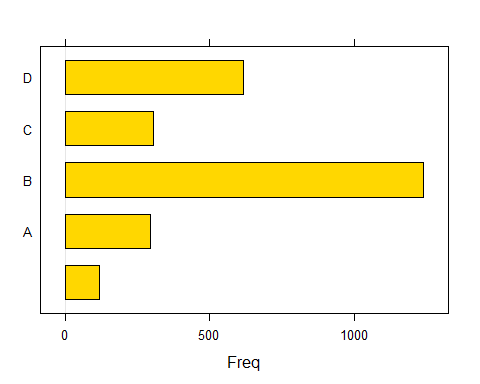
## Brand.Code Carb.Volume Fill.Ounces PC.Volume   
## 0 10 38 39   
## Carb.Pressure Carb.Temp PSC PSC.Fill   
## 27 26 33 23   
## PSC.CO2 Mnf.Flow Carb.Pressure1 Fill.Pressure   
## 39 2 32 22   
## Hyd.Pressure1 Hyd.Pressure2 Hyd.Pressure3 Hyd.Pressure4   
## 11 15 15 30   
## Filler.Level Filler.Speed Temperature Usage.cont   
## 20 57 14 5   
## Carb.Flow Density MFR Balling   
## 2 1 212 1   
## Pressure.Vacuum PH Oxygen.Filler Bowl.Setpoint   
## 0 4 12 2   
## Pressure.Setpoint Air.Pressurer Alch.Rel Carb.Rel   
## 12 0 9 10   
## Balling.Lvl   
## 1

We see many variables with NA’s – notably “MFR” has 212. Still, roughly 8% NA is workable, so we’ll choose to use imputation process to fill in the missing data.

Also, it appears that we have one categorical variable: Brand.Code

### Barchart

#Visualizing the single categorical variable  
barchart(dfBevMod[,1], col="Gold")



It appears that brand “B” occurs most frequently, followed by “D”

### Continuous and Discrete variables

Next we will review all the variables we are working with in order to better understand the data they are presenting us. We can see the mean, variation and other metrics within the following table for a quick detailed reference.

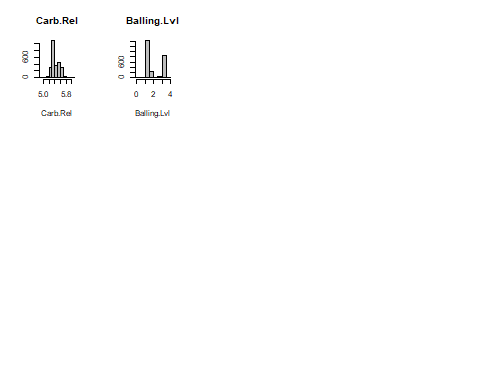
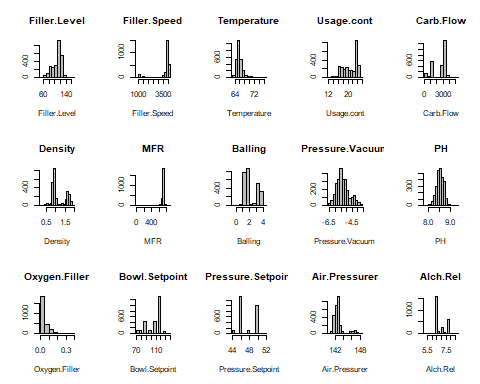
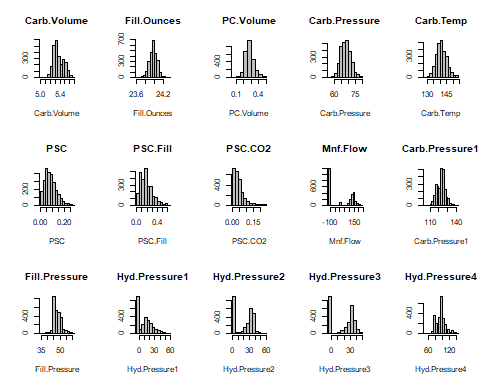
library(psych)  
library(knitr)  
table.desc <- describe(dfBevMod[,-1])  
table.prep <- as.matrix(table.desc)  
table.round <- round((table.prep), 2)  
kable(table.round)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | vars | n | mean | sd | median | trimmed | mad | min | max | range | skew | kurtosis | se |
| Carb.Volume | 1 | 2561 | 5.37 | 0.11 | 5.35 | 5.37 | 0.11 | 5.04 | 5.70 | 0.66 | 0.39 | -0.47 | 0.00 |
| Fill.Ounces | 2 | 2533 | 23.97 | 0.09 | 23.97 | 23.98 | 0.08 | 23.63 | 24.32 | 0.69 | -0.02 | 0.86 | 0.00 |
| PC.Volume | 3 | 2532 | 0.28 | 0.06 | 0.27 | 0.27 | 0.05 | 0.08 | 0.48 | 0.40 | 0.34 | 0.67 | 0.00 |
| Carb.Pressure | 4 | 2544 | 68.19 | 3.54 | 68.20 | 68.12 | 3.56 | 57.00 | 79.40 | 22.40 | 0.18 | -0.01 | 0.07 |
| Carb.Temp | 5 | 2545 | 141.09 | 4.04 | 140.80 | 140.99 | 3.85 | 128.60 | 154.00 | 25.40 | 0.25 | 0.24 | 0.08 |
| PSC | 6 | 2538 | 0.08 | 0.05 | 0.08 | 0.08 | 0.05 | 0.00 | 0.27 | 0.27 | 0.85 | 0.65 | 0.00 |
| PSC.Fill | 7 | 2548 | 0.20 | 0.12 | 0.18 | 0.18 | 0.12 | 0.00 | 0.62 | 0.62 | 0.93 | 0.77 | 0.00 |
| PSC.CO2 | 8 | 2532 | 0.06 | 0.04 | 0.04 | 0.05 | 0.03 | 0.00 | 0.24 | 0.24 | 1.73 | 3.73 | 0.00 |
| Mnf.Flow | 9 | 2569 | 24.57 | 119.48 | 65.20 | 21.07 | 169.02 | -100.20 | 229.40 | 329.60 | 0.00 | -1.87 | 2.36 |
| Carb.Pressure1 | 10 | 2539 | 122.59 | 4.74 | 123.20 | 122.54 | 4.45 | 105.60 | 140.20 | 34.60 | 0.05 | 0.14 | 0.09 |
| Fill.Pressure | 11 | 2549 | 47.92 | 3.18 | 46.40 | 47.71 | 2.37 | 34.60 | 60.40 | 25.80 | 0.55 | 1.41 | 0.06 |
| Hyd.Pressure1 | 12 | 2560 | 12.44 | 12.43 | 11.40 | 10.84 | 16.90 | -0.80 | 58.00 | 58.80 | 0.78 | -0.14 | 0.25 |
| Hyd.Pressure2 | 13 | 2556 | 20.96 | 16.39 | 28.60 | 21.05 | 13.34 | 0.00 | 59.40 | 59.40 | -0.30 | -1.56 | 0.32 |
| Hyd.Pressure3 | 14 | 2556 | 20.46 | 15.98 | 27.60 | 20.51 | 13.94 | -1.20 | 50.00 | 51.20 | -0.32 | -1.57 | 0.32 |
| Hyd.Pressure4 | 15 | 2541 | 96.29 | 13.12 | 96.00 | 95.45 | 11.86 | 52.00 | 142.00 | 90.00 | 0.55 | 0.63 | 0.26 |
| Filler.Level | 16 | 2551 | 109.25 | 15.70 | 118.40 | 111.04 | 9.19 | 55.80 | 161.20 | 105.40 | -0.85 | 0.05 | 0.31 |
| Filler.Speed | 17 | 2514 | 3687.20 | 770.82 | 3982.00 | 3919.99 | 47.44 | 998.00 | 4030.00 | 3032.00 | -2.87 | 6.71 | 15.37 |
| Temperature | 18 | 2557 | 65.97 | 1.38 | 65.60 | 65.80 | 0.89 | 63.60 | 76.20 | 12.60 | 2.39 | 10.16 | 0.03 |
| Usage.cont | 19 | 2566 | 20.99 | 2.98 | 21.79 | 21.25 | 3.19 | 12.08 | 25.90 | 13.82 | -0.54 | -1.02 | 0.06 |
| Carb.Flow | 20 | 2569 | 2468.35 | 1073.70 | 3028.00 | 2601.14 | 326.17 | 26.00 | 5104.00 | 5078.00 | -0.99 | -0.58 | 21.18 |
| Density | 21 | 2570 | 1.17 | 0.38 | 0.98 | 1.15 | 0.15 | 0.24 | 1.92 | 1.68 | 0.53 | -1.20 | 0.01 |
| MFR | 22 | 2359 | 704.05 | 73.90 | 724.00 | 718.16 | 15.42 | 31.40 | 868.60 | 837.20 | -5.09 | 30.46 | 1.52 |
| Balling | 23 | 2570 | 2.20 | 0.93 | 1.65 | 2.13 | 0.37 | -0.17 | 4.01 | 4.18 | 0.59 | -1.39 | 0.02 |
| Pressure.Vacuum | 24 | 2571 | -5.22 | 0.57 | -5.40 | -5.25 | 0.59 | -6.60 | -3.60 | 3.00 | 0.53 | -0.03 | 0.01 |
| PH | 25 | 2567 | 8.55 | 0.17 | 8.54 | 8.55 | 0.18 | 7.88 | 9.36 | 1.48 | -0.29 | 0.06 | 0.00 |
| Oxygen.Filler | 26 | 2559 | 0.05 | 0.05 | 0.03 | 0.04 | 0.02 | 0.00 | 0.40 | 0.40 | 2.66 | 11.09 | 0.00 |
| Bowl.Setpoint | 27 | 2569 | 109.33 | 15.30 | 120.00 | 111.35 | 0.00 | 70.00 | 140.00 | 70.00 | -0.97 | -0.06 | 0.30 |
| Pressure.Setpoint | 28 | 2559 | 47.62 | 2.04 | 46.00 | 47.60 | 0.00 | 44.00 | 52.00 | 8.00 | 0.20 | -1.60 | 0.04 |
| Air.Pressurer | 29 | 2571 | 142.83 | 1.21 | 142.60 | 142.58 | 0.59 | 140.80 | 148.20 | 7.40 | 2.25 | 4.73 | 0.02 |
| Alch.Rel | 30 | 2562 | 6.90 | 0.51 | 6.56 | 6.84 | 0.06 | 5.28 | 8.62 | 3.34 | 0.88 | -0.85 | 0.01 |
| Carb.Rel | 31 | 2561 | 5.44 | 0.13 | 5.40 | 5.43 | 0.12 | 4.96 | 6.06 | 1.10 | 0.50 | -0.29 | 0.00 |
| Balling.Lvl | 32 | 2570 | 2.05 | 0.87 | 1.48 | 1.98 | 0.21 | 0.00 | 3.66 | 3.66 | 0.59 | -1.49 | 0.02 |

### Nominal Variable Histogram

Next we will visually view each one of the factors, its easier to visually navigate through a large number of variables. We are interested to see how data is distributed for each one of the variables. Please refer to above table for more specific information.

dfBevModH <- dfBevMod[2:ncol(dfBevMod)] #removing factor var  
par(mfrow = c(3,5), cex = .5)  
for(i in colnames(dfBevModH)){  
hist(dfBevModH[,i], xlab = names(dfBevMod[i]),  
 main = names(dfBevModH[i]), col="grey", ylab="")  
}



We can see that Mnf.Flow and Hyd.Pressure 1,2,3 each have many values below 0 – possibly null-type entered values. Several variables are strongly skewed – some of which appear to have outliers.

Next we want to explore differences by Brand

BrandA <- dfBevMod[dfBevMod$Brand.Code == "A",]  
BAM <- colMeans(BrandA[,2:ncol(BrandA)], na.rm = TRUE)  
BrandB <- dfBevMod[dfBevMod$Brand.Code == "B",]  
BBM <- colMeans(BrandB[,2:ncol(BrandB)], na.rm = TRUE)  
BrandC <- dfBevMod[dfBevMod$Brand.Code == "C",]  
BCM <- colMeans(BrandC[,2:ncol(BrandC)], na.rm = TRUE)  
BrandD <- dfBevMod[dfBevMod$Brand.Code == "D",]  
BDM <- colMeans(BrandD[,2:ncol(BrandD)], na.rm = TRUE)  
BrandE <- dfBevMod[dfBevMod$Brand.Code == "",]  
BEM <- colMeans(BrandE[,2:ncol(BrandE)], na.rm = TRUE)  
  
combBrand <- cbind(BAM, BBM, BCM, BDM, BEM)  
combBrand

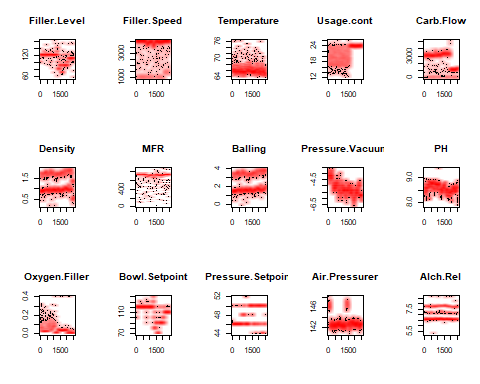
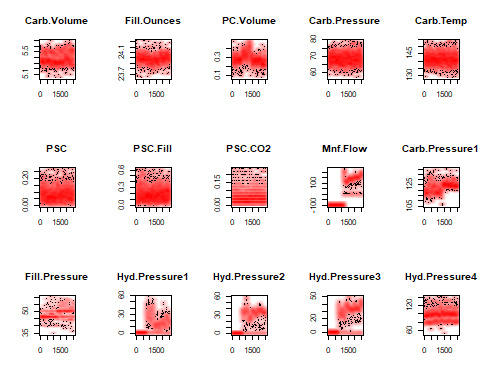
## BAM BBM BCM BDM  
## Carb.Volume 5.42568828 5.31248514 5.30019802 5.51044190  
## Fill.Ounces 23.98210526 23.97748705 23.98486667 23.95658456  
## PC.Volume 0.26877855 0.28239989 0.29019778 0.26174013  
## Carb.Pressure 69.28041237 67.19804719 66.92476821 70.69817579  
## Carb.Temp 141.22876712 141.11078431 141.05364238 141.13013115  
## PSC 0.07761672 0.08635948 0.08976821 0.08056579  
## PSC.Fill 0.19835052 0.19371847 0.21026490 0.19104918  
## PSC.CO2 0.05551724 0.05709360 0.06214765 0.05276316  
## Mnf.Flow 39.71467577 20.46176233 23.28092105 25.74666667  
## Carb.Pressure1 122.81862069 122.47500000 122.19401993 122.72363636  
## Fill.Pressure 48.22123288 48.17010561 48.22866667 46.96124795  
## Hyd.Pressure1 12.90238908 12.52469636 12.32828947 12.39671053  
## Hyd.Pressure2 21.15821918 21.10510949 19.16250000 21.97331137  
## Hyd.Pressure3 20.97465753 20.19253852 19.04671053 21.59011532  
## Hyd.Pressure4 101.28368794 100.05858421 102.54000000 83.45276873  
## Filler.Level 108.62671233 107.96156352 111.60198020 110.42262295  
## Filler.Speed 3582.07719298 3730.61943987 3673.66216216 3688.97674419  
## Temperature 66.06323024 65.89083536 66.71655629 65.46209150  
## Usage.cont 21.14034247 20.96623482 21.03657895 21.00653659  
## Carb.Flow 2387.35395189 2532.70056497 2311.53947368 2437.18048780  
## Density 1.57105802 0.90843296 0.92217105 1.68250407  
## MFR 704.98365759 705.63864818 704.02826087 703.97628319  
## Balling 3.19896928 1.51114216 1.63171053 3.48558699  
## Pressure.Vacuum -5.23890785 -5.15431800 -5.31052632 -5.26016260  
## PH 8.49740614 8.56678543 8.41368421 8.60250407  
## Oxygen.Filler 0.04147423 0.04783906 0.05160930 0.04495863  
## Bowl.Setpoint 109.11340206 107.76432607 111.61842105 110.88780488  
## Pressure.Setpoint 47.76219931 47.94269572 48.03367003 46.64600326  
## Air.Pressurer 142.73583618 142.95948345 142.76644737 142.69300813  
## Alch.Rel 7.13486301 6.55051095 6.56488449 7.69352846  
## Carb.Rel 5.51773973 5.36215385 5.35242525 5.60725203  
## Balling.Lvl 3.07706485 1.40611784 1.52059211 3.23206504  
## BEM  
## Carb.Volume 5.29086111  
## Fill.Ounces 23.99712644  
## PC.Volume 0.28882759  
## Carb.Pressure 66.26050420  
## Carb.Temp 140.51794872  
## PSC 0.09037607  
## PSC.Fill 0.18931034  
## PSC.CO2 0.05593220  
## Mnf.Flow 27.15333333  
## Carb.Pressure1 123.46050420  
## Fill.Pressure 48.78290598  
## Hyd.Pressure1 10.89000000  
## Hyd.Pressure2 18.43666667  
## Hyd.Pressure3 19.78666667  
## Hyd.Pressure4 95.98275862  
## Filler.Level 112.15084746  
## Filler.Speed 3517.82905983  
## Temperature 67.22689076  
## Usage.cont 20.72933333  
## Carb.Flow 2557.43333333  
## Density 0.96866667  
## MFR 685.10280374  
## Balling 1.67083333  
## Pressure.Vacuum -5.33333333  
## PH 8.48883333  
## Oxygen.Filler 0.04730084  
## Bowl.Setpoint 112.16666667  
## Pressure.Setpoint 47.79831933  
## Air.Pressurer 142.67166667  
## Alch.Rel 6.64151261  
## Carb.Rel 5.34423729  
## Balling.Lvl 1.46873950

Notable differences exist among brands by Hyd.Pressure4, Density, Balling, and Balling.Lvl

### Density Plot

Next we will use density plots to better understand the data and look for any abnormalities.

par(mfrow = c(3,5), cex = .5)  
for (i in colnames(dfBevModH)) {  
 smoothScatter(dfBevModH[,i], main = names(dfBevModH[i]), ylab = "",   
 xlab = "", colramp = colorRampPalette(c("white", "red")))  
 }

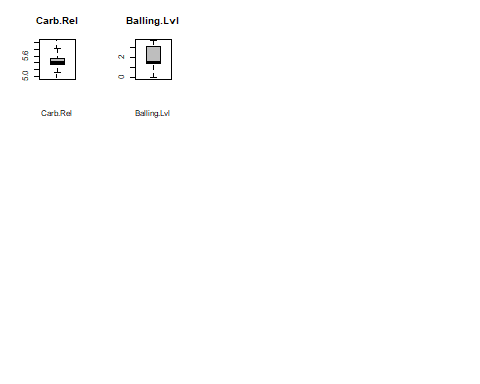
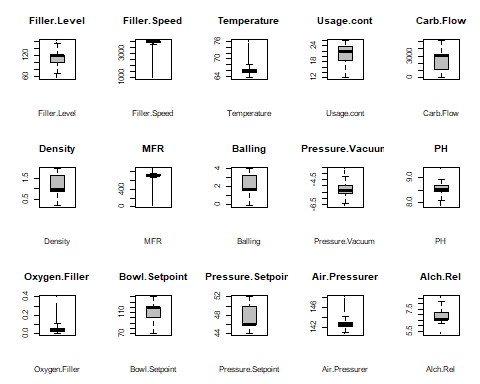
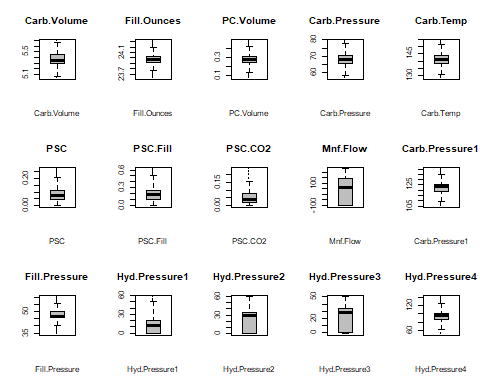


The odd data in Mnf.Flow appears to be related to similarly zero-out data in Hyd.Pressure1, Hyd.Press2, and Hyd.Pressure3. Several other variables have dichotomous patterns in data behavior, including “Carb.Pressure1”, “Filler.Level”, “Usage.cont”, “Carb.Flow”, and “Oxygen.Filler”. This leaves two options; we can alter these gaps by possibly inputing new values in, or we can use algorithms that can easily handle quick pattern shifts, such as forests and MARS.

### BoxPlots

Next we want to take a look at any outliers within our variables. Boxplots provide a quick and effective way of view the data and look for any skew or outliers

par(mfrow = c(3,5), cex = .5)  
for(i in colnames(dfBevModH)){  
boxplot(dfBevModH[,i], xlab = names(dfBevModH[i]),  
 main = names(dfBevModH[i]), col="grey", ylab="")  
}



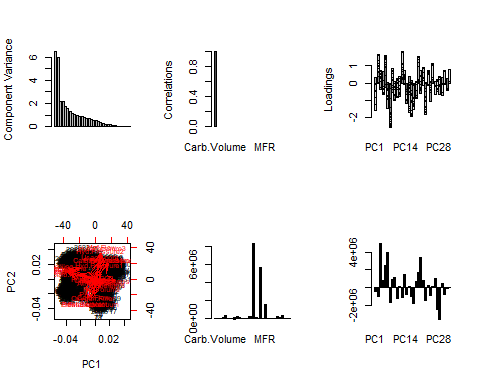
Again, several variables with large skews and outliers are present we will need to use transformation techniques later on to handle this issue.

### Principle Component Analysis

Since we are dealing with 32 predictor variables and not all of them can be relevant to our study, we will need a way to filter through the variables to reduce the number of variables we are working with. This can be done with PCA

PCA <- function(X) {  
 Xpca <- prcomp(na.omit(X), center = T, scale. = T)   
 M <- as.matrix(na.omit(X)); R <- as.matrix(Xpca$rotation); score <- M %\*% R  
 print(list("Importance of Components" = summary(Xpca)$importance[ ,1:5],   
 "Rotation (Variable Loadings)" = Xpca$rotation[ ,1:5],  
 "Correlation between X and PC" = cor(na.omit(X), score)[ ,1:5]))  
 par(mfrow=c(2,3))  
 barplot(Xpca$sdev^2, ylab = "Component Variance")  
 barplot(cor(cbind(X)), ylab = "Correlations")  
 barplot(Xpca$rotation, ylab = "Loadings")   
 biplot(Xpca); barplot(M); barplot(score)  
}  
PCA(dfBevModH)

## $`Importance of Components`  
## PC1 PC2 PC3 PC4 PC5  
## Standard deviation 2.540088 2.449148 1.477634 1.47603 1.310421  
## Proportion of Variance 0.201630 0.187450 0.068230 0.06808 0.053660  
## Cumulative Proportion 0.201630 0.389070 0.457310 0.52539 0.579050  
##   
## $`Rotation (Variable Loadings)`  
## PC1 PC2 PC3 PC4  
## Carb.Volume -0.320871550 0.126525924 -0.063988169 0.0689976465  
## Fill.Ounces 0.033313610 -0.010524814 0.073850141 0.2358285280  
## PC.Volume 0.072479708 -0.108552900 -0.069562774 -0.3956296184  
## Carb.Pressure -0.208006297 0.060068377 -0.068128144 0.1465200114  
## Carb.Temp -0.033305138 -0.006187462 -0.040764654 0.1184540029  
## PSC 0.034729843 -0.007932387 0.007672846 0.0166335686  
## PSC.Fill 0.005088232 -0.025583769 -0.008017467 0.0803836356  
## PSC.CO2 0.029539988 0.013537129 -0.010869845 0.1017169746  
## Mnf.Flow 0.080644367 0.358177152 0.044915507 0.0252965168  
## Carb.Pressure1 0.037910196 0.212302734 0.028684308 0.1765112631  
## Fill.Pressure 0.162738779 0.217724987 -0.208305845 0.0873750286  
## Hyd.Pressure1 0.040714287 0.156202166 0.028117601 -0.4998183585  
## Hyd.Pressure2 0.057453860 0.313403614 0.076107892 -0.3208825682  
## Hyd.Pressure3 0.071682634 0.348252038 0.032135037 -0.2278539437  
## Hyd.Pressure4 0.268513018 -0.023674824 -0.168038357 -0.0027778884  
## Filler.Level -0.098309141 -0.276213319 0.109275148 -0.2476992005  
## Filler.Speed -0.036355308 0.001197480 0.619301867 0.0628275426  
## Temperature 0.105282829 -0.066203240 -0.105105120 0.0321416038  
## Usage.cont 0.053341657 0.248028048 0.106147980 0.1153232541  
## Carb.Flow -0.019747290 -0.192400631 -0.190052445 0.0123151046  
## Density -0.355713158 0.107013359 -0.067887877 -0.0009745302  
## MFR -0.040893045 -0.001083672 0.620594464 0.0666977237  
## Balling -0.354342525 0.141148693 -0.040775757 -0.0141207975  
## Pressure.Vacuum -0.046505652 -0.268137181 0.018936117 0.2043414157  
## PH -0.115017697 -0.166446153 -0.043839565 -0.1546589220  
## Oxygen.Filler -0.017291886 -0.210617922 -0.096839733 -0.0417513911  
## Bowl.Setpoint -0.103923455 -0.273585690 0.086519475 -0.2471199864  
## Pressure.Setpoint 0.181047509 0.208820308 -0.105820469 0.0562644751  
## Air.Pressurer 0.031569404 -0.035033996 -0.071242723 0.2609739989  
## Alch.Rel -0.364650212 0.099430202 -0.076928665 -0.0194895814  
## Carb.Rel -0.352557712 0.079691885 -0.073882285 -0.0418290381  
## Balling.Lvl -0.361905544 0.108856318 -0.069790925 0.0073156499  
## PC5  
## Carb.Volume -0.1289819052  
## Fill.Ounces -0.1229858345  
## PC.Volume 0.1598460185  
## Carb.Pressure 0.5485412373  
## Carb.Temp 0.6879491706  
## PSC -0.0797130932  
## PSC.Fill -0.1062661776  
## PSC.CO2 -0.0212550379  
## Mnf.Flow -0.0159976226  
## Carb.Pressure1 -0.0290714575  
## Fill.Pressure 0.0003314597  
## Hyd.Pressure1 0.0951143049  
## Hyd.Pressure2 0.0947102915  
## Hyd.Pressure3 0.0851266883  
## Hyd.Pressure4 -0.0064600449  
## Filler.Level -0.0659270082  
## Filler.Speed 0.1239047672  
## Temperature 0.1020654842  
## Usage.cont -0.1185660052  
## Carb.Flow 0.1730300450  
## Density -0.0203213313  
## MFR 0.1118598210  
## Balling -0.0496622835  
## Pressure.Vacuum -0.0377417295  
## PH 0.0428191812  
## Oxygen.Filler 0.0913810644  
## Bowl.Setpoint -0.0734787536  
## Pressure.Setpoint 0.0479371223  
## Air.Pressurer 0.0998622820  
## Alch.Rel -0.0372397655  
## Carb.Rel -0.0243017040  
## Balling.Lvl -0.0570128001  
##   
## $`Correlation between X and PC`  
## PC1 PC2 PC3 PC4  
## Carb.Volume 0.002672235 0.111348522 0.063613298 -0.011847085  
## Fill.Ounces 0.040529153 0.087715367 0.097669459 0.059771410  
## PC.Volume -0.180340769 -0.271489773 -0.199880836 -0.115598261  
## Carb.Pressure -0.066204190 0.025722112 0.036530429 0.061413374  
## Carb.Temp -0.069518362 -0.039493236 -0.003344882 0.074340078  
## PSC 0.040662924 0.037176123 0.018514307 0.006021336  
## PSC.Fill -0.021548310 -0.028862504 -0.023591759 0.007699516  
## PSC.CO2 0.035580707 0.022245482 0.003727000 0.020391565  
## Mnf.Flow 0.723291814 0.693451292 0.295235376 -0.147391760  
## Carb.Pressure1 0.375637900 0.371884050 0.168815533 0.006765179  
## Fill.Pressure 0.498230248 0.300422256 -0.075514622 -0.207326749  
## Hyd.Pressure1 0.275073293 0.247417853 0.076691158 -0.437731705  
## Hyd.Pressure2 0.487809573 0.485370725 0.238650794 -0.300196428  
## Hyd.Pressure3 0.580859072 0.545232467 0.225502363 -0.280195784  
## Hyd.Pressure4 0.296714868 0.029386412 -0.206499978 -0.204162315  
## Filler.Level -0.389979312 -0.217909690 0.031229026 -0.134337700  
## Filler.Speed -0.470607794 0.059325780 0.832380594 0.798933852  
## Temperature -0.032400811 -0.131025575 -0.131935567 0.013501453  
## Usage.cont 0.515662927 0.567466858 0.306198713 -0.105542769  
## Carb.Flow -0.728865476 -0.966223188 -0.604417233 0.368469211  
## Density -0.117230400 -0.008474646 0.016244812 0.022657754  
## MFR -0.455045229 0.066334864 0.812095962 0.771973495  
## Balling -0.017953566 0.121197295 0.097470770 -0.026074148  
## Pressure.Vacuum -0.457986670 -0.407097197 -0.119183474 0.240746451  
## PH -0.345239874 -0.323918002 -0.164411012 -0.017197343  
## Oxygen.Filler -0.424027239 -0.467026196 -0.247762941 0.093078019  
## Bowl.Setpoint -0.373730212 -0.214155886 0.007211244 -0.163635722  
## Pressure.Setpoint 0.411830072 0.282182390 0.035998404 -0.095125936  
## Air.Pressurer -0.075890499 -0.120293496 -0.078403739 0.157033933  
## Alch.Rel -0.102425243 0.006988524 0.003604029 -0.018809036  
## Carb.Rel -0.108413466 0.008531971 0.019415972 -0.047949293  
## Balling.Lvl -0.060110506 0.060109493 0.043885246 -0.025858266  
## PC5  
## Carb.Volume -0.093658221  
## Fill.Ounces -0.100673596  
## PC.Volume 0.233542757  
## Carb.Pressure 0.011754537  
## Carb.Temp 0.072779034  
## PSC -0.036486635  
## PSC.Fill 0.008941563  
## PSC.CO2 -0.014123789  
## Mnf.Flow -0.460810554  
## Carb.Pressure1 -0.260300884  
## Fill.Pressure -0.238063463  
## Hyd.Pressure1 -0.154832629  
## Hyd.Pressure2 -0.275314801  
## Hyd.Pressure3 -0.323771761  
## Hyd.Pressure4 -0.090604320  
## Filler.Level 0.028038837  
## Filler.Speed 0.247282208  
## Temperature 0.103770602  
## Usage.cont -0.454543589  
## Carb.Flow 0.950568080  
## Density 0.034322442  
## MFR 0.224469906  
## Balling -0.084814051  
## Pressure.Vacuum 0.277242437  
## PH 0.211975600  
## Oxygen.Filler 0.354347799  
## Bowl.Setpoint 0.012366381  
## Pressure.Setpoint -0.169959804  
## Air.Pressurer 0.125926876  
## Alch.Rel 0.005207278  
## Carb.Rel -0.012596007  
## Balling.Lvl -0.050934875

 First two components account for most of the variance, although Mnf.Flow is highly prioritized, so I’m concerned that it may be a function of the null-like values.

# Data Transformation

Since Brand Code is a categorical variable, we will have to transform it to a binary so that models can use the data.

#df\_imputed$Brand.Code = NULL  
dfBevMod$Brand.Code[dfBevMod$Brand.Code == ""] <- NA  
dfBevMod$Brand.Code <- droplevels(dfBevMod$Brand.Code)  
  
dfBevPred$Brand.Code[dfBevPred$Brand.Code == ""] <- NA  
dfBevPred$Brand.Code <- droplevels(dfBevPred$Brand.Code)  
  
#Recode categorical factor  
dfBevMod$A <- ifelse(dfBevMod$Brand.Code == "A", 1, 0)  
dfBevMod$B <- ifelse(dfBevMod$Brand.Code == "B", 1, 0)  
dfBevMod$C <- ifelse(dfBevMod$Brand.Code == "C", 1, 0)  
dfBevMod$D <- ifelse(dfBevMod$Brand.Code == "D", 1, 0)  
dfBevMod$Brand.Code <- NULL  
  
dfBevPred$A <- ifelse(dfBevPred$Brand.Code == "A", 1, 0)  
dfBevPred$B <- ifelse(dfBevPred$Brand.Code == "B", 1, 0)  
dfBevPred$C <- ifelse(dfBevPred$Brand.Code == "C", 1, 0)  
dfBevPred$D <- ifelse(dfBevPred$Brand.Code == "D", 1, 0)  
dfBevPred$Brand.Code <- NULL

### Imputation

Next we will use the missForest library to impute the missing variable of the predictor variables. The library will use the best method in filling in the missing data.

Please note: We need to perform this step in order to have a complete dataset. Most models will not run on a data set with missing data. This is a common step in data science and filling in the data does not compromise the final results

#dfImpMod = missForest(dfBevMod)  
#dfImpMod$OOBerror #error rate looks good?  
#dfModImp <- dfImpMod$ximp  
  
#dfImpPred <- missForest(dfBevPred)  
#dfPredImp <- dfImpPred$ximp  
#dfPredImp$PH <- NA #redadding PH  
  
#write.csv(dfModImp, "TrainImputeData.csv")  
#write.csv(dfPredImp, "PredictImputeData.csv")  
  
#Stored current imputation results on github to quicken knitr iterations  
dfModImp <- read.csv("https://raw.githubusercontent.com/ChristopheHunt/CUNY-DATA624/master/data/TrainImputeData.csv")  
dfPredImp <- read.csv("https://raw.githubusercontent.com/ChristopheHunt/CUNY-DATA624/master/data/PredictImputeData.csv")

### Preprocessing

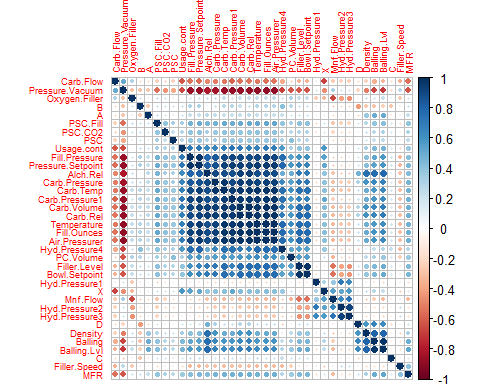
Next we will preprocess the data using various methods, this step will handle the issues of outliers and will get the data in the final stage where it can be used with the predictive models. Due to the different types of model inputs (some preprocess, other’s dont), we will be creating a range of preprocessed variables

dfModImpX <- dfModImp[,!(names(dfModImp) == "PH")]  
dfModImpY <- dfModImp[, names(dfModImp) == "PH"]  
  
dfPredImpX <- dfPredImp[,!(names(dfPredImp) == "PH")]  
dfPredImpY <- dfPredImp[, names(dfPredImp) == "PH"]  
  
#Spatial Sign outlier processing  
dfModImpSsX <- spatialSign(dfModImpX)  
dfPredImpSsX <- spatialSign(dfPredImpX)  
  
#BoxCox Only  
transModB <- preProcess(dfModImpSsX, method = "BoxCox") #transformed all 22 variables  
dfModBX <- predict(transModB, dfModImpSsX)  
  
transPredB <- preProcess(dfPredImpSsX, method = "BoxCox") #transformed 23 variables (should we use the Modeling model from above, instead of predicting model?)  
dfPredBX <- predict(transPredB, dfPredImpSsX)  
  
#BoxCox, Centering, and Scaling  
transModBCS <- preProcess(dfModImpSsX, method = c("BoxCox", "center", "scale")) #22 BC, 35 centered, 35 scaled  
dfModBCSX <- predict(transModBCS, dfModImpSsX)  
  
transPredBCS <- preProcess(dfPredImpSsX, method = c("BoxCox", "center", "scale")) #23 BC, 35 centered, 35 scaled  
dfPredBCSX <- predict(transPredBCS, dfPredImpSsX)  
  
#BoxCox, Centering, Scaling, and PCA  
transModBCSP <- preProcess(dfModImpSsX, method = c("BoxCox", "center", "scale", "pca")) #22 BC, 35 centered, 35 scaled  
dfModBCSPX <- predict(transModBCSP, dfModImpSsX)  
  
transPredBCSP <- preProcess(dfPredImpSsX, method = c("BoxCox", "center", "scale", "pca")) #23 BC, 35 centered, 35 scaled  
dfPredBCSPX <- predict(transPredBCSP, dfPredImpSsX)

Next we will attempt to reduce the number of predictor variables. We will review the correlation between the variables and find the highly correlated ones that can be reduced.

The dark blue and dark red dots indicate a string correlation, normally models do not improve if we feed them highly correlated data, therefore identifying and removing the highly correlated data will help us reduce processing speed and improve accuracy.

#corrgram(dfModBCSX, order=TRUE,  
# upper.panel=panel.cor, main="Correlation Matrix")  
library(corrplot)  
#install.packages("corrplot")  
correlations <- cor(dfModBCSX)  
corrplot(correlations, order = "hclust", tl.cex = 0.55)



We can see several very highly correlated variables. We will reduce our dataset and remove pairs that have correlation above 0.75.

hc = findCorrelation(correlations, cutoff=0.75)  
length(hc) #18 vars

## [1] 16

#Reducing  
dfModBCSRX = dfModBCSX[,-c(hc)] #Box-Cox, Center, Scale  
dfPredBCSRX = dfPredBCSX[,-c(hc)]  
  
dfModBRX = dfModBX[,-c(hc)] #Box-Cox  
dfPredBRX = dfPredBCSX[,-c(hc)]  
  
dfModSRX = dfModImpSsX[,-c(hc)] #Only Spatial Sign  
dfPredSRX = dfPredImpSsX[,-c(hc)]

Finalizing data

set.seed(2017)  
n75 <- floor(0.75 \* nrow(dfBevMod)) #75$ of sample size  
n <- sample(seq\_len(nrow(dfBevMod)), size = n75)  
  
#Box-Cox, Center, Scale  
dfTrainBCSX <- dfModBCSRX[n,]  
dfTestBCSX <- dfModBCSRX[-n,]  
  
#Box-Cox  
dfTrainBX <- dfModBX[n,]  
dfTestBX <- dfModBX[-n,]  
  
#Only Spatial Sign  
dfTrainX <- dfModSRX[n,]  
dfTestX <- dfModSRX[-n,]  
  
#Response variable  
dfTrainY <- dfModImpY[n]  
dfTestY <- dfModImpY[-n]

At this point data is ready and we can proceed to the modeling step.

# Model Development

## Nonlinear Models

Using the several data sets created from our previous transformations we attempted to fit several non-linear models. Specifically, a Neural Network and MARS model. While data transformations are not always necessary for the MARS method, we will nonetheless benefit from removing data features that would be adding unncessary noise to our final models. The Neural Network can be greatly impacted by highly correlated variables.

### Neural Network

Neural Networks can be thought of as models that work in similar ways to our brain. Inputs are provided and transformed at nodes by assigned weights that then feed-forward to any additional layers containing additional nodes [^1]. A drawback to this method is that without limitations on our linear combinations from one layer to another, the coefficients will have little context [^3].

In the below code snippet, we set our seed for reproducibility, then we set trainControl for 3 repeats of the cross validated method and keep our resamples by setting returnResamp = "all". We then manually tune our grid with the expand.grid function and set the Weight Decay via .decay, the Hidden Units via .size, and then prevent Bagging since we have sufficiently preprocessed our data .bag = FALSE. Also, since we are preforming a regression and not a classfication we set linout to TRUE.

[1] Bishop, Christopher M. Neural Networks for Pattern Recognition. Oxford: New York: Clarendon Press; Oxford University Press, 1995.

[3] Kuhn, Max, and Kjell Johnson. Applied Predictive Modeling. New York: Springer, 2013.

set.seed(1234)  
  
nnet <- function(df, y){  
   
 fitControl <- trainControl(method = "cv",   
 number = 3,   
 returnResamp = "all")  
   
 nnetGrid <- expand.grid(.decay = c(0, 0.01, .1, .5),  
 .size = c(5:15),  
 .bag = FALSE)  
   
 return(train(df, y,   
 method = "avNNet",   
 tuneGrid = nnetGrid,  
 trControl = fitControl,  
 trace = FALSE,  
 linout = TRUE))  
}  
  
nnet.fit.bcsx <- nnet(dfTrainBCSX, dfTrainY)  
nnet.fit.bx <- nnet(dfTrainBX, dfTrainY)  
nnet.fit.x <- nnet(dfTrainX, dfTrainY)

Now that we have trained our models on the available data sets, we will measure the root mean squared error of all three and select the lowest.

list(RMSE\_NNET.BCSX = RMSE(predict(nnet.fit.bcsx, dfTestBCSX), dfTestY),  
 RMSE\_NNET.BCS = RMSE(predict(nnet.fit.bx, dfTestBX), dfTestY),  
 RMSE\_NNET.X = RMSE(predict(nnet.fit.x, dfTestX), dfTestY))

## $RMSE\_NNET.BCSX  
## [1] 0.1182677  
##   
## $RMSE\_NNET.BCS  
## [1] 0.1550596  
##   
## $RMSE\_NNET.X  
## [1] 0.1407773

Based on our RMSE results the final model chosen for the neural network is nnet.fit.bcsx

nnet.fit.bcsx$finalModel

## Model Averaged Neural Network with 5 Repeats   
##   
## a 20-11-1 network with 243 weights  
## options were - linear output units

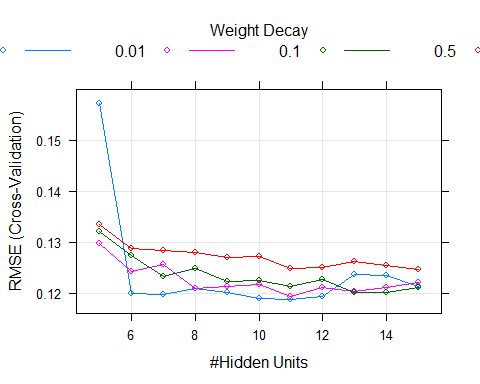
Further, we see that X and Mnf.Flow are the variables with the greatest importance in the model.

varImp(nnet.fit.bcsx)

## loess r-squared variable importance  
##   
## Overall  
## X 100.000  
## Mnf.Flow 86.463  
## Filler.Speed 49.488  
## Filler.Level 42.865  
## Usage.cont 33.177  
## Hyd.Pressure1 24.689  
## Carb.Flow 24.557  
## Hyd.Pressure2 23.417  
## C 22.814  
## Oxygen.Filler 21.224  
## Hyd.Pressure4 16.019  
## Density 15.534  
## B 10.433  
## PC.Volume 5.387  
## PSC 4.496  
## A 1.658  
## PSC.CO2 1.274  
## PSC.Fill 0.976  
## MFR 0.000  
## D 0.000

The grid parameters we set earlier are plotted below to visualize how tuning impacts the model performance.

plot(nnet.fit.bcsx)



### MARS Model

The Multivariate Adaptive Regression Splines or “MARS” model is a nonparametric method, i.e. we are not required to make any assumptions about any underlying distributions such as the neural network. It can achieve this by pivoting on naturally occuring breaks in the data set and essentially building a model out of many linear models developed for specific segemets of the data set [^2] [^3].

[2] J.H. Friedman, “Multivariate adaptive regression splines”, The Annals of Statistics, 19 (1991), pp. 1-141

[3] Kuhn, Max, and Kjell Johnson. Applied Predictive Modeling. New York: Springer, 2013.

In the below code snippet, we set our seed for reproducibility, then we set trainControl for 3 repeats of the cross validated method and keep our resamples by setting returnResamp = "all". We then set our tune grid for 2,3, and 4 product degrees degree= 2:4 and the number of terms possible from 20 to 60 nprune = seq(20,60,20). We would not need to set these values if we intended to use the training set without any resampling or parameter tuning.

library(caret)  
  
set.seed(1234)  
  
mars <- function(df, y){  
   
 fitControl <- trainControl(method = "cv",   
 number = 3,   
 returnResamp = "all")  
  
 MARSGrid <- expand.grid(degree= 3:5,   
 nprune = seq(20,60,20))  
   
 return(train(df, y,   
 method = "earth",   
 tuneGrid = MARSGrid,  
 trControl = fitControl))  
 }  
  
mars.fit.bcsx <- mars(dfTrainBCSX, dfTrainY)  
mars.fit.bx <- mars(dfTrainBX, dfTrainY)  
mars.fit.x <- mars(dfTrainX, dfTrainY)

Now that we have trained our models on the available data sets, we will measure the root mean squared error of all three and select the lowest.

list(RMSE\_MARS.BCSX = RMSE(predict(mars.fit.bcsx, dfTestBCSX), dfTestY),  
 RMSE\_MARS.BCS = RMSE(predict(mars.fit.bx, dfTestBX), dfTestY),  
 RMSE\_MARS.X = RMSE(predict(mars.fit.x, dfTestX), dfTestY))

## $RMSE\_MARS.BCSX  
## [1] 0.1218774  
##   
## $RMSE\_MARS.BCS  
## [1] 0.1131589  
##   
## $RMSE\_MARS.X  
## [1] 0.1260135

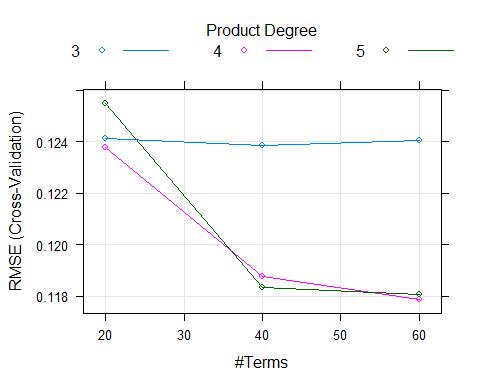
The model fitted to the mars.fit.bx preformed the best and is further selected to be evaluated. The model results are provided below.

mars.fit.bx$finalModel

## Selected 39 of 48 terms, and 15 of 36 predictors  
## Termination condition: Reached maximum RSq 0.9990 at 48 terms  
## Importance: Mnf.Flow, C, X, Temperature, Air.Pressurer, Filler.Speed, ...  
## Number of terms at each degree of interaction: 1 9 16 11 2  
## GCV 0.01211891 RSS 21.09636 GRSq 0.5952478 RSq 0.6341722

The below figure provides insight into the tuning parameters. We can see that our 5 degree model begins to outperform our 4 degree model at the max of 40 terms but the 4 degree model outperforms the 5 degree model at the 60 maximum terms.

plot(mars.fit.bx)



In our final MARS model, the variable of greatest importance is Mnf.Flow.

varImp(mars.fit.bx)

## earth variable importance  
##   
## only 20 most important variables shown (out of 36)  
##   
## Overall  
## Mnf.Flow 100.00  
## C 81.67  
## X 74.73  
## Temperature 63.89  
## Air.Pressurer 63.89  
## Oxygen.Filler 54.50  
## Filler.Speed 52.43  
## Density 52.43  
## D 49.52  
## Carb.Pressure1 45.23  
## Carb.Rel 45.23  
## Alch.Rel 30.39  
## Balling 29.13  
## Pressure.Vacuum 21.28  
## Carb.Volume 11.80  
## Balling.Lvl 0.00  
## Hyd.Pressure2 0.00  
## Filler.Level 0.00  
## PSC 0.00  
## Fill.Pressure 0.00

### Model Choice

Between the two non linear models the lowest RMSE measure is MARS, so we will move forward with that model. It appears that our Neural Net model may suffer from overfitting since the Neural Net has a much better RMSE on the training data set than the MARS model.

list(RMSE\_MARS = RMSE(predict(mars.fit.bx, dfTestBX), dfTestY),  
 RMSE\_NNET = RMSE(predict(nnet.fit.bcsx, dfTestBCSX), dfTestY))

## $RMSE\_MARS  
## [1] 0.1131589  
##   
## $RMSE\_NNET  
## [1] 0.1182677