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))

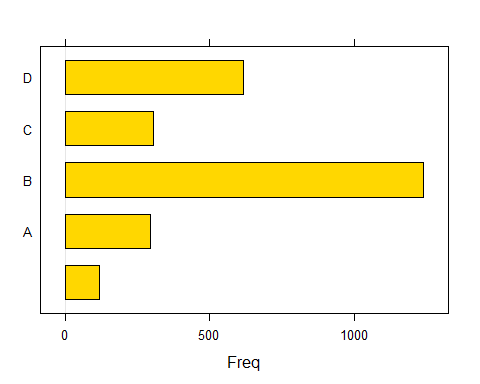
## Brand.Code Carb.Volume Fill.Ounces PC.Volume   
## 0 10 38 39   
## Carb.Pressure Carb.Temp PSC PSC.Fill   
## 27 26 33 23   
## PSC.CO2 Mnf.Flow Carb.Pressure1 Fill.Pressure   
## 39 2 32 22   
## Hyd.Pressure1 Hyd.Pressure2 Hyd.Pressure3 Hyd.Pressure4   
## 11 15 15 30   
## Filler.Level Filler.Speed 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.

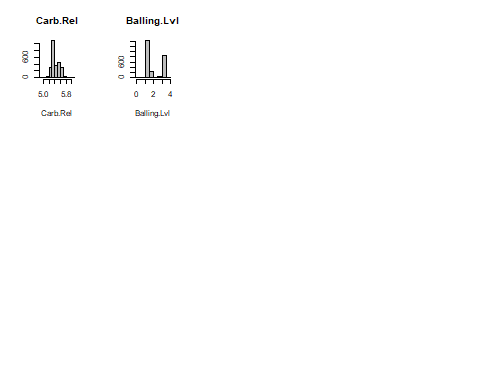
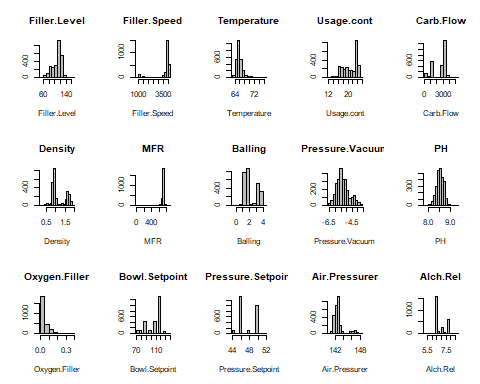
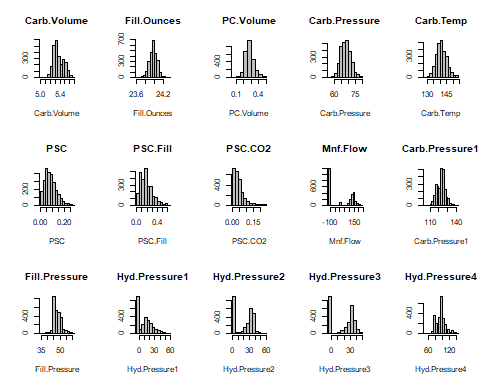
library(psych)  
library(knitr)  
table.desc <- describe(dfBevMod[,-1])  
table.prep <- as.matrix(table.desc)  
table.round <- round((table.prep), 2)  
kable(table.round)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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)  
round(combBrand, 4)

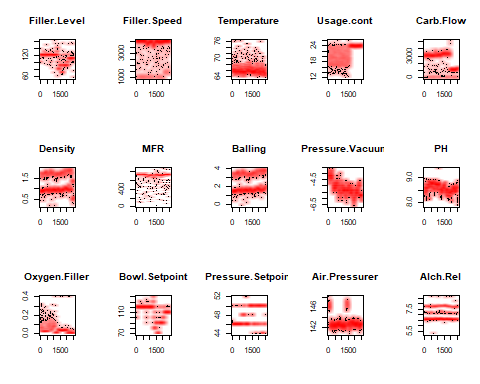
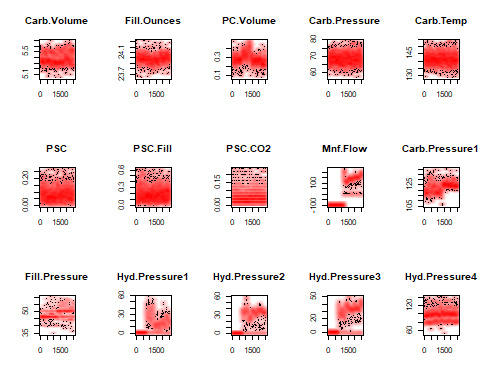
## BAM BBM BCM BDM BEM  
## Carb.Volume 5.4257 5.3125 5.3002 5.5104 5.2909  
## Fill.Ounces 23.9821 23.9775 23.9849 23.9566 23.9971  
## PC.Volume 0.2688 0.2824 0.2902 0.2617 0.2888  
## Carb.Pressure 69.2804 67.1980 66.9248 70.6982 66.2605  
## Carb.Temp 141.2288 141.1108 141.0536 141.1301 140.5179  
## PSC 0.0776 0.0864 0.0898 0.0806 0.0904  
## PSC.Fill 0.1984 0.1937 0.2103 0.1910 0.1893  
## PSC.CO2 0.0555 0.0571 0.0621 0.0528 0.0559  
## Mnf.Flow 39.7147 20.4618 23.2809 25.7467 27.1533  
## Carb.Pressure1 122.8186 122.4750 122.1940 122.7236 123.4605  
## Fill.Pressure 48.2212 48.1701 48.2287 46.9612 48.7829  
## Hyd.Pressure1 12.9024 12.5247 12.3283 12.3967 10.8900  
## Hyd.Pressure2 21.1582 21.1051 19.1625 21.9733 18.4367  
## Hyd.Pressure3 20.9747 20.1925 19.0467 21.5901 19.7867  
## Hyd.Pressure4 101.2837 100.0586 102.5400 83.4528 95.9828  
## Filler.Level 108.6267 107.9616 111.6020 110.4226 112.1508  
## Filler.Speed 3582.0772 3730.6194 3673.6622 3688.9767 3517.8291  
## Temperature 66.0632 65.8908 66.7166 65.4621 67.2269  
## Usage.cont 21.1403 20.9662 21.0366 21.0065 20.7293  
## Carb.Flow 2387.3540 2532.7006 2311.5395 2437.1805 2557.4333  
## Density 1.5711 0.9084 0.9222 1.6825 0.9687  
## MFR 704.9837 705.6386 704.0283 703.9763 685.1028  
## Balling 3.1990 1.5111 1.6317 3.4856 1.6708  
## Pressure.Vacuum -5.2389 -5.1543 -5.3105 -5.2602 -5.3333  
## PH 8.4974 8.5668 8.4137 8.6025 8.4888  
## Oxygen.Filler 0.0415 0.0478 0.0516 0.0450 0.0473  
## Bowl.Setpoint 109.1134 107.7643 111.6184 110.8878 112.1667  
## Pressure.Setpoint 47.7622 47.9427 48.0337 46.6460 47.7983  
## Air.Pressurer 142.7358 142.9595 142.7664 142.6930 142.6717  
## Alch.Rel 7.1349 6.5505 6.5649 7.6935 6.6415  
## Carb.Rel 5.5177 5.3622 5.3524 5.6073 5.3442  
## Balling.Lvl 3.0771 1.4061 1.5206 3.2321 1.4687

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")))  
 }

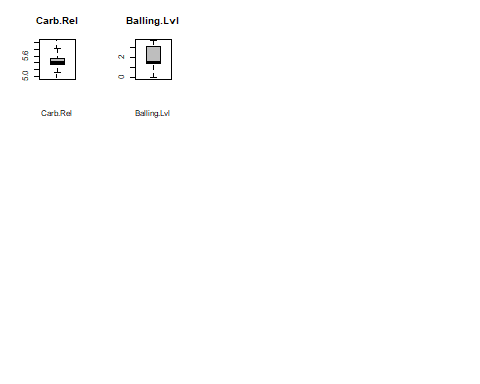
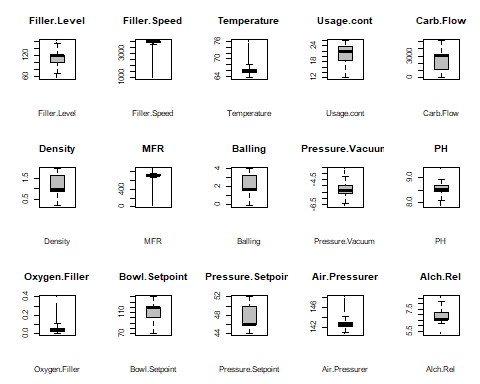
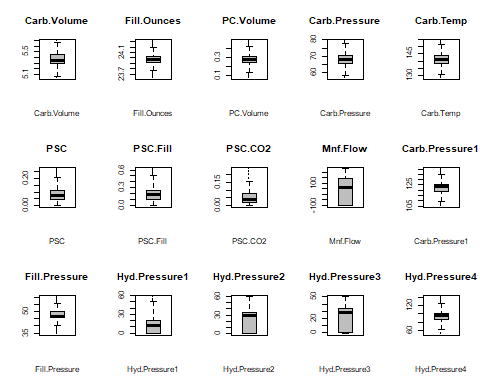


The odd data in Mnf.Flow appears to be related to similarly zero-out data in Hyd.Pressure1, Hyd.Press2, and Hyd.Pressure3. Several other variables have dichotomous patterns in data behavior, including “Carb.Pressure1”, “Filler.Level”, “Usage.cont”, “Carb.Flow”, and “Oxygen.Filler”. This leaves two options; 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="")  
}



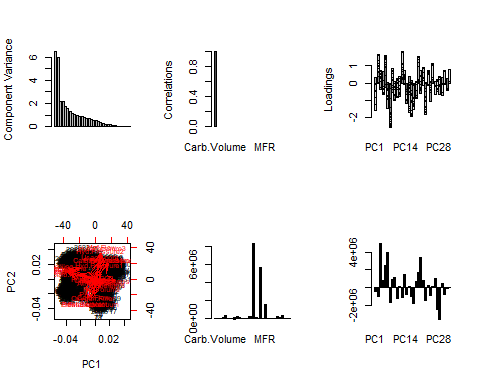
Again, several variables with large skews and outliers are present we will need to use transformation techniques later on to handle this issue.

### 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

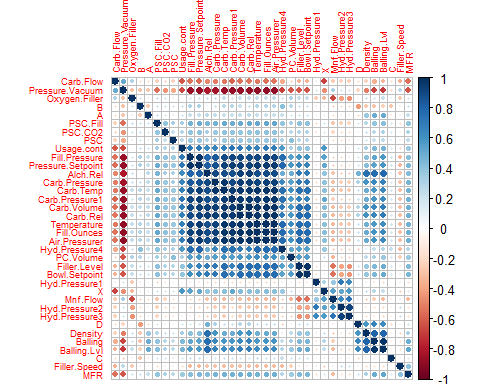
Next we will preprocess the data using various methods, this step will handle the issues of outliers and will get the data in the final stage where it can be used with the predictive models. Due to the different types of model inputs (some preprocess, other’s dont), we will be creating a range of preprocessed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

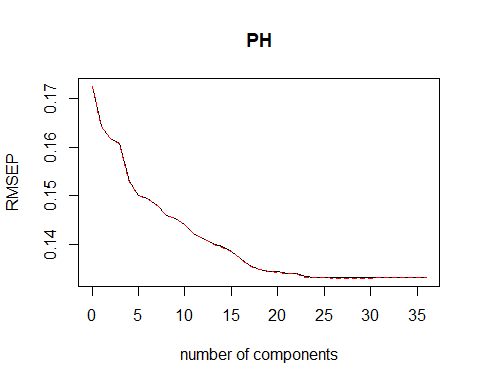
## Partial Least Squares Regression (PLSR)

Partial Least Squares Regression (PLSR) is a multivatiate method related to Principal Components Regression (PCR) that differs because it finds a linear regression that projects predicted and observed values to a new space. These methods are helpful in situations where there are many potentially correlated predictor variables and relatively small samples. In theory, PLSR should be a better method than PCA, because if too few components are selected there could potentially be bad predictions. However, in practice, there doesn’t appear to be much difference in most situations as there are very similar prediction accuracies.

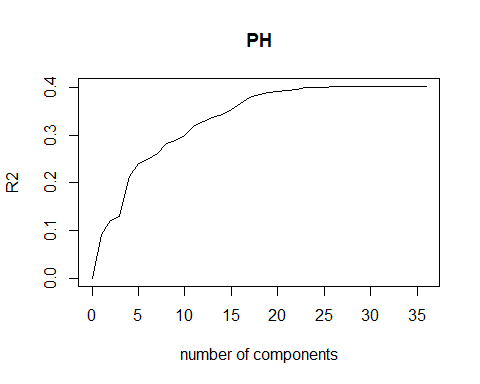
set.seed(1234)  
plsFit = plsr(PH ~ ., data=dfModImp, validation="CV")  
pls.pred = predict(plsFit, dfPredImp[1:5, ], ncomp=1:2)  
  
pls.pred

## , , 1 comps  
##   
## PH  
## 1 8.614191  
## 2 8.613145  
## 3 8.616916  
## 4 8.539746  
## 5 8.621102  
##   
## , , 2 comps  
##   
## PH  
## 1 8.635192  
## 2 8.634208  
## 3 8.635659  
## 4 8.650415  
## 5 8.636982

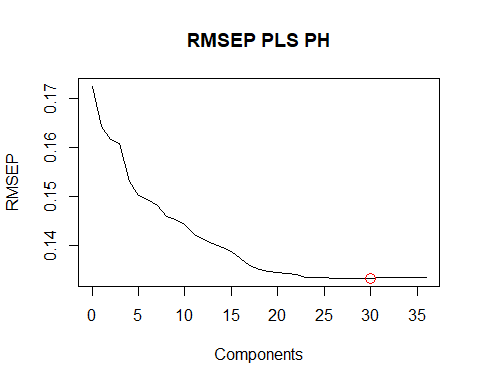
validationplot(plsFit, val.type="RMSEP")



validationplot(plsFit, val.type="R2")



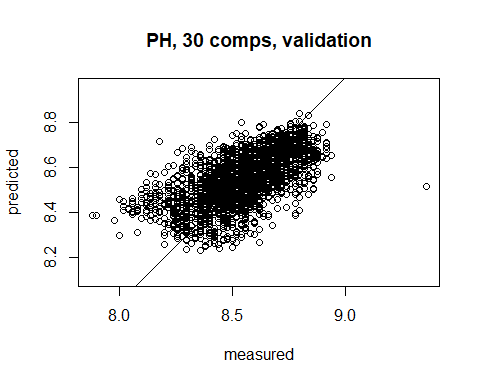
pls.RMSEP = RMSEP(plsFit, estimate="CV")  
plot(pls.RMSEP, main="RMSEP PLS PH", xlab="Components")  
min\_comp = which.min(pls.RMSEP$val)  
points(min\_comp, min(pls.RMSEP$val), pch=1, col="red", cex=1.5)



min\_comp

## [1] 30

plot(plsFit, ncomp=30, asp=1, line=TRUE)



pls.pred2 = predict(plsFit, dfPredImp, ncomp=30)  
pls.pred2

## , , 30 comps  
##   
## PH  
## 1 8.583340  
## 2 8.601578  
## 3 8.605833  
## 4 8.548635  
## 5 8.528368  
## 6 8.628903  
## 7 8.579779  
## 8 8.572863  
## 9 8.665425  
## 10 8.603080  
## 11 8.637534  
## 12 8.606228  
## 13 8.589768  
## 14 8.619004  
## 15 8.408674  
## 16 8.542402  
## 17 8.610035  
## 18 8.591883  
## 19 8.620827  
## 20 8.626382  
## 21 8.583062  
## 22 8.721609  
## 23 8.481919  
## 24 8.674872  
## 25 8.637988  
## 26 8.439427  
## 27 8.528859  
## 28 8.652329  
## 29 8.665838  
## 30 8.697615  
## 31 8.625927  
## 32 8.580866  
## 33 8.652165  
## 34 8.589373  
## 35 8.713613  
## 36 8.562809  
## 37 8.587600  
## 38 8.544160  
## 39 8.647712  
## 40 8.686181  
## 41 8.750200  
## 42 8.642188  
## 43 8.556191  
## 44 8.515569  
## 45 8.604593  
## 46 8.612154  
## 47 8.634431  
## 48 8.622042  
## 49 8.611582  
## 50 8.644532  
## 51 8.600851  
## 52 8.625026  
## 53 8.676077  
## 54 8.622463  
## 55 8.692805  
## 56 8.647025  
## 57 8.421324  
## 58 8.482953  
## 59 8.623701  
## 60 8.656599  
## 61 8.747833  
## 62 8.633775  
## 63 8.711970  
## 64 8.676809  
## 65 8.671551  
## 66 8.571380  
## 67 8.557620  
## 68 8.565365  
## 69 8.615030  
## 70 8.585019  
## 71 8.446092  
## 72 8.470805  
## 73 8.604864  
## 74 8.545763  
## 75 8.564722  
## 76 8.517739  
## 77 8.744208  
## 78 8.687437  
## 79 8.719003  
## 80 8.688091  
## 81 8.778676  
## 82 8.492765  
## 83 8.529838  
## 84 8.522084  
## 85 8.577033  
## 86 8.573122  
## 87 8.708556  
## 88 8.649894  
## 89 8.515728  
## 90 8.684509  
## 91 8.599790  
## 92 8.165542  
## 93 8.511175  
## 94 8.352300  
## 95 8.448409  
## 96 8.628027  
## 97 8.598708  
## 98 8.620877  
## 99 8.536881  
## 100 8.539324  
## 101 8.546716  
## 102 8.686072  
## 103 8.649214  
## 104 8.639911  
## 105 8.681627  
## 106 8.683792  
## 107 8.511794  
## 108 8.506181  
## 109 8.521203  
## 110 8.499428  
## 111 8.636831  
## 112 8.645815  
## 113 8.640391  
## 114 8.726315  
## 115 8.590566  
## 116 8.687818  
## 117 8.652248  
## 118 8.717283  
## 119 8.710723  
## 120 8.676863  
## 121 8.651755  
## 122 8.660012  
## 123 8.731341  
## 124 8.505011  
## 125 8.480895  
## 126 8.470299  
## 127 8.415256  
## 128 8.265968  
## 129 8.457516  
## 130 8.387919  
## 131 8.461432  
## 132 8.460244  
## 133 8.471375  
## 134 8.377787  
## 135 8.393968  
## 136 8.463764  
## 137 8.515683  
## 138 8.481518  
## 139 8.515785  
## 140 8.550155  
## 141 8.415264  
## 142 8.478054  
## 143 8.410519  
## 144 8.352375  
## 145 8.338040  
## 146 8.407509  
## 147 8.363635  
## 148 8.447091  
## 149 8.419357  
## 150 8.422146  
## 151 8.468902  
## 152 8.484013  
## 153 8.417408  
## 154 8.533998  
## 155 8.297211  
## 156 8.340601  
## 157 8.343720  
## 158 8.406906  
## 159 8.460847  
## 160 8.478005  
## 161 8.539701  
## 162 8.504673  
## 163 8.485437  
## 164 8.486665  
## 165 8.436546  
## 166 8.465926  
## 167 8.394476  
## 168 8.541670  
## 169 8.614516  
## 170 8.488418  
## 171 8.425781  
## 172 8.514705  
## 173 8.414915  
## 174 8.458232  
## 175 8.486961  
## 176 8.526935  
## 177 8.432330  
## 178 8.414950  
## 179 8.302110  
## 180 8.383137  
## 181 8.352659  
## 182 8.444637  
## 183 8.375415  
## 184 8.418838  
## 185 8.423220  
## 186 8.302828  
## 187 8.326489  
## 188 8.620683  
## 189 8.421743  
## 190 8.367473  
## 191 8.380718  
## 192 8.477821  
## 193 8.449976  
## 194 8.537251  
## 195 8.449129  
## 196 8.318196  
## 197 8.337999  
## 198 8.361088  
## 199 8.365810  
## 200 8.427305  
## 201 8.461596  
## 202 8.444313  
## 203 8.579866  
## 204 8.278126  
## 205 8.348888  
## 206 8.421156  
## 207 8.411937  
## 208 8.455462  
## 209 8.368689  
## 210 8.361183  
## 211 8.274483  
## 212 8.240345  
## 213 8.238177  
## 214 8.387309  
## 215 8.314191  
## 216 8.315406  
## 217 8.383751  
## 218 8.368191  
## 219 8.439755  
## 220 8.475282  
## 221 8.438573  
## 222 8.457265  
## 223 8.428929  
## 224 8.207657  
## 225 8.449068  
## 226 8.393044  
## 227 8.413349  
## 228 8.425461  
## 229 8.459476  
## 230 8.437051  
## 231 8.470639  
## 232 8.438299  
## 233 8.424943  
## 234 8.453663  
## 235 8.489971  
## 236 8.415962  
## 237 8.473326  
## 238 8.519842  
## 239 8.393520  
## 240 8.383809  
## 241 8.361971  
## 242 8.394161  
## 243 8.350275  
## 244 8.304780  
## 245 8.399470  
## 246 8.431180  
## 247 8.415712  
## 248 8.477259  
## 249 8.504913  
## 250 8.471101  
## 251 8.451834  
## 252 8.442229  
## 253 8.457188  
## 254 8.430043  
## 255 8.520361  
## 256 8.260783  
## 257 8.458854  
## 258 8.499821  
## 259 8.445846  
## 260 8.431023  
## 261 8.469259  
## 262 8.522249  
## 263 8.503038  
## 264 8.484833  
## 265 8.324782  
## 266 8.312529  
## 267 8.331692

With the vast amounts of algorithms available, several should be explored to help make accurate predictions on the dataset. Linear regression is often the simple first step as these models are fast to train albeit with a high bias. The final models are typically easier to analyze and if they are accurate enough it may save moving on to more complex non-linear models. In this example we will start with a simple Generalized Linear Model (GLM) to perform a logistic regression, mix in more complex models then compare the results. A 10-fold cross validation is used for comparison and although not utilized below, a repeated cross validation can be set although it should be noted that repeating a cross valdation with exactly the same splitting will yield exactly the same result for every repetition.

Non-linear algorithms make fewer assumptions about the function being modeled so this will result in higher variance but often result in higher accuracy. Because of its flexibility these models tend to be slower to train and may require increased memory resources. We will explore a Neural Network and MARS model in further detail but will take a quick look at a k-Nearest Neigbor (KNN) and a Support Vector Machine (SVM) algorithm below.

Classification and Regression Trees (CART) and a Bagged CART are also compared below. CART split attributes based on values that minimize a loss function such as RMSE in the below example. Bagging CART is an ensemble method, which we will explore further later, that creates multiple models of the same type from different subsets of the same data. The predictions are then combined together for better results. This approach is particularly useful for high variance methods such as decision trees.

trainControl <- trainControl(method = "cv", number = 10)  
  
#GLM  
set.seed(1234)  
fit.glm <- train(PH~., data=dfModImp, method="glm", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.glm

## Generalized Linear Model   
##   
## 2571 samples  
## 36 predictor  
##   
## Pre-processing: centered (36), scaled (36)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 2314, 2314, 2314, 2312, 2314, 2314, ...   
## Resampling results:  
##   
## RMSE Rsquared MAE   
## 0.1336941 0.4014538 0.1035013

#SVM  
set.seed(1234)  
fit.svm <- train(PH~., data=dfModImp, method="svmLinear3", metric="RMSE", trControl=trainControl, tuneLength = 5, svr\_eps = .1, preProc = c("center", "scale"))  
fit.svm

## L2 Regularized Support Vector Machine (dual) with Linear Kernel   
##   
## 2571 samples  
## 36 predictor  
##   
## Pre-processing: centered (36), scaled (36)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 2314, 2314, 2314, 2312, 2314, 2314, ...   
## Resampling results across tuning parameters:  
##   
## cost Loss RMSE Rsquared MAE   
## 0.25 L1 0.1794758 0.3230042 0.1480714  
## 0.25 L2 0.1347935 0.3912482 0.1043152  
## 0.50 L1 0.1773276 0.3229858 0.1460276  
## 0.50 L2 0.1353488 0.3864330 0.1044418  
## 1.00 L1 0.1762712 0.3229765 0.1450213  
## 1.00 L2 0.1351602 0.3882844 0.1040097  
## 2.00 L1 0.1757476 0.3229719 0.1445198  
## 2.00 L2 0.1367263 0.3791064 0.1055107  
## 4.00 L1 0.1754869 0.3229696 0.1442695  
## 4.00 L2 0.1415372 0.3428813 0.1098640  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were cost = 0.25 and Loss = L2.

#KNN  
set.seed(1234)  
fit.knn <- train(PH~., data=dfModImp, method="knn", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.knn

## k-Nearest Neighbors   
##   
## 2571 samples  
## 36 predictor  
##   
## Pre-processing: centered (36), scaled (36)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 2314, 2314, 2314, 2312, 2314, 2314, ...   
## Resampling results across tuning parameters:  
##   
## k RMSE Rsquared MAE   
## 5 0.1186557 0.5329320 0.08750895  
## 7 0.1177431 0.5391565 0.08733818  
## 9 0.1183880 0.5346838 0.08860023  
## 11 0.1191883 0.5297174 0.08971221  
## 13 0.1199656 0.5239618 0.09067160  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 7.

#CART  
set.seed(1234)  
fit.cart <- train(PH~., data=dfModImp, method="rpart", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.cart

## CART   
##   
## 2571 samples  
## 36 predictor  
##   
## Pre-processing: centered (36), scaled (36)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 2314, 2314, 2314, 2312, 2314, 2314, ...   
## Resampling results across tuning parameters:  
##   
## cp RMSE Rsquared MAE   
## 0.02449866 0.1320985 0.4139347 0.1021212  
## 0.04289798 0.1357536 0.3805193 0.1047916  
## 0.06257969 0.1428339 0.3151390 0.1088900  
## 0.07576762 0.1491084 0.2560344 0.1151092  
## 0.22944632 0.1613498 0.2031147 0.1272420  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was cp = 0.02449866.

#Bagged CART  
set.seed(1234)  
fit.bagcart <- train(PH~., data=dfModImp, method="treebag", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.bagcart

## Bagged CART   
##   
## 2571 samples  
## 36 predictor  
##   
## Pre-processing: centered (36), scaled (36)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 2314, 2314, 2314, 2312, 2314, 2314, ...   
## Resampling results:  
##   
## RMSE Rsquared MAE   
## 0.106051 0.6279657 0.0807397

#Compare results of the algorithms we ran  
results <- resamples(list(GLM=fit.glm, SVM=fit.svm, KNN=fit.knn, CART=fit.cart, BaggedCart=fit.bagcart))  
  
summary(results)

##   
## Call:  
## summary.resamples(object = results)  
##   
## Models: GLM, SVM, KNN, CART, BaggedCart   
## Number of resamples: 10   
##   
## MAE   
## Min. 1st Qu. Median Mean 3rd Qu.  
## GLM 0.09584651 0.10003234 0.10240396 0.10350131 0.10766877  
## SVM 0.09735266 0.10113564 0.10191450 0.10431520 0.10906841  
## KNN 0.08086076 0.08450493 0.08733073 0.08733818 0.09068830  
## CART 0.09253838 0.10015558 0.10212189 0.10212123 0.10514431  
## BaggedCart 0.07234370 0.07802265 0.08148214 0.08073970 0.08403983  
## Max. NA's  
## GLM 0.11002310 0  
## SVM 0.11115424 0  
## KNN 0.09399666 0  
## CART 0.10964481 0  
## BaggedCart 0.08613952 0  
##   
## RMSE   
## Min. 1st Qu. Median Mean 3rd Qu. Max.  
## GLM 0.12464203 0.1283189 0.1315466 0.1336941 0.1397657 0.1436487  
## SVM 0.12641331 0.1292742 0.1323995 0.1347935 0.1422470 0.1438852  
## KNN 0.10633303 0.1141141 0.1151281 0.1177431 0.1222481 0.1305251  
## CART 0.11574383 0.1280689 0.1318011 0.1320985 0.1377113 0.1448673  
## BaggedCart 0.09435403 0.1014102 0.1053985 0.1060510 0.1124663 0.1185575  
## NA's  
## GLM 0  
## SVM 0  
## KNN 0  
## CART 0  
## BaggedCart 0  
##   
## Rsquared   
## Min. 1st Qu. Median Mean 3rd Qu. Max.  
## GLM 0.3404238 0.3899162 0.4014200 0.4014538 0.4206822 0.4609980  
## SVM 0.3223913 0.3764739 0.3979777 0.3912482 0.4152274 0.4518719  
## KNN 0.4803567 0.5015028 0.5415167 0.5391565 0.5649686 0.6199366  
## CART 0.3302931 0.3880648 0.4124382 0.4139347 0.4499189 0.4700435  
## BaggedCart 0.5629121 0.6152075 0.6317394 0.6279657 0.6497779 0.7010340  
## NA's  
## GLM 0  
## SVM 0  
## KNN 0  
## CART 0  
## BaggedCart 0

The Bagged CART performed the best from the above models with a mean RMSE of 0.1060510 and a mean R-Squared of 0.6279657. We will explore further to see if we can find better results.

Ensemble Regression Ensemble methods generally refer to models combining predictions, either in classification or regression, of several base estimators to improve robustness over a single estimator. The goal of ensemble regression is to combine many models in order to increase the prediction accuracy in learning problems with a numerical target. The focus will be on regression not classification with Random Forest and Gradient Boosting (GBM).

Regression trees tend to be unstable, a seemingly insignificant change in the data can have a large impact on the model. The Random Forest can help solve this problem through bagging. Bagging originates from bootstrap aggregating which is a machine learning techique proposed by Breiman to stabalize potentially unstable estimators. Essentially, each variable is given several opportunities to be in the model across multiple bootstrap samples and the final forecast will be the average forecast across all samples.

Gradient Boosting (GBM) is another ensemble method explored that is similarly applied to regression and classification problems. Boosting originated from the notion that weak learners can become better. The first successful boosting algorithm was Adaptive Boosting or AdaBoost. Later, based on this framework, GBM attempted to solve a numerical optimization problem by minimizing the loss of the model by adding weak learners using a gradient descent. These class of algorithms were described as stage-wise additive because a new weak learner is added incrementally, simultaneaously an existing weak learner is left unchanged.

## Regression Trees and Rule-based Models

Random Forest

library(randomForest)  
  
set.seed(1234)  
  
rf <- function(df,y){  
 fitControl <- trainControl(method = "cv",  
 number = 10, #5folds)  
   
rfgrid <- expand.grid(interaction.depth = 2,  
 n.trees = 500,  
 shrinkage = 0.1,  
 n.minobsinnode = 10))  
   
return(train(df, y,   
 method = "randomForest",   
 tuneGrid = rfgrid,   
 trControl = fitControl))  
}  
  
rf.fit.bcsx <- randomForest(dfTrainBCSX, dfTrainY)  
rf.fit.bx <- randomForest(dfTrainBX, dfTrainY)  
rf.fit.x <- randomForest(dfTrainX, dfTrainY)

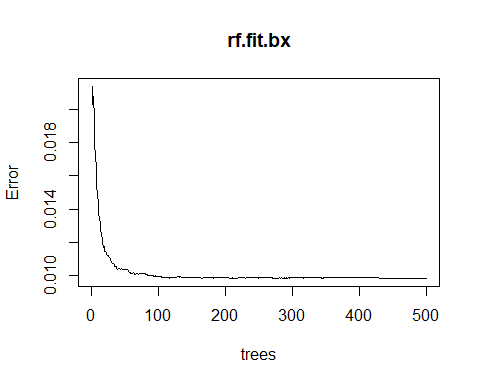
list(RMSE\_RF.BCSX = RMSE(predict(rf.fit.bcsx, dfTestBCSX), dfTestY),  
 RMSE\_RF.BCS = RMSE(predict(rf.fit.bx, dfTestBX), dfTestY),  
 RMSE\_RF.X = RMSE(predict(rf.fit.x, dfTestX), dfTestY))

## $RMSE\_RF.BCSX  
## [1] 0.09756769  
##   
## $RMSE\_RF.BCS  
## [1] 0.09797164  
##   
## $RMSE\_RF.X  
## [1] 0.09772549

summary(rf.fit.bx, digit=3)

## Length Class Mode   
## call 3 -none- call   
## type 1 -none- character  
## predicted 1928 -none- numeric   
## mse 500 -none- numeric   
## rsq 500 -none- numeric   
## oob.times 1928 -none- numeric   
## importance 20 -none- numeric   
## importanceSD 0 -none- NULL   
## localImportance 0 -none- NULL   
## proximity 0 -none- NULL   
## ntree 1 -none- numeric   
## mtry 1 -none- numeric   
## forest 11 -none- list   
## coefs 0 -none- NULL   
## y 1928 -none- numeric   
## test 0 -none- NULL   
## inbag 0 -none- NULL

plot(rf.fit.bx)



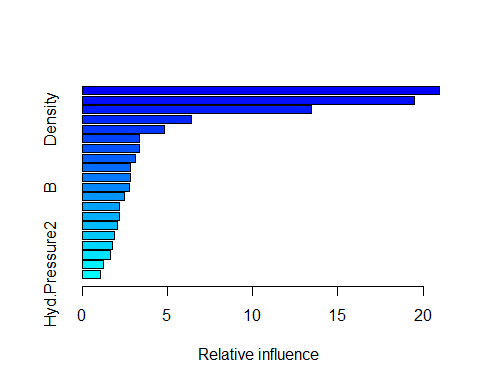
GBM

library(caret)  
  
set.seed(1234)  
  
gbm <- function(df,y){  
 fitControl <- trainControl(method = "cv",  
 number = 10)  
   
gbmgrid <- expand.grid(interaction.depth = 2,  
 n.trees = 500,  
 shrinkage = 0.1,  
 n.minobsinnode = 10)  
   
return(train(df, y,   
 method = "gbm",   
 tuneGrid = gbmgrid,   
 trControl = fitControl,  
 verbose = FALSE))  
}  
  
gbm.fit.bcsx <- gbm(dfTrainBCSX, dfTrainY)  
gbm.fit.bx <- gbm(dfTrainBX, dfTrainY)  
gbm.fit.x <- gbm(dfTrainX, dfTrainY)

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

## $RMSE\_GBM.BCSX  
## [1] 0.1068503  
##   
## $RMSE\_GBM.BCS  
## [1] 0.1088862  
##   
## $RMSE\_GBM.X  
## [1] 0.1085995

summary(gbm.fit.bx, digit=3)



## var rel.inf  
## X X 20.975846  
## Mnf.Flow Mnf.Flow 19.501731  
## C C 13.473770  
## Density Density 6.404560  
## Filler.Speed Filler.Speed 4.801597  
## Filler.Level Filler.Level 3.362863  
## PC.Volume PC.Volume 3.351190  
## Oxygen.Filler Oxygen.Filler 3.096258  
## D D 2.850327  
## A A 2.849153  
## B B 2.754844  
## Usage.cont Usage.cont 2.451267  
## PSC.CO2 PSC.CO2 2.199206  
## MFR MFR 2.178534  
## Carb.Flow Carb.Flow 2.089048  
## Hyd.Pressure1 Hyd.Pressure1 1.915598  
## PSC PSC 1.797518  
## PSC.Fill PSC.Fill 1.650409  
## Hyd.Pressure4 Hyd.Pressure4 1.253464  
## Hyd.Pressure2 Hyd.Pressure2 1.042819

#varImp(gbm.fit.bx)

### Best Performing Regression Trees and Rule-based Models

GBM appears to perform better than the group of models, specifically Bagged CART, with a RMSE of 0.1029799, but we see an improvement with Random Forest, 0.09255377 RMSE.

list(RMSE\_GBM.BCS = RMSE(predict(gbm.fit.bx, dfTestBX), dfTestY),  
 RMSE\_RF.BCS = RMSE(predict(rf.fit.bcsx, dfTestBCSX), dfTestY))

## $RMSE\_GBM.BCS  
## [1] 0.1088862  
##   
## $RMSE\_RF.BCS  
## [1] 0.09756769

## 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] 0.1525009  
##   
## $RMSE\_NNET.X  
## [1] 0.1417475

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

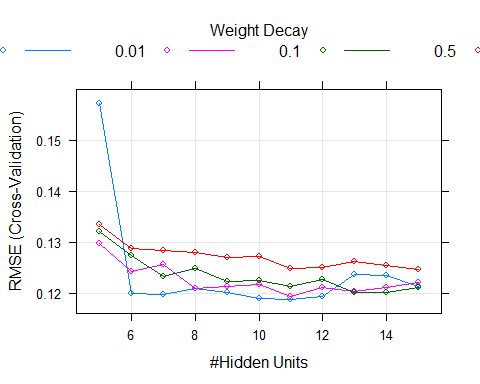
Further, we see that X and Mnf.Flow are the variables with the greatest importance in the model.

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  
## D 0.000  
## MFR 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] 0.1194248  
##   
## $RMSE\_MARS.X  
## [1] 0.1260135

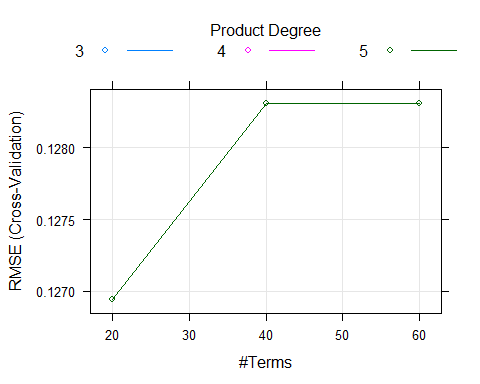
The model fitted to the mars.fit.bx preformed the best and is further selected to be evaluated. The model results are provided below.

mars.fit.bx$finalModel

## Selected 20 of 34 terms, and 9 of 20 predictors  
## Termination condition: Reached nk 41  
## Importance: Mnf.Flow, C, X, D, Density, Filler.Speed, Oxygen.Filler, ...  
## Number of terms at each degree of interaction: 1 4 9 6  
## GCV 0.01401573 RSS 25.6799 GRSq 0.5318973 RSq 0.5546901

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  
##   
## Overall  
## Mnf.Flow 100.00  
## C 79.18  
## X 71.00  
## D 55.92  
## Filler.Speed 52.94  
## Density 52.42  
## Oxygen.Filler 39.24  
## Carb.Flow 36.39  
## PC.Volume 20.39  
## B 0.00  
## PSC.CO2 0.00  
## PSC 0.00  
## Filler.Level 0.00  
## Hyd.Pressure1 0.00  
## A 0.00  
## Hyd.Pressure2 0.00  
## Hyd.Pressure4 0.00  
## Usage.cont 0.00  
## PSC.Fill 0.00  
## MFR 0.00

### Best Performing Nonlinear Model

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] 0.1194248  
##   
## $RMSE\_NNET  
## [1] 0.1182677

# Model Selection

In order to select our most appropriate model we will use the RSME as our selection criteria. While

## Final Model

# Predictions

# Appendix A

## Session Info

R version 3.4.1 (2017-06-30) Platform: x86\_64-w64-mingw32/x64 (64-bit) Running under: Windows 10 x64 (build 15063)

Matrix products: default

locale: [1] LC\_COLLATE=English\_United States.1252 [2] LC\_CTYPE=English\_United States.1252  
[3] LC\_MONETARY=English\_United States.1252 [4] LC\_NUMERIC=C  
[5] LC\_TIME=English\_United States.1252

attached base packages: [1] stats graphics grDevices utils datasets methods base

other attached packages: [1] earth\_4.5.1 plotmo\_3.3.4  
[3] TeachingDemos\_2.10 plotrix\_3.7  
[5] Metrics\_0.1.3 forcats\_0.2.0  
[7] stringr\_1.2.0 dplyr\_0.7.4  
[9] purrr\_0.2.4 readr\_1.1.1  
[11] tidyr\_0.7.2 tibble\_1.3.4  
[13] tidyverse\_1.2.1 e1071\_1.6-8  
[15] rpart\_4.1-11 elasticnet\_1.1  
[17] pls\_2.6-0 lars\_1.2  
[19] AppliedPredictiveModeling\_1.1-6 MASS\_7.3-47  
[21] mlbench\_2.1-1 corrplot\_0.84  
[23] knitr\_1.17 psych\_1.7.8  
[25] caret\_6.0-78 ggplot2\_2.2.1  
[27] lattice\_0.20-35 corrgram\_1.12  
[29] missForest\_1.4 itertools\_0.1-3  
[31] iterators\_1.0.8 foreach\_1.4.3  
[33] randomForest\_4.6-12

loaded via a namespace (and not attached): [1] colorspace\_1.3-2 class\_7.3-14 modeltools\_0.2-21 [4] mclust\_5.4 rprojroot\_1.2 rstudioapi\_0.7  
[7] DRR\_0.0.2 flexmix\_2.3-14 prodlim\_1.6.1  
[10] mvtnorm\_1.0-6 lubridate\_1.7.1 xml2\_1.1.1  
[13] codetools\_0.2-15 splines\_3.4.1 mnormt\_1.5-5  
[16] robustbase\_0.92-8 RcppRoll\_0.2.2 jsonlite\_1.5  
[19] broom\_0.4.3 ddalpha\_1.3.1 cluster\_2.0.6  
[22] kernlab\_0.9-25 sfsmisc\_1.1-1 compiler\_3.4.1  
[25] httr\_1.3.1 backports\_1.1.1 assertthat\_0.2.0  
[28] Matrix\_1.2-12 lazyeval\_0.2.1 cli\_1.0.0  
[31] htmltools\_0.3.6 tools\_3.4.1 bindrcpp\_0.2  
[34] gtable\_0.2.0 glue\_1.2.0 reshape2\_1.4.2  
[37] LiblineaR\_2.10-8 Rcpp\_0.12.14 cellranger\_1.1.0  
[40] trimcluster\_0.1-2 gdata\_2.18.0 nlme\_3.1-131  
[43] fpc\_2.1-10 timeDate\_3042.101 gower\_0.1.2  
[46] rvest\_0.3.2 gtools\_3.5.0 dendextend\_1.6.0  
[49] DEoptimR\_1.0-8 scales\_0.5.0 ipred\_0.9-6  
[52] TSP\_1.1-5 hms\_0.4.0 parallel\_3.4.1  
[55] yaml\_2.1.15 gridExtra\_2.3 stringi\_1.1.6  
[58] highr\_0.6 gclus\_1.3.1 seriation\_1.2-2  
[61] caTools\_1.17.1 lava\_1.5.1 rlang\_0.1.4  
[64] pkgconfig\_2.0.1 prabclus\_2.2-6 bitops\_1.0-6  
[67] evaluate\_0.10.1 bindr\_0.1 recipes\_0.1.1  
[70] CVST\_0.2-1 tidyselect\_0.2.3 gbm\_2.1.3  
[73] plyr\_1.8.4 magrittr\_1.5 CORElearn\_1.51.2  
[76] R6\_2.2.2 gplots\_3.0.1 dimRed\_0.1.0  
[79] haven\_1.1.0 whisker\_0.3-2 foreign\_0.8-69  
[82] withr\_2.1.0 survival\_2.41-3 nnet\_7.3-12  
[85] crayon\_1.3.4 modelr\_0.1.1 KernSmooth\_2.23-15 [88] rmarkdown\_1.8 viridis\_0.4.0 readxl\_1.0.0  
[91] grid\_3.4.1 ModelMetrics\_1.1.0 digest\_0.6.12  
[94] diptest\_0.75-7 stats4\_3.4.1 munsell\_0.4.3  
[97] registry\_0.5 viridisLite\_0.2.0

## All Code Used

knitr::opts\_chunk$set(echo = TRUE, message=FALSE, warning=FALSE, cache=TRUE)  
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)  
dim(dfBevMod)  
colSums(is.na(dfBevMod))  
#Visualizing the single categorical variable  
barchart(dfBevMod[,1], col="Gold")  
library(psych)  
library(knitr)  
table.desc <- describe(dfBevMod[,-1])  
table.prep <- as.matrix(table.desc)  
table.round <- round((table.prep), 2)  
kable(table.round)  
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="")  
}  
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)  
round(combBrand, 4)  
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")))  
 }  
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="")  
}  
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$rotation, ylab = "Loadings")   
 biplot(Xpca); barplot(M); barplot(score)  
}  
PCA(dfBevModH)  
#df\_imputed$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$Brand.Code <- NULL  
#dfImpMod = missForest(dfBevMod)  
#dfImpMod$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")  
  
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)  
#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)  
hc = findCorrelation(correlations, cutoff=0.75)  
length(hc) #18 vars  
  
#Reducing  
dfModBCSRX = dfModBCSX[,-c(hc)] #Box-Cox, Center, Scale  
dfPredBCSRX = dfPredBCSX[,-c(hc)]  
  
dfModBRX = dfModBX[,-c(hc)] #Box-Cox  
dfPredBRX = dfPredBX[,-c(hc)]  
  
dfModSRX = dfModImpSsX[,-c(hc)] #Only Spatial Sign  
dfPredSRX = dfPredImpSsX[,-c(hc)]  
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 <- dfModBRX[n,]  
dfTestBX <- dfModBRX[-n,]  
  
#Only Spatial Sign  
dfTrainX <- dfModSRX[n,]  
dfTestX <- dfModSRX[-n,]  
  
#Response variable  
dfTrainY <- dfModImpY[n]  
dfTestY <- dfModImpY[-n]  
  
knitr::opts\_chunk$set(echo = TRUE)  
library(caret)  
library(mlbench)  
library(MASS)  
library(AppliedPredictiveModeling)  
library(lars)  
library(pls)  
library(elasticnet)  
library(rpart)  
library(e1071)  
set.seed(1234)  
plsFit = plsr(PH ~ ., data=dfModImp, validation="CV")  
pls.pred = predict(plsFit, dfPredImp[1:5, ], ncomp=1:2)  
  
pls.pred  
  
validationplot(plsFit, val.type="RMSEP")  
validationplot(plsFit, val.type="R2")  
  
pls.RMSEP = RMSEP(plsFit, estimate="CV")  
plot(pls.RMSEP, main="RMSEP PLS PH", xlab="Components")  
min\_comp = which.min(pls.RMSEP$val)  
points(min\_comp, min(pls.RMSEP$val), pch=1, col="red", cex=1.5)  
  
min\_comp  
  
plot(plsFit, ncomp=30, asp=1, line=TRUE)  
  
pls.pred2 = predict(plsFit, dfPredImp, ncomp=30)  
pls.pred2  
trainControl <- trainControl(method = "cv", number = 10)  
  
#GLM  
set.seed(1234)  
fit.glm <- train(PH~., data=dfModImp, method="glm", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.glm  
  
#SVM  
set.seed(1234)  
fit.svm <- train(PH~., data=dfModImp, method="svmLinear3", metric="RMSE", trControl=trainControl, tuneLength = 5, svr\_eps = .1, preProc = c("center", "scale"))  
fit.svm  
  
#KNN  
set.seed(1234)  
fit.knn <- train(PH~., data=dfModImp, method="knn", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.knn  
  
#CART  
set.seed(1234)  
fit.cart <- train(PH~., data=dfModImp, method="rpart", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.cart  
  
#Bagged CART  
set.seed(1234)  
fit.bagcart <- train(PH~., data=dfModImp, method="treebag", metric="RMSE", trControl=trainControl,  
 tuneLength = 5, preProc = c("center", "scale"))  
fit.bagcart  
  
#Compare results of the algorithms we ran  
results <- resamples(list(GLM=fit.glm, SVM=fit.svm, KNN=fit.knn, CART=fit.cart, BaggedCart=fit.bagcart))  
  
summary(results)  
  
library(randomForest)  
  
set.seed(1234)  
  
rf <- function(df,y){  
 fitControl <- trainControl(method = "cv",  
 number = 10, #5folds)  
   
rfgrid <- expand.grid(interaction.depth = 2,  
 n.trees = 500,  
 shrinkage = 0.1,  
 n.minobsinnode = 10))  
   
return(train(df, y,   
 method = "randomForest",   
 tuneGrid = rfgrid,   
 trControl = fitControl))  
}  
  
rf.fit.bcsx <- randomForest(dfTrainBCSX, dfTrainY)  
rf.fit.bx <- randomForest(dfTrainBX, dfTrainY)  
rf.fit.x <- randomForest(dfTrainX, dfTrainY)  
list(RMSE\_RF.BCSX = RMSE(predict(rf.fit.bcsx, dfTestBCSX), dfTestY),  
 RMSE\_RF.BCS = RMSE(predict(rf.fit.bx, dfTestBX), dfTestY),  
 RMSE\_RF.X = RMSE(predict(rf.fit.x, dfTestX), dfTestY))  
summary(rf.fit.bx, digit=3)  
plot(rf.fit.bx)  
  
library(caret)  
  
set.seed(1234)  
  
gbm <- function(df,y){  
 fitControl <- trainControl(method = "cv",  
 number = 10)  
   
gbmgrid <- expand.grid(interaction.depth = 2,  
 n.trees = 500,  
 shrinkage = 0.1,  
 n.minobsinnode = 10)  
   
return(train(df, y,   
 method = "gbm",   
 tuneGrid = gbmgrid,   
 trControl = fitControl,  
 verbose = FALSE))  
}  
  
gbm.fit.bcsx <- gbm(dfTrainBCSX, dfTrainY)  
gbm.fit.bx <- gbm(dfTrainBX, dfTrainY)  
gbm.fit.x <- gbm(dfTrainX, dfTrainY)  
list(RMSE\_GBM.BCSX = RMSE(predict(gbm.fit.bcsx, dfTestBCSX), dfTestY),  
 RMSE\_GBM.BCS = RMSE(predict(gbm.fit.bx, dfTestBX), dfTestY),  
 RMSE\_GBM.X = RMSE(predict(gbm.fit.x, dfTestX), dfTestY))  
summary(gbm.fit.bx, digit=3)  
#varImp(gbm.fit.bx)  
list(RMSE\_GBM.BCS = RMSE(predict(gbm.fit.bx, dfTestBX), dfTestY),  
 RMSE\_RF.BCS = RMSE(predict(rf.fit.bcsx, dfTestBCSX), dfTestY))  
library(caret)  
library(tidyverse)  
library(Metrics)  
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)  
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))  
nnet.fit.bcsx$finalModel  
varImp(nnet.fit.bcsx)  
plot(nnet.fit.bcsx)  
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)  
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))  
mars.fit.bx$finalModel  
plot(mars.fit.bx)  
varImp(mars.fit.bx)  
list(RMSE\_MARS = RMSE(predict(mars.fit.bx, dfTestBX), dfTestY),  
 RMSE\_NNET = RMSE(predict(nnet.fit.bcsx, dfTestBCSX), dfTestY))  
sessionInfo()  
## NA