DATA24 Project 2

Forhad Akbar, Shovan Biswas, Elina Azrilyan

11/23/2020

Project 2 Description

This is role playing. I am your new boss. I am in charge of production at ABC Beverage and you are a team of data scientists reporting to me. My leadership has told me that new regulations are requiring us to understand our manufacturing process, the predictive factors and be able to report to them our predictive model of PH.

Please use the historical data set I am providing. Build and report the factors in BOTH a technical and non-technical report. I like to use Word and Excel. Please provide your non-technical report in a business friendly readable document and your predictions in an Excel readable format. The technical report should show clearly the models you tested and how you selected your final approach.

Libraries

```
library(Amelia)
library(AppliedPredictiveModeling)
library(car)
library(caret)
#library(corrgram)
library(corrplot)
library(data.table)
#library(dlookr)
library(dplyr)
library(DT)
library(e1071)
library(forecast)
library(fpp2)
#library(ggcorrplot)
library(ggplot2)
library(glmnet)
#library(glmulti)
library(gridExtra)
#library(Hmisc)
library(kableExtra)
library(knitr)
library(lubridate)
library(MASS)
library(mice)
#library(plotly)
```

```
library(pROC)
#library(pscl)
library(psych)
library(RANN)
#library(RColorBrewer)
library(readxl)
library(reshape2)
library(stringr)
library(tidyverse)
library(tseries)
library(urca)
#library(xlsx)
```

Reading datasets.

```
Ins_train_data <- read.csv("./StudentData.csv", stringsAsFactors = FALSE)
Ins_eval_data <- read.csv("./StudentEvaluation.csv", stringsAsFactors = FALSE)
```

Data Exploration of StudentData.csv.

Initially, we'll do a cursory exploration of the data. After that, we'll iteratively prepare and explore the data, wherever required.

```
dim1 <- dim(Ins_train_data)
print(paste0('Dimension of training set:   ', 'Number of rows: ', dim1[1], ', ', 'Number of cols: ', di
```

```
## [1] "Dimension of training set:   Number of rows: 2571, Number of cols: 33"
```

```
print('Head of training data set:')
```

```
## [1] "Head of training data set:"
```

```
head(Ins_train_data)
```

```
##   Brand.Code Carb.Volume Fill.Ounces PC.Volume Carb.Pressure Carb.Temp   PSC
## 1          B    5.340000    23.96667 0.2633333          68.2     141.2 0.104
## 2          A    5.426667    24.00667 0.2386667          68.4     139.6 0.124
## 3          B    5.286667    24.06000 0.2633333          70.8     144.8 0.090
## 4          A    5.440000    24.00667 0.2933333          63.0     132.6    NA
## 5          A    5.486667    24.31333 0.1113333          67.2     136.8 0.026
## 6          A    5.380000    23.92667 0.2693333          66.6     138.4 0.090
##   PSC.Fill PSC.CO2 Mnf.Flow Carb.Pressure1 Fill.Pressure Hyd.Pressure1
## 1     0.26    0.04     -100          118.8          46.0             0
## 2     0.22    0.04     -100          121.6          46.0             0
## 3     0.34    0.16     -100          120.2          46.0             0
## 4     0.42    0.04     -100          115.2          46.4             0
## 5     0.16    0.12     -100          118.4          45.8             0
## 6     0.24    0.04     -100          119.6          45.6             0
##   Hyd.Pressure2 Hyd.Pressure3 Hyd.Pressure4 Filler.Level Filler.Speed
## 1            NA            NA           118        121.2         4002
## 2            NA            NA           106        118.6         3986
## 3            NA            NA            82        120.0         4020
## 4             0             0            92        117.8         4012
## 5             0             0            92        118.6         4010
## 6             0             0           116        120.2         4014
##   Temperature Usage.cont Carb.Flow Density   MFR Balling Pressure.Vacuum   PH
## 1        66.0      16.18      2932    0.88 725.0   1.398            -4.0 8.36
## 2        67.6      19.90      3144    0.92 726.8   1.498            -4.0 8.26
## 3        67.0      17.76      2914    1.58 735.0   3.142            -3.8 8.94
## 4        65.6      17.42      3062    1.54 730.6   3.042            -4.4 8.24
## 5        65.6      17.68      3054    1.54 722.8   3.042            -4.4 8.26
## 6        66.2      23.82      2948    1.52 738.8   2.992            -4.4 8.32
##   Oxygen.Filler Bowl.Setpoint Pressure.Setpoint Air.Pressurer Alch.Rel Carb.Rel
## 1         0.022           120              46.4         142.6     6.58     5.32
## 2         0.026           120              46.8         143.0     6.56     5.30
## 3         0.024           120              46.6         142.0     7.66     5.84
## 4         0.030           120              46.0         146.2     7.14     5.42
## 5         0.030           120              46.0         146.2     7.14     5.44
## 6         0.024           120              46.0         146.6     7.16     5.44
##   Balling.Lvl
```

```
## 1       1.48
## 2       1.56
## 3       3.28
## 4       3.04
## 5       3.04
## 6       3.02
```

```
print('Structure of training data set:')
```

```
## [1] "Structure of training data set:"
```

```
str(Ins_train_data)
```

```
## 'data.frame':    2571 obs. of  33 variables:
##  $ Brand.Code       : chr  "B" "A" "B" "A" ...
##  $ Carb.Volume      : num  5.34 5.43 5.29 5.44 5.49 ...
##  $ Fill.Ounces      : num  24 24 24.1 24 24.3 ...
##  $ PC.Volume        : num  0.263 0.239 0.263 0.293 0.111 ...
##  $ Carb.Pressure    : num  68.2 68.4 70.8 63 67.2 66.6 64.2 67.6 64.2 72 ...
##  $ Carb.Temp        : num  141 140 145 133 137 ...
##  $ PSC              : num  0.104 0.124 0.09 NA 0.026 0.09 0.128 0.154 0.132 0.014 ...
##  $ PSC.Fill         : num  0.26 0.22 0.34 0.42 0.16 0.24 0.4 0.34 0.12 0.24 ...
##  $ PSC.CO2          : num  0.04 0.04 0.16 0.04 0.12 0.04 0.04 0.04 0.14 0.06 ...
##  $ Mnf.Flow         : num  -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...
##  $ Carb.Pressure1   : num  119 122 120 115 118 ...
##  $ Fill.Pressure    : num  46 46 46 46.4 45.8 45.6 51.8 46.8 46 45.2 ...
##  $ Hyd.Pressure1    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure2    : num  NA NA NA 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure3    : num  NA NA NA 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure4    : int  118 106 82 92 92 116 124 132 90 108 ...
##  $ Filler.Level     : num  121 119 120 118 119 ...
##  $ Filler.Speed     : int  4002 3986 4020 4012 4010 4014 NA 1004 4014 4028 ...
##  $ Temperature      : num  66 67.6 67 65.6 65.6 66.2 65.8 65.2 65.4 66.6 ...
##  $ Usage.cont       : num  16.2 19.9 17.8 17.4 17.7 ...
##  $ Carb.Flow        : int  2932 3144 2914 3062 3054 2948 30 684 2902 3038 ...
##  $ Density          : num  0.88 0.92 1.58 1.54 1.54 1.52 0.84 0.84 0.9 0.9 ...
##  $ MFR              : num  725 727 735 731 723 ...
##  $ Balling          : num  1.4 1.5 3.14 3.04 3.04 ...
##  $ Pressure.Vacuum  : num  -4 -4 -3.8 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 ...
##  $ PH               : num  8.36 8.26 8.94 8.24 8.26 8.32 8.4 8.38 8.38 8.5 ...
##  $ Oxygen.Filler    : num  0.022 0.026 0.024 0.03 0.03 0.024 0.066 0.046 0.064 0.022 ...
##  $ Bowl.Setpoint    : int  120 120 120 120 120 120 120 120 120 120 ...
##  $ Pressure.Setpoint: num  46.4 46.8 46.6 46 46 46 46 46 46 46 ...
##  $ Air.Pressurer    : num  143 143 142 146 146 ...
##  $ Alch.Rel         : num  6.58 6.56 7.66 7.14 7.14 7.16 6.54 6.52 6.52 6.54 ...
##  $ Carb.Rel         : num  5.32 5.3 5.84 5.42 5.44 5.44 5.38 5.34 5.34 5.34 ...
##  $ Balling.Lvl      : num  1.48 1.56 3.28 3.04 3.04 3.02 1.44 1.44 1.44 1.38 ...
```

There are few fields, which have missing values, which we'll investigate in greater details later.

Data Preparation of StudentData.csv.

At this stage, we'll explore and prepare iteratively. Now, we'll check for NA. After that if required, we'll impute them. After that we'll show some boxplots of the numeric fields.

Checking for NA.

```
any(is.na(Ins_train_data))
```

```
## [1] TRUE
```

NA does exist. So, we'll impute with mice().

```
Ins_train_imputed <- mice(Ins_train_data, m = 1, method = "pmm", print = F) %>% mice::complete()
```

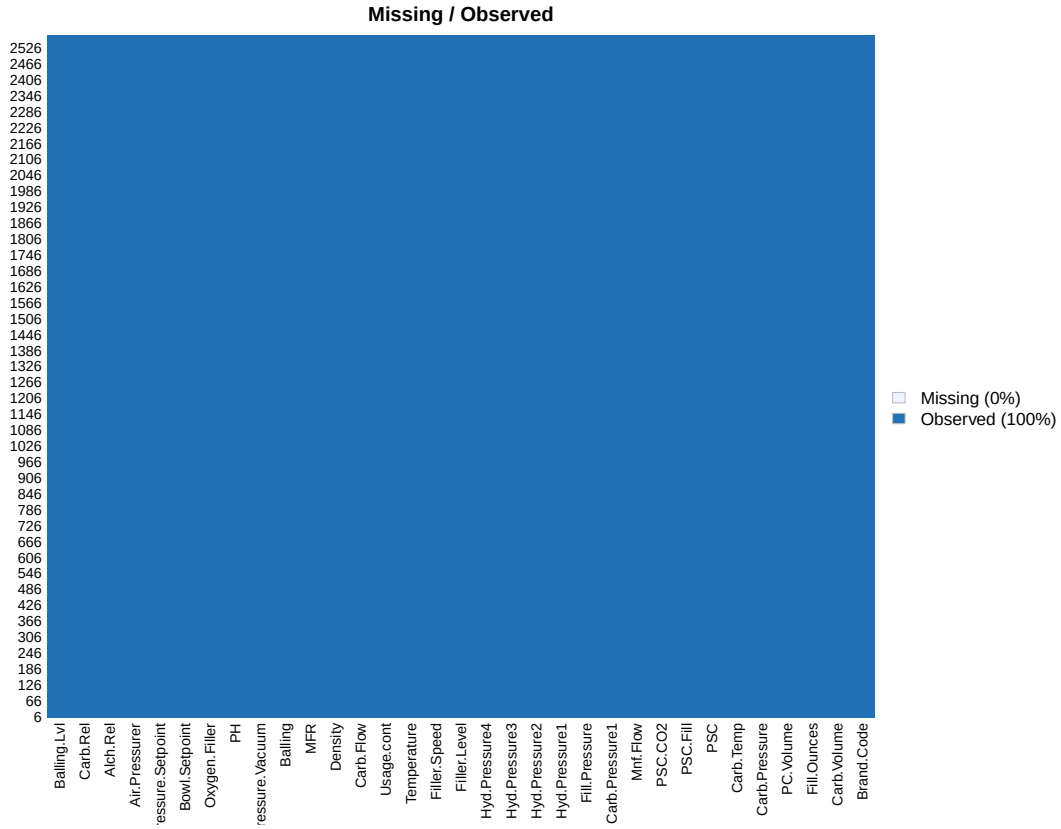
Rechecking for NA after imputation.

```
any(is.na(Ins_train_imputed))
```

```
## [1] FALSE
```

We observe that NA were removed. In the following, we'll visualize with missmap().

```
Ins_train_imputed %>% missmap(main = "Missing / Observed")
```



Both is.na() and missmap() confirm that NA were eliminated.

More Data Exploration of StudentData.csv.

Here, we'll explore the data a little further. First, we'll take a quick look at min, 1st quartile, median, mean, 2nd quartile, max etc.

```
summary(Ins_train_imputed) # %>% kable
```

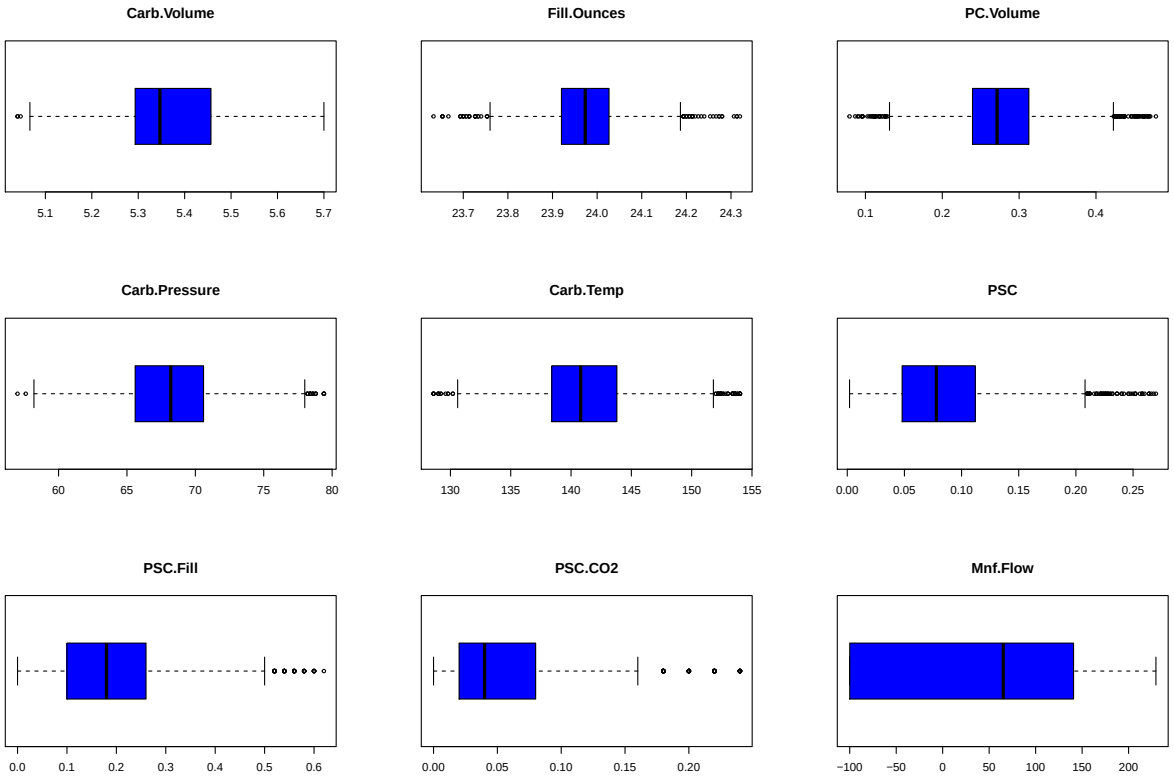
```
##   Brand.Code          Carb.Volume     Fill.Ounces      PC.Volume
##  Length:2571        Min.   :5.040   Min.   :23.63   Min.   :0.07933
##  Class :character   1st Qu.:5.293   1st Qu.:23.92   1st Qu.:0.23933
##  Mode  :character   Median :5.347   Median :23.97   Median :0.27133
##                     Mean   :5.370   Mean   :23.97   Mean   :0.27768
##                     3rd Qu.:5.457   3rd Qu.:24.03   3rd Qu.:0.31267
##                     Max.   :5.700   Max.   :24.32   Max.   :0.47800
##  Carb.Pressure     Carb.Temp           PSC             PSC.Fill
##  Min.   :57.00   Min.   :128.6   Min.   :0.00200   Min.   :0.0000
##  1st Qu.:65.60   1st Qu.:138.4   1st Qu.:0.04800   1st Qu.:0.1000
##  Median :68.20   Median :140.8   Median :0.07800   Median :0.1800
##  Mean   :68.21   Mean   :141.1   Mean   :0.08503   Mean   :0.1958
##  3rd Qu.:70.60   3rd Qu.:143.8   3rd Qu.:0.11200   3rd Qu.:0.2600
##  Max.   :79.40   Max.   :154.0   Max.   :0.27000   Max.   :0.6200
##     PSC.CO2          Mnf.Flow       Carb.Pressure1  Fill.Pressure
##  Min.   :0.0000   Min.   :-100.20   Min.   :105.6   Min.   :34.60
##  1st Qu.:0.0200   1st Qu.:-100.00   1st Qu.:119.0   1st Qu.:46.00
##  Median :0.0400   Median :  65.20   Median :123.2   Median :46.40
##  Mean   :0.0563   Mean   :  24.57   Mean   :122.6   Mean   :47.92
##  3rd Qu.:0.0800   3rd Qu.: 140.80   3rd Qu.:125.4   3rd Qu.:50.00
##  Max.   :0.2400   Max.   : 229.40   Max.   :140.2   Max.   :60.40
##  Hyd.Pressure1   Hyd.Pressure2   Hyd.Pressure3   Hyd.Pressure4
##  Min.   :-0.80   Min.   : 0.00   Min.   :-1.20   Min.   : 52.00
##  1st Qu.: 0.00   1st Qu.: 0.00   1st Qu.: 0.00   1st Qu.: 86.00
##  Median :11.40   Median :28.60   Median :27.40   Median : 96.00
##  Mean   :12.45   Mean   :20.95   Mean   :20.43   Mean   : 96.55
##  3rd Qu.:20.20   3rd Qu.:34.60   3rd Qu.:33.20   3rd Qu.:102.00
##  Max.   :58.00   Max.   :59.40   Max.   :50.00   Max.   :142.00
##   Filler.Level    Filler.Speed   Temperature      Usage.cont      Carb.Flow
##  Min.   : 55.8   Min.   : 998   Min.   :63.60   Min.   :12.08   Min.   :  26
##  1st Qu.: 98.3   1st Qu.:3815   1st Qu.:65.20   1st Qu.:18.36   1st Qu.:1151
##  Median :118.4   Median :3980   Median :65.60   Median :21.80   Median :3028
##  Mean   :109.2   Mean   :3637   Mean   :65.98   Mean   :20.99   Mean   :2469
##  3rd Qu.:120.0   3rd Qu.:3996   3rd Qu.:66.40   3rd Qu.:23.75   3rd Qu.:3187
##  Max.   :161.2   Max.   :4030   Max.   :76.20   Max.   :25.90   Max.   :5104
##     Density           MFR           Balling       Pressure.Vacuum
##  Min.   :0.240   Min.   : 31.4   Min.   :-0.170   Min.   :-6.600
##  1st Qu.:0.900   1st Qu.:694.9   1st Qu.: 1.496   1st Qu.:-5.600
##  Median :0.980   Median :721.4   Median : 1.648   Median :-5.400
##  Mean   :1.174   Mean   :670.6   Mean   : 2.198   Mean   :-5.216
##  3rd Qu.:1.620   3rd Qu.:730.4   3rd Qu.: 3.292   3rd Qu.:-5.000
##  Max.   :1.920   Max.   :868.6   Max.   : 4.012   Max.   :-3.600
##        PH        Oxygen.Filler     Bowl.Setpoint   Pressure.Setpoint
##  Min.   :7.880   Min.   :0.00240   Min.   : 70.0   Min.   :44.00
##  1st Qu.:8.440   1st Qu.:0.02200   1st Qu.:100.0   1st Qu.:46.00
##  Median :8.540   Median :0.03340   Median :120.0   Median :46.00
```

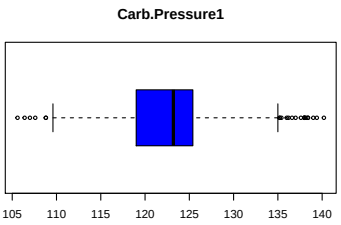
```
##  Mean   :8.546   Mean   :0.04706   Mean   :109.3   Mean   :47.61
##  3rd Qu.:8.680   3rd Qu.:0.06000   3rd Qu.:120.0   3rd Qu.:50.00
##  Max.   :9.360   Max.   :0.40000   Max.   :140.0   Max.   :52.00
##  Air.Pressurer      Alch.Rel        Carb.Rel      Balling.Lvl
##  Min.   :140.8   Min.   :5.280   Min.   :4.960   Min.   :0.00
##  1st Qu.:142.2   1st Qu.:6.540   1st Qu.:5.340   1st Qu.:1.38
##  Median :142.6   Median :6.560   Median :5.400   Median :1.48
##  Mean   :142.8   Mean   :6.897   Mean   :5.436   Mean   :2.05
##  3rd Qu.:143.0   3rd Qu.:7.230   3rd Qu.:5.540   3rd Qu.:3.14
##  Max.   :148.2   Max.   :8.620   Max.   :6.060   Max.   :3.66
```

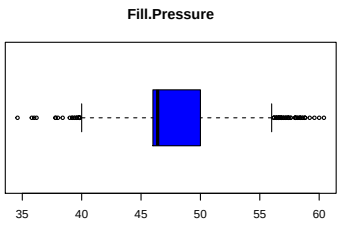
Boxplots

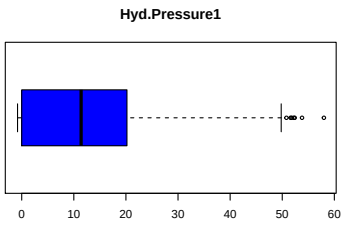
First look at the boxplots.

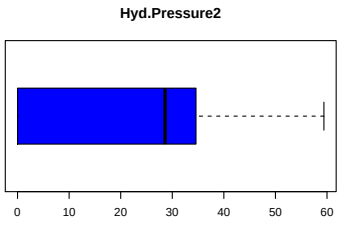
```
par(mfrow = c(3, 3))
for(i in 2:33) {
    if (is.numeric(Ins_train_imputed[,i])) {
      boxplot(Ins_train_imputed[,i], main = names(Ins_train_imputed[i]), col = 4, horizontal = TRUE)
   }
}
```

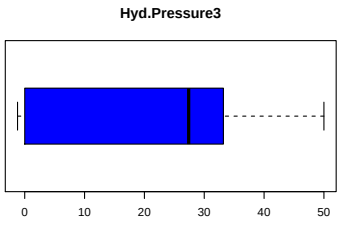


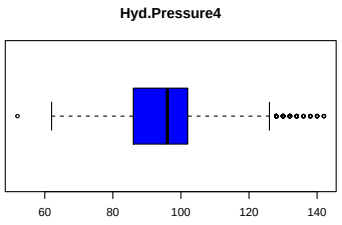

Carb.Pressure1
105 110 115 120 125 130 135 140

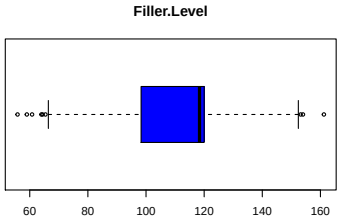

Fill.Pressure
35 40 45 50 55 60

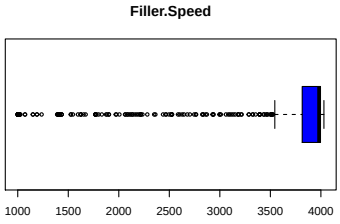

Hyd.Pressure1
0 10 20 30 40 50 60

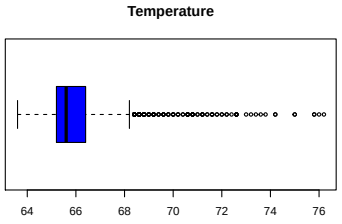

Hyd.Pressure2
0 10 20 30 40 50 60

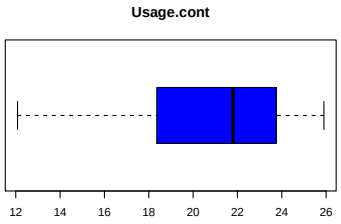

Hyd.Pressure3
0 10 20 30 40 50

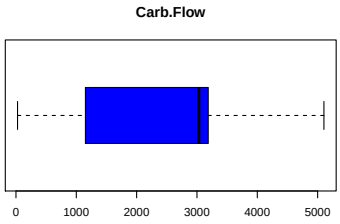

Hyd.Pressure4
60 80 100 120 140

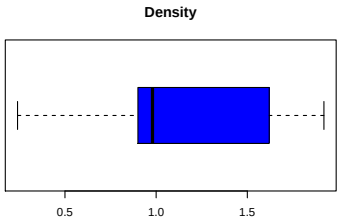

Filler.Level
60 80 100 120 140 160

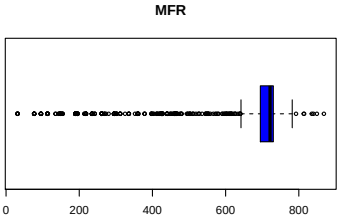

Filler.Speed
1000 1500 2000 2500 3000 3500 4000

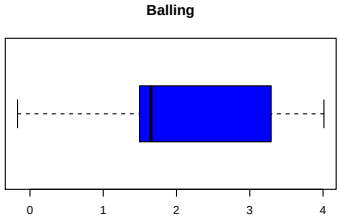

Temperature
64 66 68 70 72 74 76

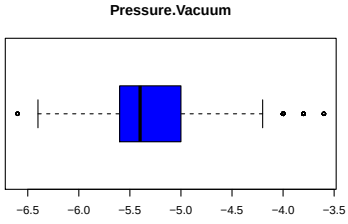

Usage.cont
12 14 16 18 20 22 24 26

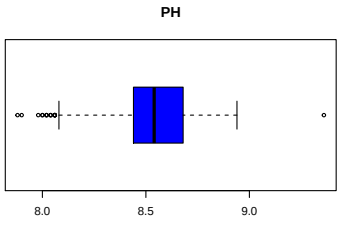

Carb.Flow
0 1000 2000 3000 4000 5000

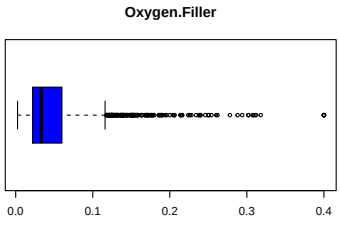

Density
0.5 1.0 1.5

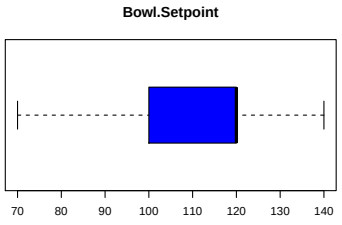

MFR
0 200 400 600 800

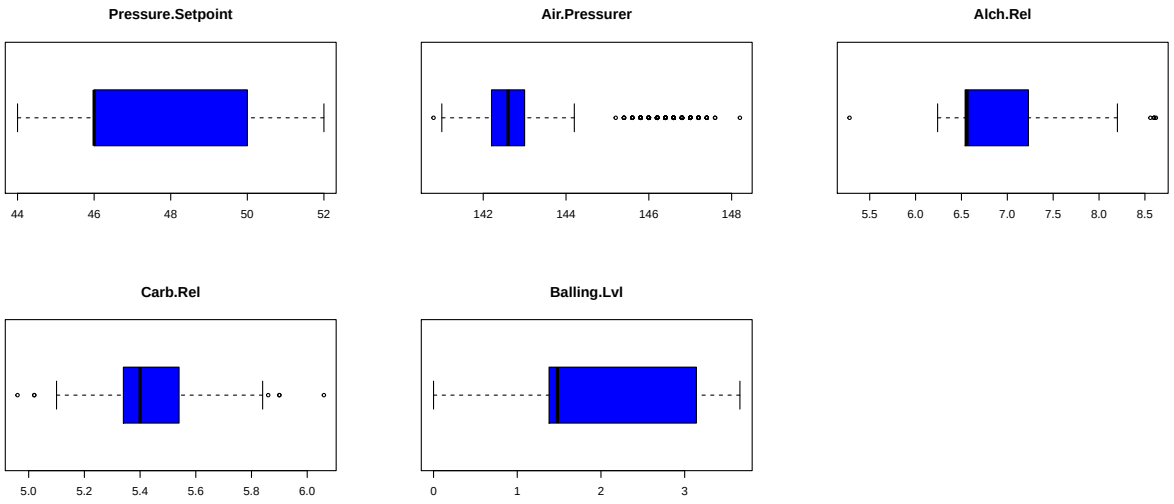

Balling
0 1 2 3 4


Pressure.Vacuum
−6.5 −6.0 −5.5 −5.0 −4.5 −4.0 −3.5


PH
8.0 8.5 9.0


Oxygen.Filler
0.0 0.1 0.2 0.3 0.4

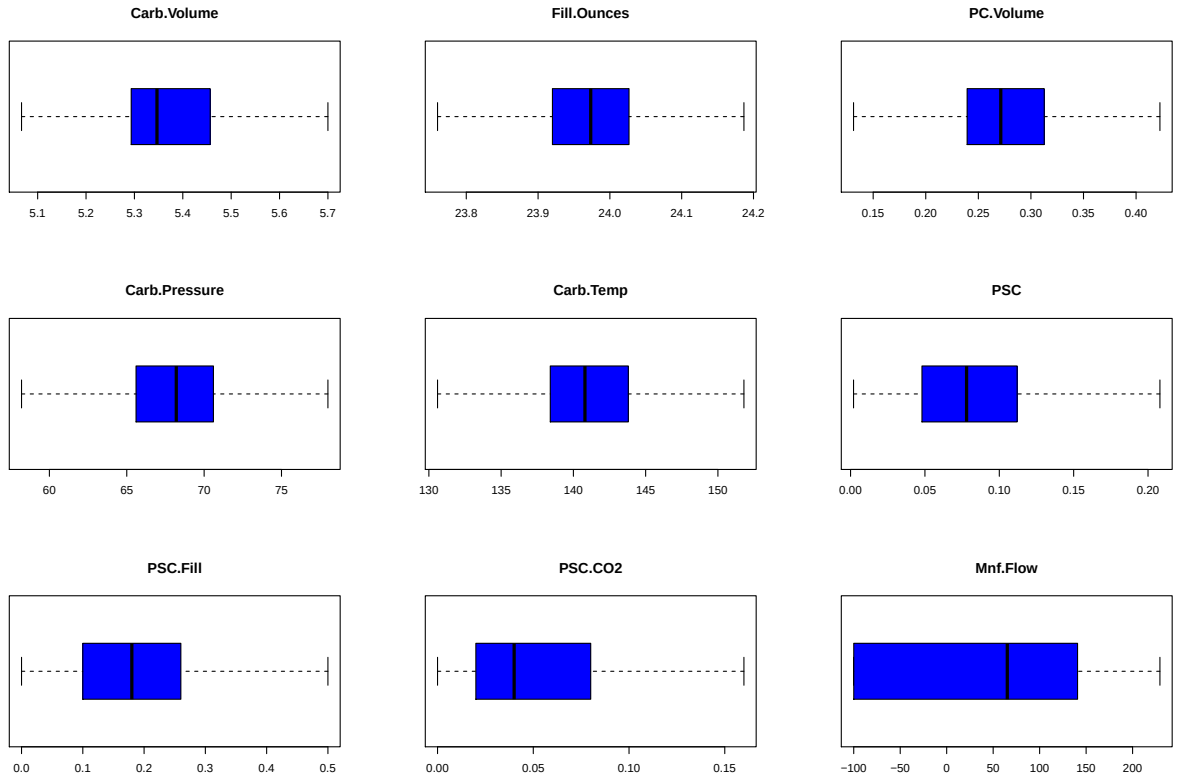

Bowl.Setpoint
70 80 90 100 110 120 130 140

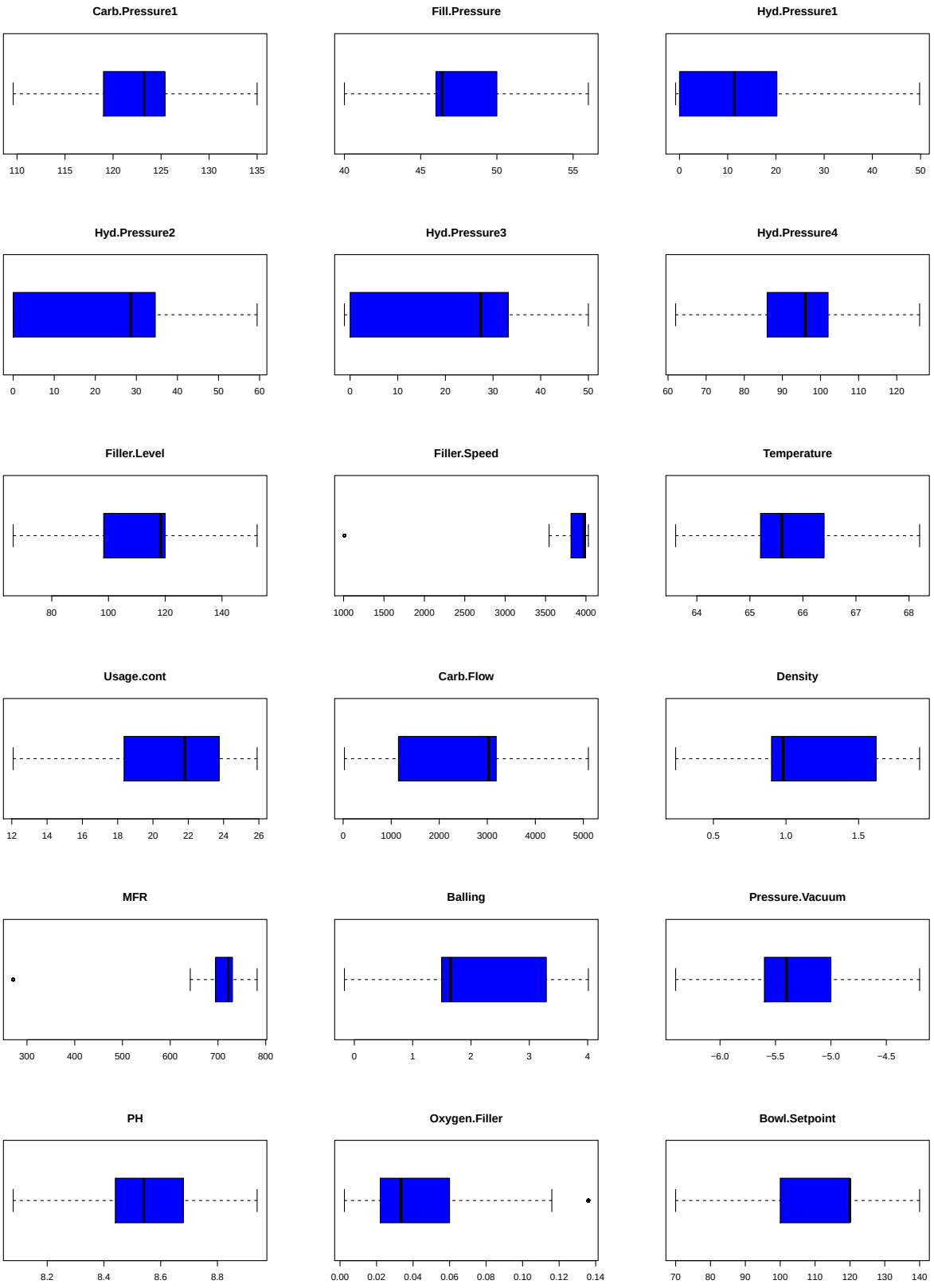

Pressure.Setpoint
44
46
48
50
52
Air.Pressurer
142
144
146
148
Alch.Rel
5.5
6.0
6.5
7.0
7.5
8.0
8.5
Carb.Rel
5.0
5.2
5.4
5.6
5.8
6.0
Balling.Lvl
0
1
2
3

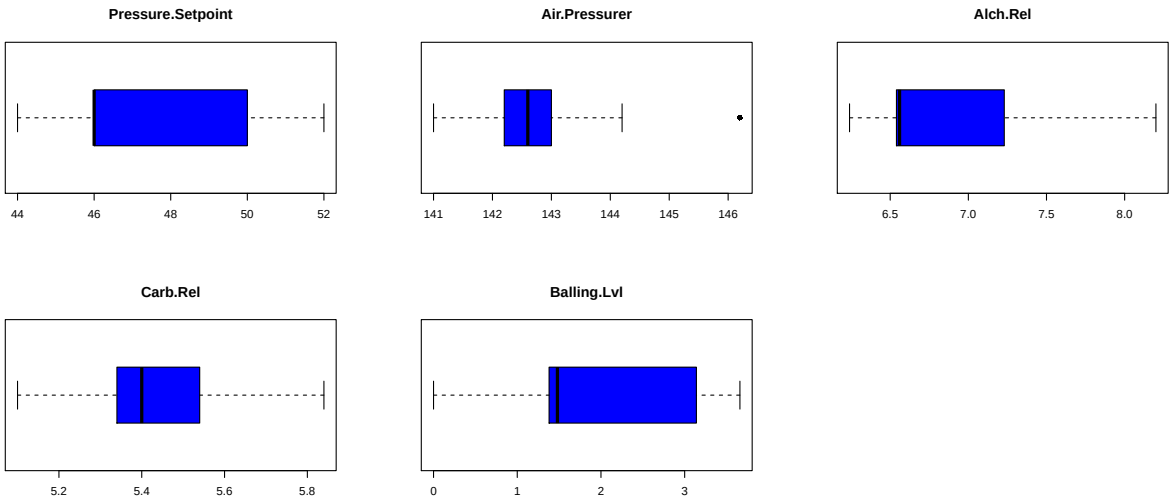
The boxplots show that some of the variables have outliers in them. So, we'll cap them.

```
Ins_train_cap <- Ins_train_imputed

for (i in 2:33) {
  qntl <- quantile(Ins_train_cap[,i], probs = c(0.25, 0.75), na.rm = T)
  cap_amt <- quantile(Ins_train_cap[,i], probs = c(0.05, 0.95), na.rm = T)
  High <- 1.5 * IQR(Ins_train_cap[,i], na.rm = T)

  Ins_train_cap[,i][Ins_train_cap[,i] < (qntl[1] - High)] <- cap_amt[1]
  Ins_train_cap[,i][Ins_train_cap[,i] > (qntl[2] + High)] <- cap_amt[2]
}
```

```
par(mfrow = c(3, 3))
for(i in 2:33) {
    if (is.numeric(Ins_train_cap[,i])) {
      boxplot(Ins_train_cap[,i], main = names(Ins_train_cap[i]), col = 4, horizontal = TRUE)
   }
}
```




Carb.Pressure1
110 115 120 125 130 135
Fill.Pressure
40 45 50 55
Hyd.Pressure1
0 10 20 30 40 50
Hyd.Pressure2
0 10 20 30 40 50 60
Hyd.Pressure3
0 10 20 30 40 50
Hyd.Pressure4
60 70 80 90 100 110 120
Filler.Level
80 100 120 140
Filler.Speed
1000 1500 2000 2500 3000 3500 4000
Temperature
64 65 66 67 68
Usage.cont
12 14 16 18 20 22 24 26
Carb.Flow
0 1000 2000 3000 4000 5000
Density
0.5 1.0 1.5
MFR
300 400 500 600 700 800
Balling
0 1 2 3 4
Pressure.Vacuum
−6.0 −5.5 −5.0 −4.5
PH
8.2 8.4 8.6 8.8
Oxygen.Filler
0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14
Bowl.Setpoint
70 80 90 100 110 120 130 140

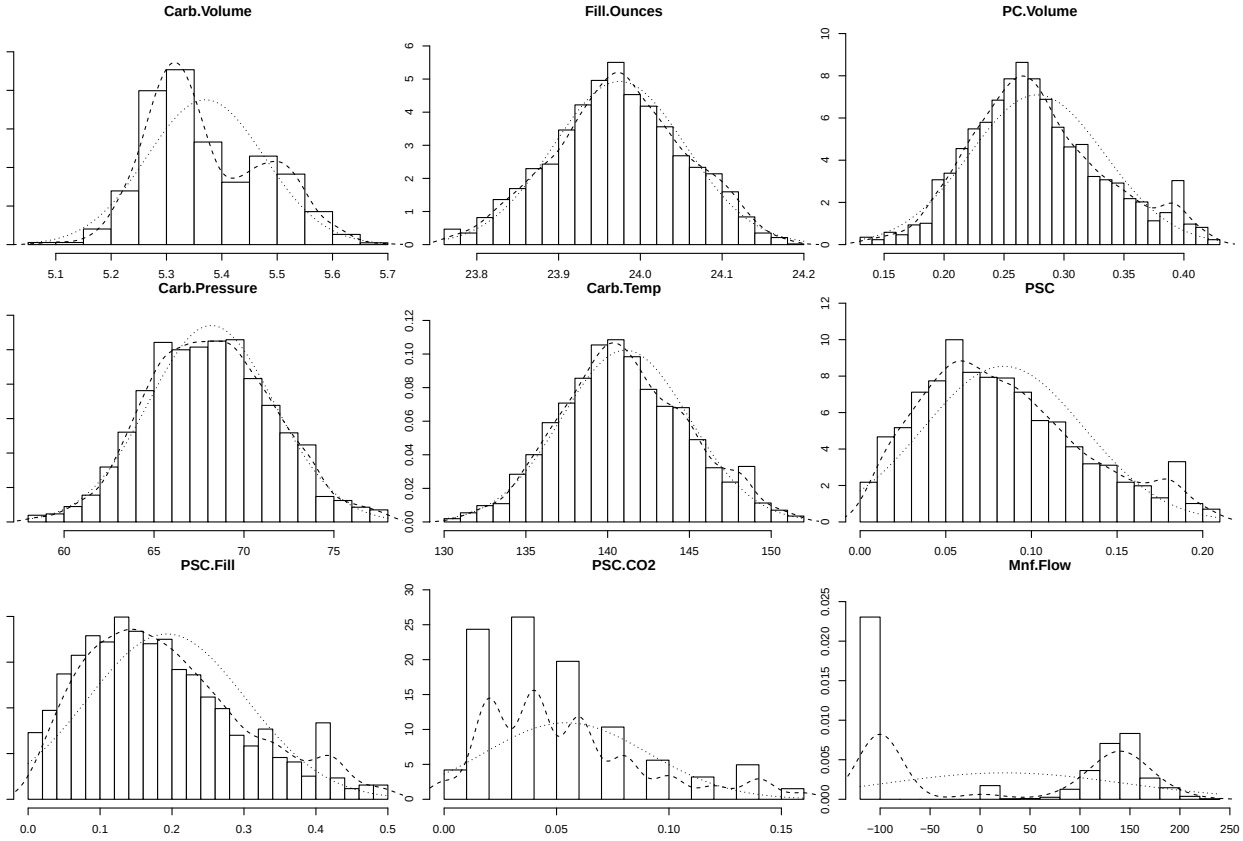


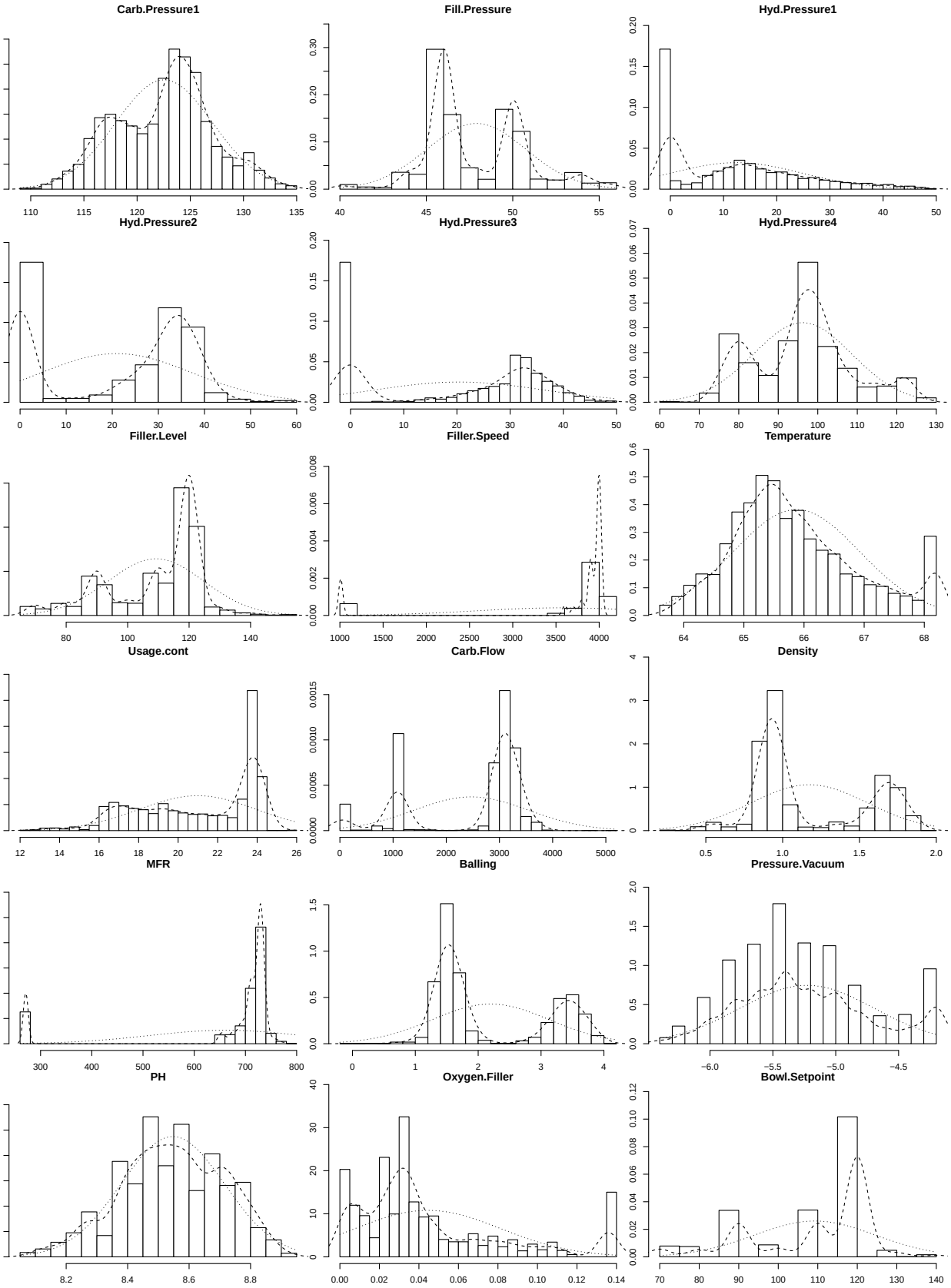
The outliers were caped and now we see that several fields PSC.FILL, PSC.C02, Mnf.Flow, Hyd.Pressure1, Hyd.Pressure2, Hyd.Pressure3, Usage.cont, Carb.Flow, Density, Balling, Oxygen.Filler, Bowl.Setpoint, Pressure.Setpoint, Alch.Rel, Balling.Lvl have high variance.

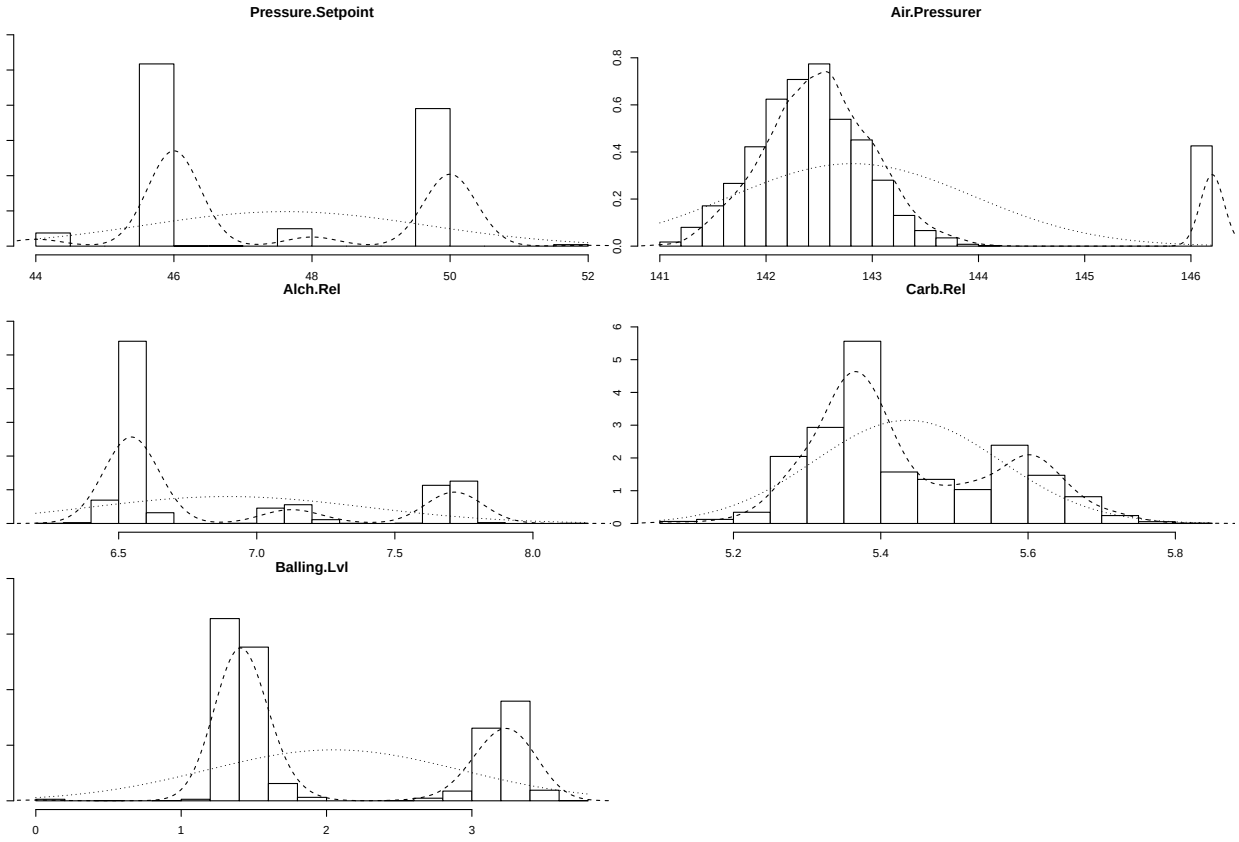
Histograms

Histograms tell us how the data is distributed in the dataset (numeric fields).

```
for(i in seq(from = 2, to = length(Ins_train_cap), by = 9)) {
  if(i <= 27) {
    multi.hist(Ins_train_cap[i:(i + 8)])
  } else {
    multi.hist(Ins_train_cap[i:(i + 4)])
  }
}
```




Carb.Pressure1
Fill.Pressure
Hyd.Pressure1
Hyd.Pressure2
Hyd.Pressure3
Hyd.Pressure4
Filler.Level
Filler.Speed
Temperature
Usage.cont
Carb.Flow
Density
MFR
Balling
Pressure.Vacuum
PH
Oxygen.Filler
Bowl.Setpoint



Observing the above histograms, I decided the critical skewness, needing BoxCox transformation, to be 0.75 or higher. Based on this critical value, I am creating a vector transform_cols, which'll contain the column names of skewed columns.

The columns, whose skewness exceed the critical value of 0.75, are printed below.

```
transform_cols <- c()

for(i in seq(from = 2, to = length(Ins_train_cap), by = 1)) {
  if(abs(skewness(Ins_train_cap[, i])) >= 1) {
    transform_cols <- append(transform_cols, names(Ins_train_cap[i]))
    print(paste0(names(Ins_train_cap[i]), ": ", skewness(Ins_train_cap[, i])))
  }
}
```

```
## [1] "Filler.Speed: -2.2048755979892"
## [1] "MFR: -2.17114153211128"
## [1] "Oxygen.Filler: 1.18681468366543"
## [1] "Air.Pressurer: 2.07962689734424"
```

Many of these histograms are skewed. So, following the recommendations of "Applied Statistical Learning" (page 105, 2nd para), I'll apply Box-Cox transformation to remove the skewness.

```
lambda <- NULL
data_imputed_2 <- Ins_train_cap

for (i in 1:length(transform_cols)) {
  lambda[transform_cols[i]] <- BoxCox.lambda(abs(Ins_train_cap[, transform_cols[i]]))

  data_imputed_2[c(transform_cols[i])] <- BoxCox(Ins_train_cap[transform_cols[i]], lambda[transform_cols
}
```

Now, we don't need to observe the histograms all over again. It will suffice to see the skewness.

We observe that skewness of most or all of the columns reduced and some even reduced to less than 1.

```
for(i in seq(from = 2, to = length(data_imputed_2), by = 1)) {
  if(abs(skewness(data_imputed_2[, i])) >= 1) {
    print(paste0(names(data_imputed_2[i]), ": ", skewness(data_imputed_2[, i])))
  }
}
```

```
## [1] "Filler.Speed: -2.14933682094622"
## [1] "MFR: -2.09361969301246"
## [1] "Air.Pressurer: 2.04308315860091"
```

Categorical variables

Now, we'll explore the Categorical variables.

```
cat('Brand.Code:')
```

```
## Brand.Code:
```

```
table(data_imputed_2$Brand.Code)
```

```
##
##         A    B    C    D
##  120  293 1239  304  615
```

Observation: In Brand.Code column, 120 rows are empty. So, we'll impute them with "X".

```
Ins_train_cap_imputed <- data_imputed_2 %>% mutate(Brand.Code = ifelse((Brand.Code == ""), "X", Brand.C

cat("Brand.Code:")
```

```
## Brand.Code:
```

```
table(Ins_train_cap_imputed$Brand.Code)
```

```
##
##    A    B    C    D    X
##  293 1239  304  615  120
```

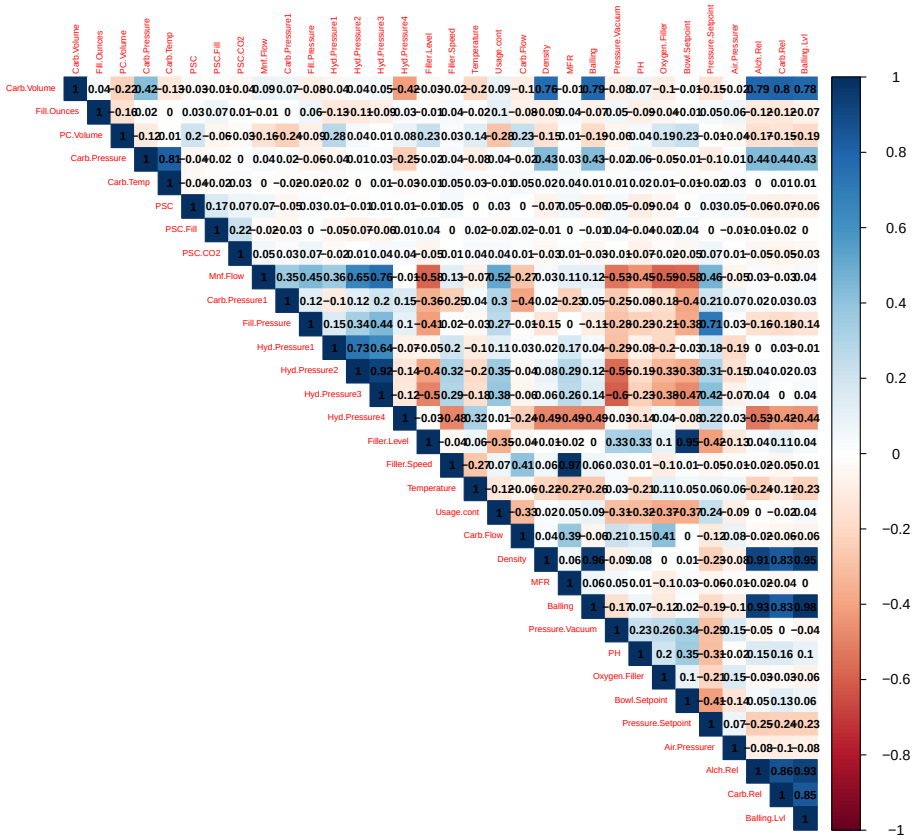
Correlations

At this point the data is prepared. So, we'll explore the top correlated variables.

For the purpose of correlation, we'll remove the only non-numeric field Brand.Code, out of the correlation.

Now, we'll look at the correlation matrix of the variables.

```
Ins_train_corr <- Ins_train_cap_imputed[-1]

cor_mx = cor(Ins_train_corr, use = "pairwise.complete.obs", method = "pearson")

corrplot(cor_mx, method = "color", type = "upper", order = "original", number.cex = .7, addCoef.col = "
                            tl.srt = 90,  # Text label color and rotation
                            diag = TRUE,  # hide correlation coefficient on the principal diagonal
                            tl.cex = 0.5)
```



At this point exploration, preparation and pair-wise correlations of **StudentData.csv** are done. So, I'll begin the same exericse for **StudentEvaluation.csv**.

Data Exploration of StudentEvaluation.csv.

Initially, we'll do a cursory exploration of the data. After that, we'll iteratively prepare and explore the data, wherever required.

```
dim2 <- dim(Ins_eval_data)
print(paste0('Dimension of training set:   ', 'Number of rows: ', dim2[1], ', ', 'Number of cols: ', di
```

```
## [1] "Dimension of training set:   Number of rows: 267, Number of cols: 33"
```

```
print('Head of training data set:')
```

```
## [1] "Head of training data set:"
```

```
head(Ins_eval_data)
```

```
##   Brand.Code Carb.Volume Fill.Ounces PC.Volume Carb.Pressure Carb.Temp   PSC
## 1          D    5.480000    24.03333 0.2700000          65.4     134.6 0.236
## 2          A    5.393333    23.95333 0.2266667          63.2     135.0 0.042
## 3          B    5.293333    23.92000 0.3033333          66.4     140.4 0.068
## 4          B    5.266667    23.94000 0.1860000          64.8     139.0 0.004
## 5          B    5.406667    24.20000 0.1600000          69.4     142.2 0.040
## 6          B    5.286667    24.10667 0.2120000          73.4     147.2 0.078
##   PSC.Fill PSC.CO2 Mnf.Flow Carb.Pressure1 Fill.Pressure Hyd.Pressure1
## 1     0.40    0.04     -100          116.6          46.0             0
## 2     0.22    0.08     -100          118.8          46.2             0
## 3     0.10    0.02     -100          120.2          45.8             0
## 4     0.20    0.02     -100          124.8          40.0             0
## 5     0.30    0.06     -100          115.0          51.4             0
## 6     0.22      NA     -100          118.6          46.4             0
##   Hyd.Pressure2 Hyd.Pressure3 Hyd.Pressure4 Filler.Level Filler.Speed
## 1            NA            NA            96        129.4         3986
## 2             0             0           112        120.0         4012
## 3             0             0            98        119.4         4010
## 4             0             0           132        120.2           NA
## 5             0             0            94        116.0         4018
## 6             0             0            94        120.4         4010
##   Temperature Usage.cont Carb.Flow Density   MFR Balling Pressure.Vacuum PH
## 1        66.0      21.66      2950    0.88 727.6   1.398            -3.8 NA
## 2        65.6      17.60      2916    1.50 735.8   2.942            -4.4 NA
## 3        65.6      24.18      3056    0.90 734.8   1.448            -4.2 NA
## 4        74.4      18.12        28    0.74    NA   1.056            -4.0 NA
## 5        66.4      21.32      3214    0.88 752.0   1.398            -4.0 NA
## 6        66.6      18.00      3064    0.84 732.0   1.298            -3.8 NA
##   Oxygen.Filler Bowl.Setpoint Pressure.Setpoint Air.Pressurer Alch.Rel Carb.Rel
## 1         0.022           130              45.2         142.6     6.56     5.34
## 2         0.030           120              46.0         147.2     7.14     5.58
## 3         0.046           120              46.0         146.6     6.52     5.34
## 4            NA           120              46.0         146.4     6.48     5.50
## 5         0.082           120              50.0         145.8     6.50     5.38
## 6         0.064           120              46.0         146.0     6.50     5.42
##   Balling.Lvl
```

```
## 1       1.48
## 2       3.04
## 3       1.46
## 4       1.48
## 5       1.46
## 6       1.44
```

```
print('Structure of training data set:')
```

```
## [1] "Structure of training data set:"
```

```
str(Ins_eval_data)
```

```
## 'data.frame':    267 obs. of  33 variables:
##  $ Brand.Code       : chr  "D" "A" "B" "B" ...
##  $ Carb.Volume      : num  5.48 5.39 5.29 5.27 5.41 ...
##  $ Fill.Ounces      : num  24 24 23.9 23.9 24.2 ...
##  $ PC.Volume        : num  0.27 0.227 0.303 0.186 0.16 ...
##  $ Carb.Pressure    : num  65.4 63.2 66.4 64.8 69.4 73.4 65.2 67.4 66.8 72.6 ...
##  $ Carb.Temp        : num  135 135 140 139 142 ...
##  $ PSC              : num  0.236 0.042 0.068 0.004 0.04 0.078 0.088 0.076 0.246 0.146 ...
##  $ PSC.Fill         : num  0.4 0.22 0.1 0.2 0.3 0.22 0.14 0.1 0.48 0.1 ...
##  $ PSC.CO2          : num  0.04 0.08 0.02 0.02 0.06 NA 0 0.04 0.04 0.02 ...
##  $ Mnf.Flow         : num  -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...
##  $ Carb.Pressure1   : num  117 119 120 125 115 ...
##  $ Fill.Pressure    : num  46 46.2 45.8 40 51.4 46.4 46.2 40 43.8 40.8 ...
##  $ Hyd.Pressure1    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure2    : num  NA 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure3    : num  NA 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure4    : int  96 112 98 132 94 94 108 108 110 106 ...
##  $ Filler.Level     : num  129 120 119 120 116 ...
##  $ Filler.Speed     : int  3986 4012 4010 NA 4018 4010 4010 NA 4010 1006 ...
##  $ Temperature      : num  66 65.6 65.6 74.4 66.4 66.6 66.8 NA 65.8 66 ...
##  $ Usage.cont       : num  21.7 17.6 24.2 18.1 21.3 ...
##  $ Carb.Flow        : int  2950 2916 3056 28 3214 3064 3042 1972 2502 28 ...
##  $ Density          : num  0.88 1.5 0.9 0.74 0.88 0.84 1.48 1.6 1.52 1.48 ...
##  $ MFR              : num  728 736 735 NA 752 ...
##  $ Balling          : num  1.4 2.94 1.45 1.06 1.4 ...
##  $ Pressure.Vacuum  : num  -3.8 -4.4 -4.2 -4 -4 -3.8 -4.2 -4.4 -4.4 -4.2 ...
##  $ PH               : logi  NA NA NA NA NA NA ...
##  $ Oxygen.Filler    : num  0.022 0.03 0.046 NA 0.082 0.064 0.042 0.096 0.046 0.096 ...
##  $ Bowl.Setpoint    : int  130 120 120 120 120 120 120 120 120 120 ...
##  $ Pressure.Setpoint: num  45.2 46 46 46 50 46 46 46 46 46 ...
##  $ Air.Pressurer    : num  143 147 147 146 146 ...
##  $ Alch.Rel         : num  6.56 7.14 6.52 6.48 6.5 6.5 7.18 7.16 7.14 7.78 ...
##  $ Carb.Rel         : num  5.34 5.58 5.34 5.5 5.38 5.42 5.46 5.42 5.44 5.52 ...
##  $ Balling.Lvl      : num  1.48 3.04 1.46 1.48 1.46 1.44 3.02 3 3.1 3.12 ...
```

There are few fields, which have missing values, which we'll investigate in greater details later.

Data Preparation of StudentEvaluation.csv.

At this stage, we'll explore and prepare iteratively. Now, we'll check for NA. After that if required, we'll impute them.

After that we'll show some boxplots of the numeric fields.

Checking for NA.

```
any(is.na(Ins_eval_data))
```

```
## [1] TRUE
```

NA does exist. So, we'll impute with mice().

```
Ins_eval_imputed <- mice(Ins_eval_data, m = 1, method = "pmm", print = F) %>% mice::complete()
```

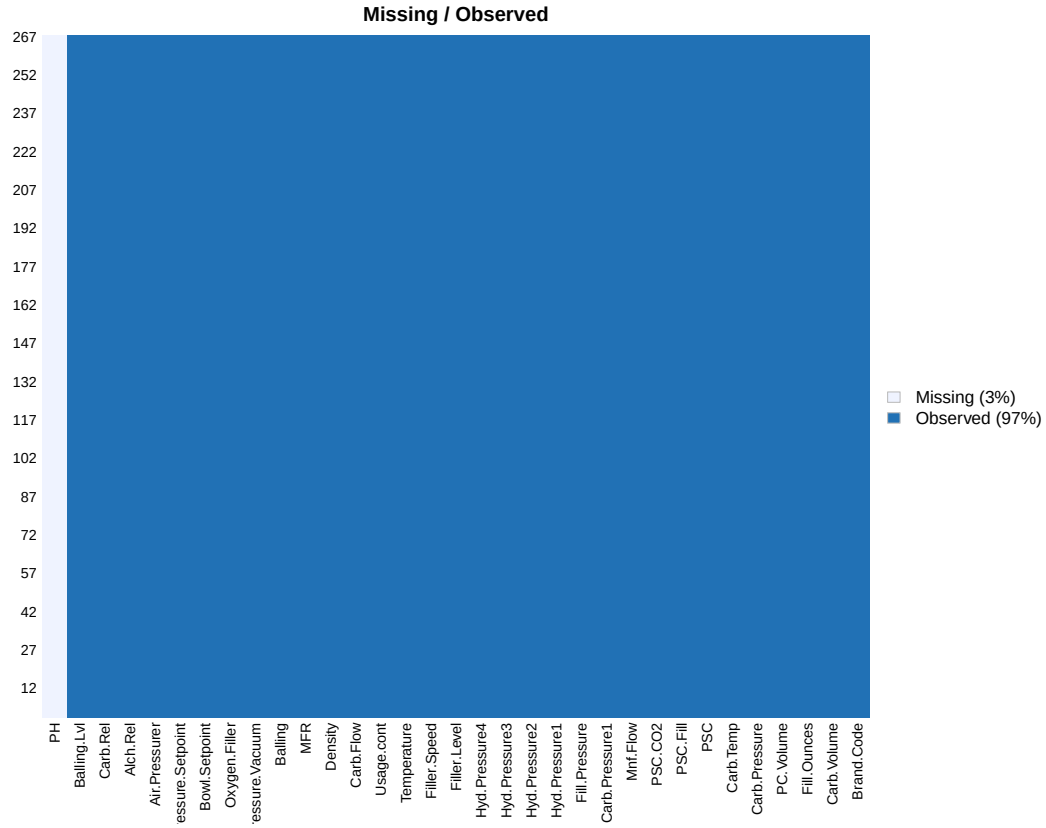
Rechecking for NA after imputation.

```
any(is.na(subset(Ins_eval_imputed, select = -c(PH))))
```

```
## [1] FALSE
```

We observe that NA were removed in all columns except TARGET_FLAG and TARGET_AMT, which is what we want. In the following, we'll visualize with missmap().

```
Ins_eval_imputed %>% missmap(main = "Missing / Observed")
```



Both is.na() and missmap() confirm that NA were eliminated.

More Data exploration of StudentEvaluation.csv.

Now, we'll explore the data a little further. First, we'll take a quick look at min, 1st quartile, median, mean, 2nd quartile, max etc.

```
summary(Ins_eval_imputed) # %>% kable
```

```
##   Brand.Code          Carb.Volume     Fill.Ounces      PC.Volume
##  Length:267         Min.   :5.147   Min.   :23.75   Min.   :0.09867
##  Class :character   1st Qu.:5.287   1st Qu.:23.92   1st Qu.:0.23433
##  Mode  :character   Median :5.340   Median :23.97   Median :0.27600
##                     Mean   :5.369   Mean   :23.97   Mean   :0.27874
##                     3rd Qu.:5.463   3rd Qu.:24.03   3rd Qu.:0.32233
##                     Max.   :5.667   Max.   :24.20   Max.   :0.46400
##  Carb.Pressure     Carb.Temp           PSC             PSC.Fill
##  Min.   :60.20   Min.   :130.0   Min.   :0.00400   Min.   :0.0200
##  1st Qu.:65.30   1st Qu.:138.4   1st Qu.:0.04600   1st Qu.:0.1000
##  Median :68.00   Median :140.8   Median :0.07800   Median :0.1800
##  Mean   :68.25   Mean   :141.3   Mean   :0.08604   Mean   :0.1907
##  3rd Qu.:70.60   3rd Qu.:143.9   3rd Qu.:0.11200   3rd Qu.:0.2600
##  Max.   :77.60   Max.   :154.0   Max.   :0.24600   Max.   :0.6200
##     PSC.CO2           Mnf.Flow       Carb.Pressure1  Fill.Pressure
##  Min.   :0.00000   Min.   :-100.20   Min.   :113.0   Min.   :37.80
##  1st Qu.:0.02000   1st Qu.:-100.00   1st Qu.:120.1   1st Qu.:46.00
##  Median :0.04000   Median :   0.20   Median :123.4   Median :47.80
##  Mean   :0.05079   Mean   :  21.03   Mean   :123.0   Mean   :48.19
##  3rd Qu.:0.06000   3rd Qu.: 141.30   3rd Qu.:125.5   3rd Qu.:50.20
##  Max.   :0.24000   Max.   : 220.40   Max.   :136.0   Max.   :60.20
##  Hyd.Pressure1    Hyd.Pressure2    Hyd.Pressure3    Hyd.Pressure4
##  Min.   :-50.00   Min.   :-50.00   Min.   :-50.00   Min.   : 68.0
##  1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.:  0.00   1st Qu.: 90.0
##  Median : 10.40   Median : 26.80   Median : 27.60   Median : 98.0
##  Mean   : 12.01   Mean   : 20.04   Mean   : 19.54   Mean   : 98.2
##  3rd Qu.: 20.40   3rd Qu.: 34.80   3rd Qu.: 33.00   3rd Qu.:104.0
##  Max.   : 50.00   Max.   : 61.40   Max.   : 49.20   Max.   :140.0
##   Filler.Level    Filler.Speed   Temperature      Usage.cont      Carb.Flow
##  Min.   : 69.2   Min.   :1006   Min.   :63.80   Min.   :12.90   Min.   :   0
##  1st Qu.:100.6   1st Qu.:3795   1st Qu.:65.40   1st Qu.:18.12   1st Qu.:1083
##  Median :118.6   Median :3918   Median :65.80   Median :21.44   Median :3038
##  Mean   :110.2   Mean   :3521   Mean   :66.24   Mean   :20.89   Mean   :2409
##  3rd Qu.:120.2   3rd Qu.:3996   3rd Qu.:66.60   3rd Qu.:23.74   3rd Qu.:3215
##  Max.   :153.2   Max.   :4020   Max.   :75.40   Max.   :24.60   Max.   :3858
##     Density           MFR           Balling      Pressure.Vacuum
##  Min.   :0.060   Min.   : 15.6   Min.   :0.902   Min.   :-6.400
##  1st Qu.:0.920   1st Qu.:689.4   1st Qu.:1.497   1st Qu.:-5.600
##  Median :0.980   Median :720.6   Median :1.648   Median :-5.200
##  Mean   :1.176   Mean   :654.8   Mean   :2.199   Mean   :-5.178
##  3rd Qu.:1.600   3rd Qu.:730.8   3rd Qu.:3.242   3rd Qu.:-4.800
##  Max.   :1.840   Max.   :784.8   Max.   :3.788   Max.   :-3.600
##     PH          Oxygen.Filler     Bowl.Setpoint   Pressure.Setpoint
##  Mode:logical   Min.   :0.00240   Min.   : 70.0   Min.   :44.00
##  NA's:267       1st Qu.:0.02070   1st Qu.:100.0   1st Qu.:46.00
##                 Median :0.03380   Median :120.0   Median :46.00
```

```
##                 Mean   :0.04726   Mean   :109.6   Mean   :47.73
##                 3rd Qu.:0.05630   3rd Qu.:120.0   3rd Qu.:50.00
##                 Max.   :0.39800   Max.   :130.0   Max.   :52.00
##  Air.Pressurer      Alch.Rel        Carb.Rel     Balling.Lvl
##  Min.   :141.2   Min.   :6.400   Min.   :5.18   Min.   :0.000
##  1st Qu.:142.2   1st Qu.:6.540   1st Qu.:5.34   1st Qu.:1.380
##  Median :142.6   Median :6.580   Median :5.40   Median :1.480
##  Mean   :142.8   Mean   :6.903   Mean   :5.44   Mean   :2.051
##  3rd Qu.:142.8   3rd Qu.:7.180   3rd Qu.:5.56   3rd Qu.:3.080
##  Max.   :147.2   Max.   :7.820   Max.   :5.74   Max.   :3.420
```

Zeroing PH column

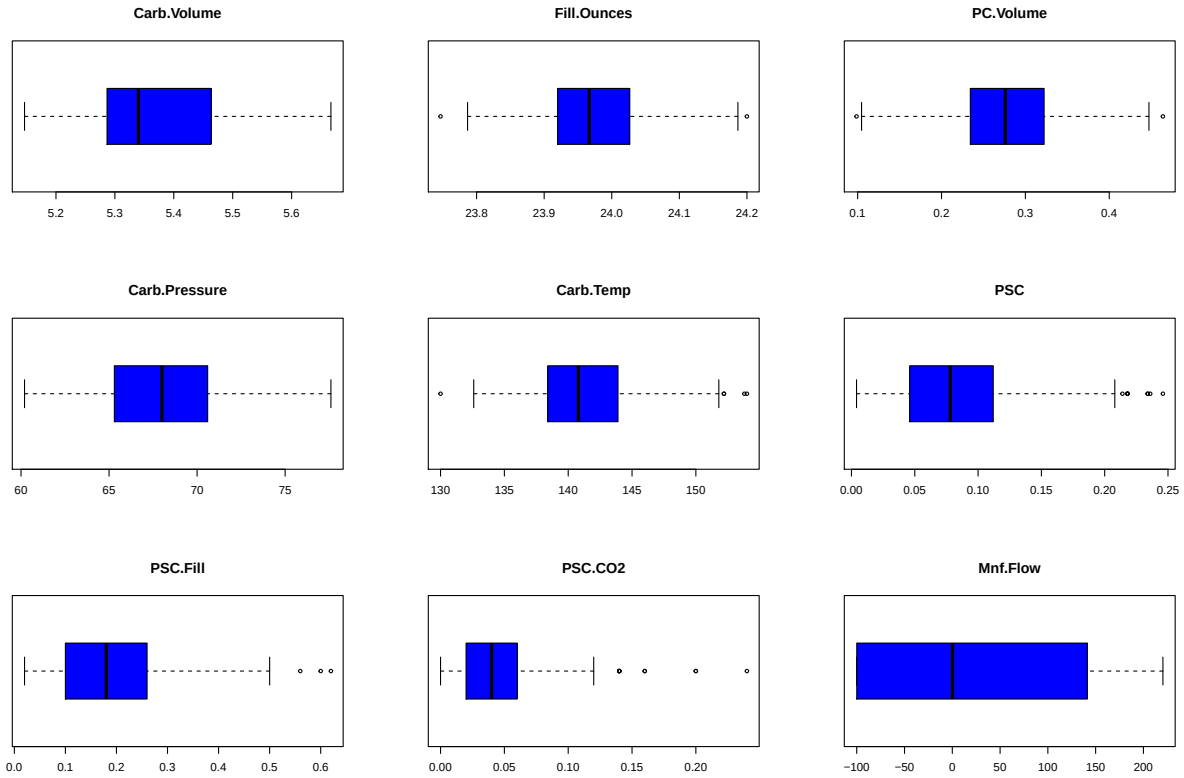
Currently, PH has NA. We'll insert zero into column PH, for convenience of analysis.

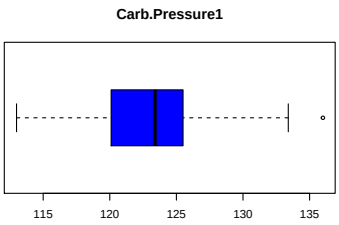
```
Ins_eval_imputed$PH[is.na(Ins_eval_imputed$PH)] <- 0
```

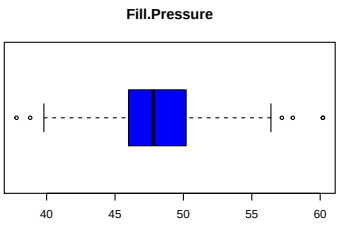
Boxplots

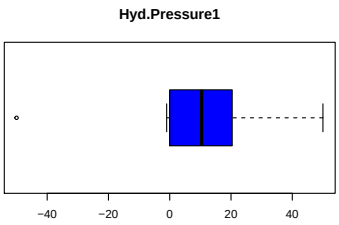
Let's take a first look at the boxplots

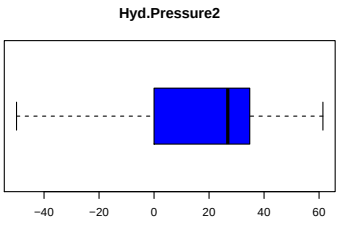
```
par(mfrow = c(3, 3))
for(i in 2:33) {
    if (is.numeric(Ins_eval_imputed[,i])) {
      boxplot(Ins_eval_imputed[,i], main = names(Ins_eval_imputed[i]), col = 4, horizontal = TRUE)
   }
}
```

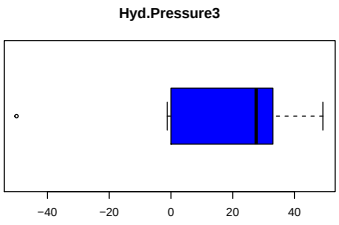


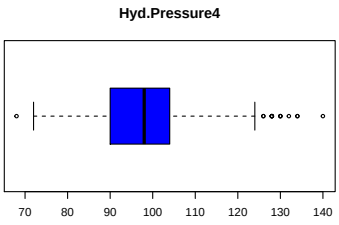

Carb.Pressure1
115
120
125
130
135

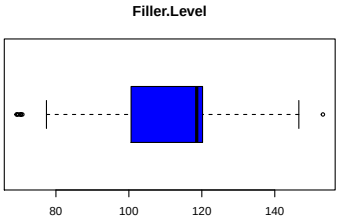

Fill.Pressure
40
45
50
55
60

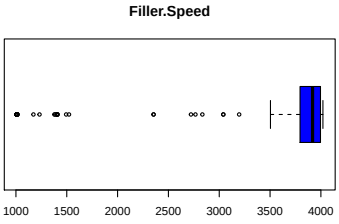

Hyd.Pressure1
−40
−20
0
20
40

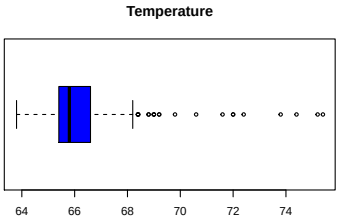

Hyd.Pressure2
−40
−20
0
20
40
60

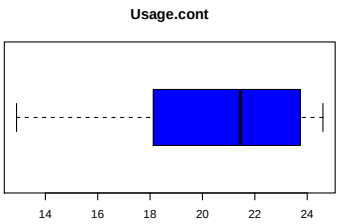

Hyd.Pressure3
−40
−20
0
20
40

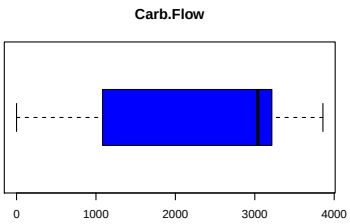

Hyd.Pressure4
70
80
90
100
110
120
130
140

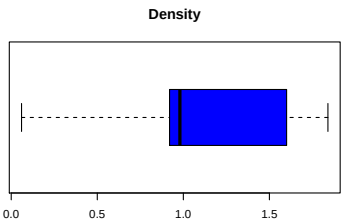

Filler.Level
80
100
120
140

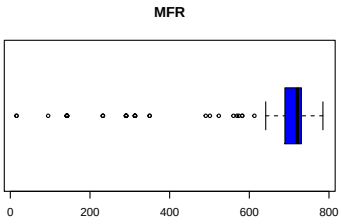

Filler.Speed
1000
1500
2000
2500
3000
3500
4000

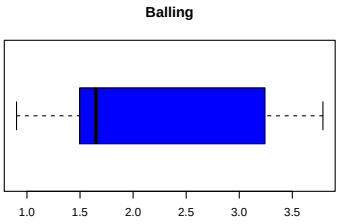

Temperature
64
66
68
70
72
74

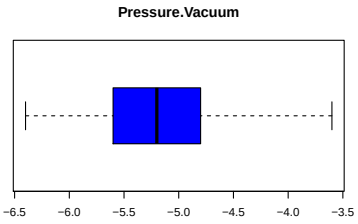

Usage.cont
14
16
18
20
22
24

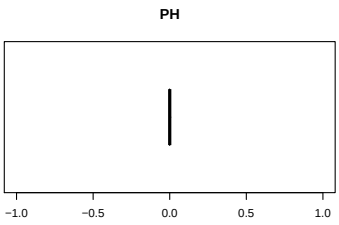

Carb.Flow
0
1000
2000
3000
4000

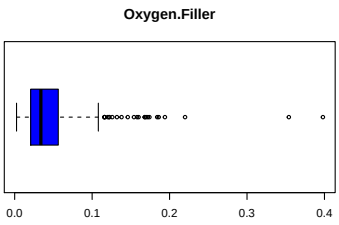

Density
0.0
0.5
1.0
1.5

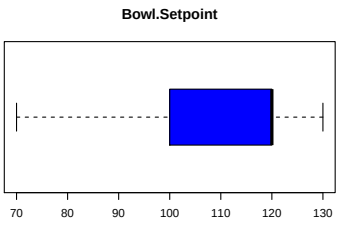

MFR
0
200
400
600
800

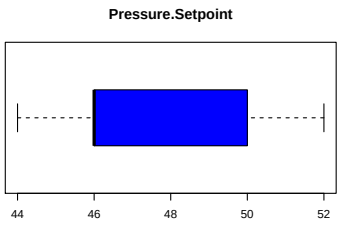

Balling
1.0
1.5
2.0
2.5
3.0
3.5

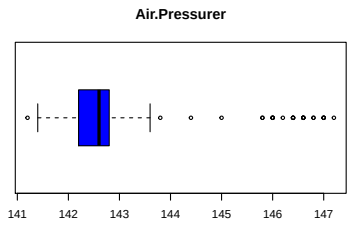

Pressure.Vacuum
−6.5
−6.0
−5.5
−5.0
−4.5
−4.0
−3.5

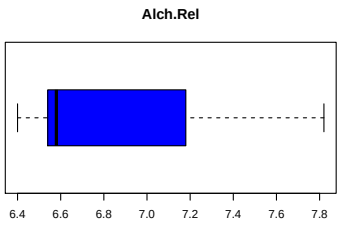

PH
−1.0
−0.5
0.0
0.5
1.0

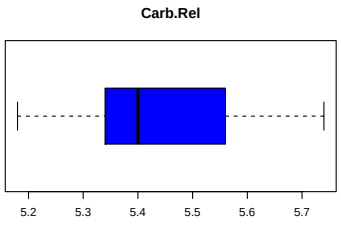

Oxygen.Filler
0.0
0.1
0.2
0.3
0.4

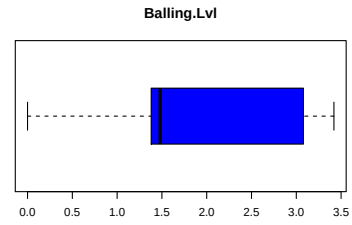

Bowl.Setpoint
70
80
90
100
110
120
130


Pressure.Setpoint
44
46
48
50
52


Air.Pressurer
141
142
143
144
145
146
147


Alch.Rel
6.4
6.6
6.8
7.0
7.2
7.4
7.6
7.8

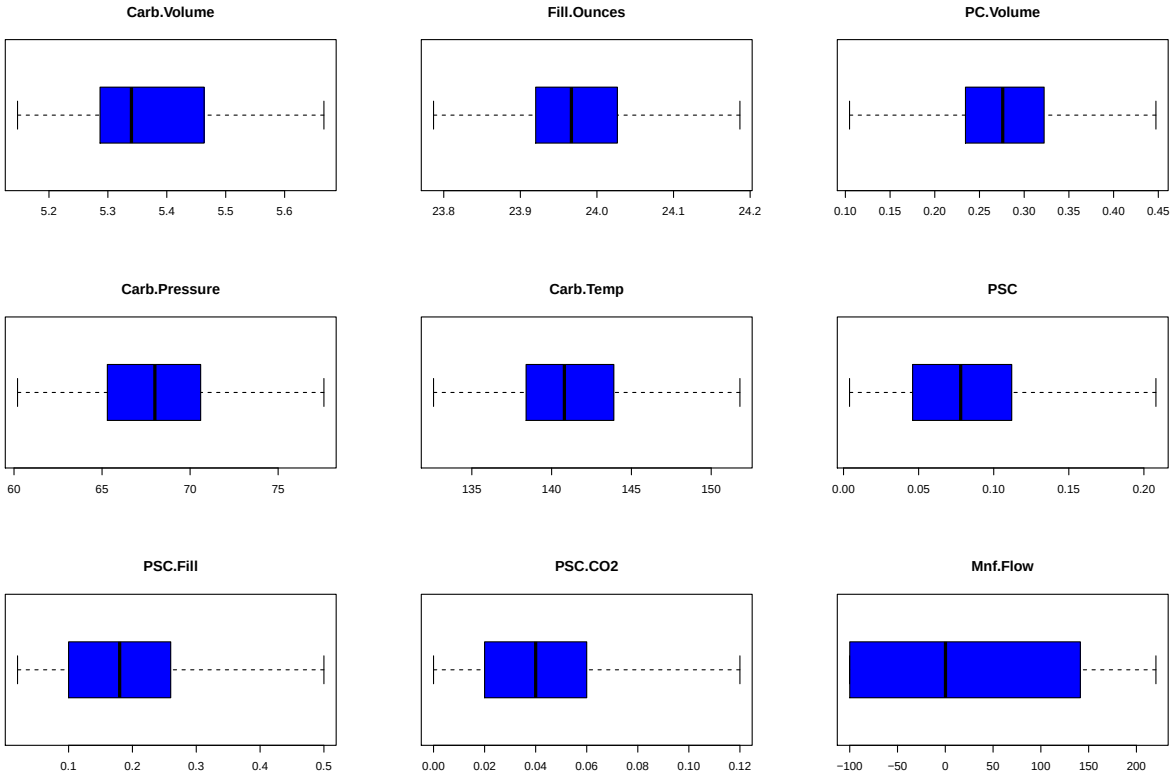

Carb.Rel
5.2
5.3
5.4
5.5
5.6
5.7

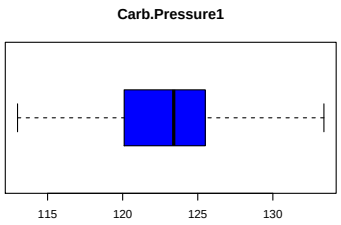

Balling.Lvl
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

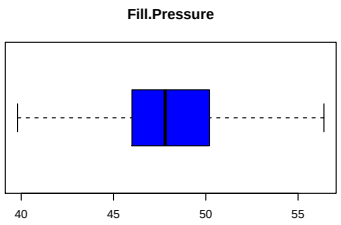
The boxplots show that some of the variables have outliers in them. So, we'll cap them.

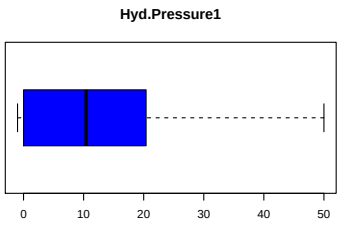
```
Ins_eval_cap <- Ins_eval_imputed

for (i in 2:33) {
  if(i == 26) next # skipping the PH column, which is a 26th column position.

  qntl <- quantile(Ins_eval_cap[,i], probs = c(0.25, 0.75), na.rm = T)
  cap_amt <- quantile(Ins_eval_cap[,i], probs = c(0.05, 0.95), na.rm = T)
  High <- 1.5 * IQR(Ins_eval_cap[,i], na.rm = T)

  Ins_eval_cap[,i][Ins_eval_cap[,i] < (qntl[1] - High)] <- cap_amt[1]
  Ins_eval_cap[,i][Ins_eval_cap[,i] > (qntl[2] + High)] <- cap_amt[2]
}
```

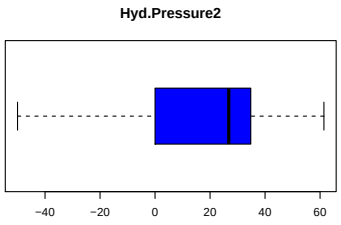
```
par(mfrow = c(3, 3))
for(i in 2:33) {
  if(i == 26) next # skipping the PH column, which is a 26th column position.

    if (is.numeric(Ins_eval_cap[,i])) {
      boxplot(Ins_eval_cap[,i], main = names(Ins_eval_cap[i]), col = 4, horizontal = TRUE)
   }
}
```

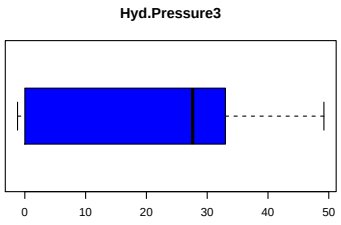


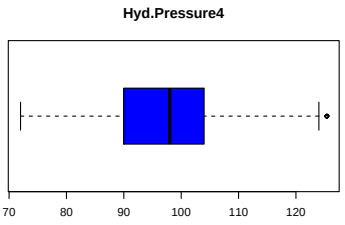

Carb.Pressure1
115
120
125
130

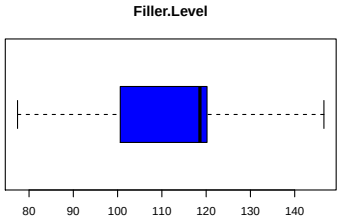

Fill.Pressure
40
45
50
55

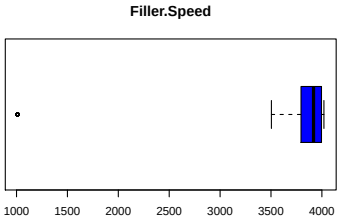

Hyd.Pressure1
0
10
20
30
40
50

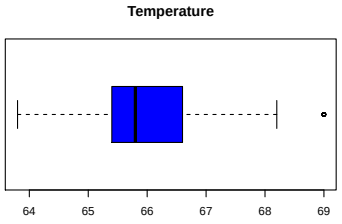

Hyd.Pressure2
−40
−20
0
20
40
60

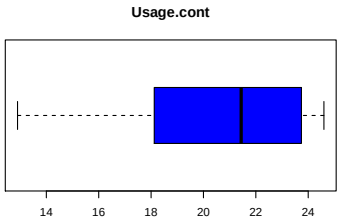

Hyd.Pressure3
0
10
20
30
40
50

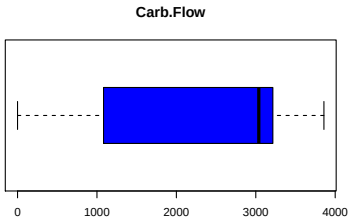

Hyd.Pressure4
70
80
90
100
110
120

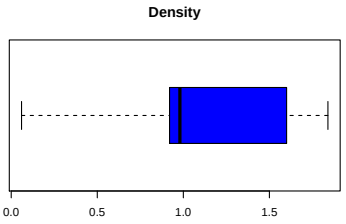

Filler.Level
80
90
100
110
120
130
140

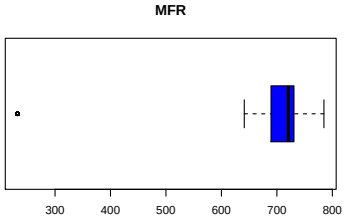

Filler.Speed
1000
1500
2000
2500
3000
3500
4000

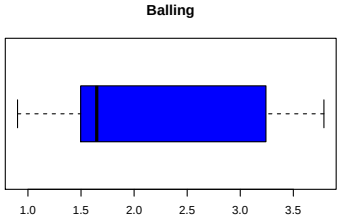

Temperature
64
65
66
67
68
69

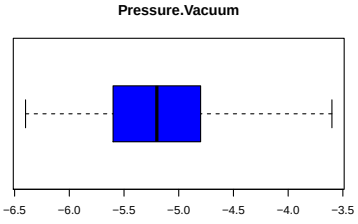

Usage.cont
14
16
18
20
22
24

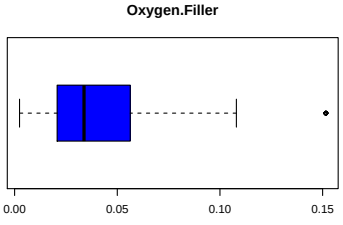

Carb.Flow
0
1000
2000
3000
4000

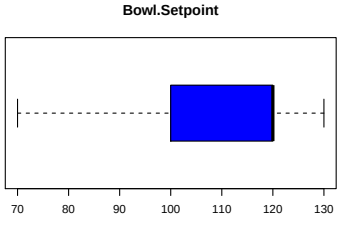

Density
0.0
0.5
1.0
1.5

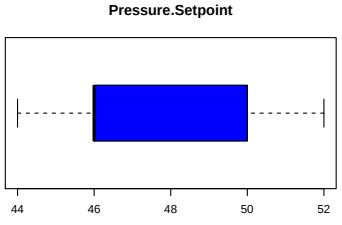

MFR
300
400
500
600
700
800

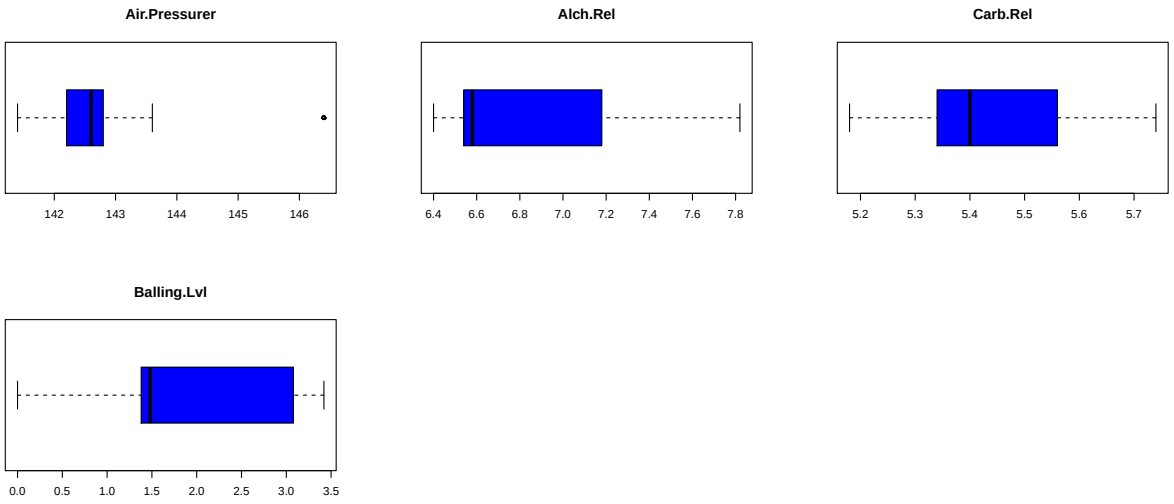

Balling
1.0
1.5
2.0
2.5
3.0
3.5


Pressure.Vacuum
−6.5
−6.0
−5.5
−5.0
−4.5
−4.0
−3.5


Oxygen.Filler
0.00
0.05
0.10
0.15


Bowl.Setpoint
70
80
90
100
110
120
130


Pressure.Setpoint
44
46
48
50
52



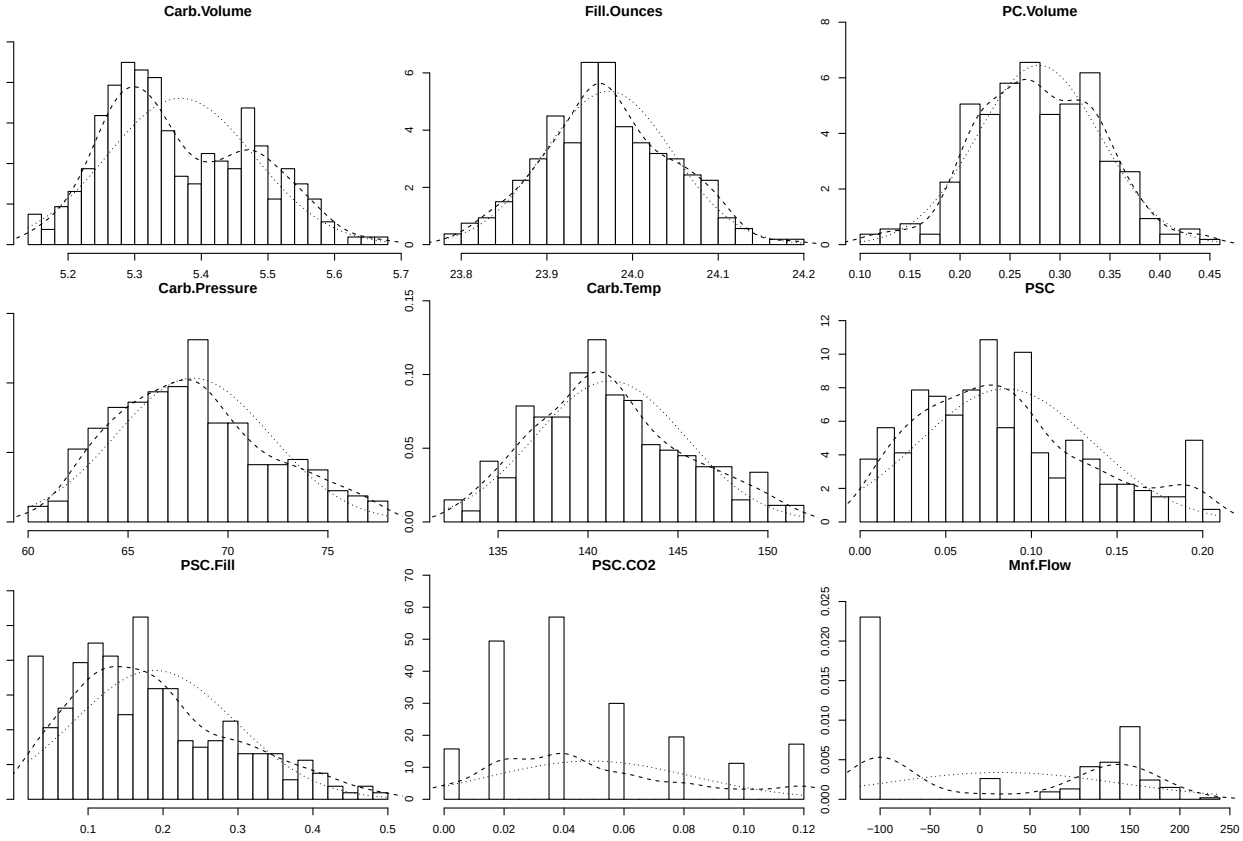
The outliers were caped and now we see that several fields Carb.Volume, PSC.FILL, PSC.C02, Mnf.Flow, Hyd.Pressure1, Hyd.Pressure2, Hyd.Pressure3, Usage.cont, Carb.Flow, Density, Balling, Bowl.Setpoint, Pressure.Setpoint, Alch.Rel, Carb.Rel, Balling.Lvl have high variance.

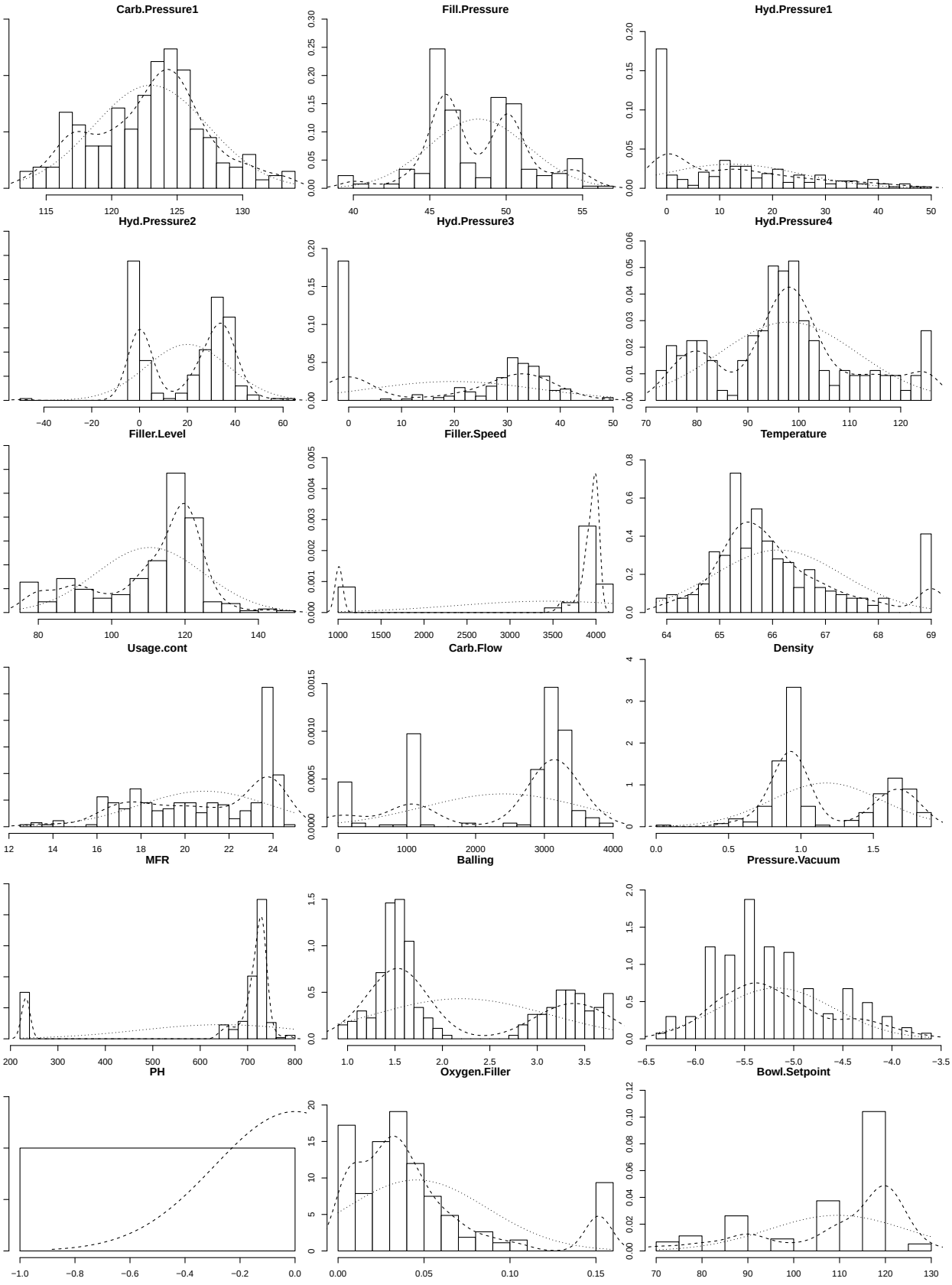
We'll do the boxplots differently, with gglplot, to check if there are any differences.

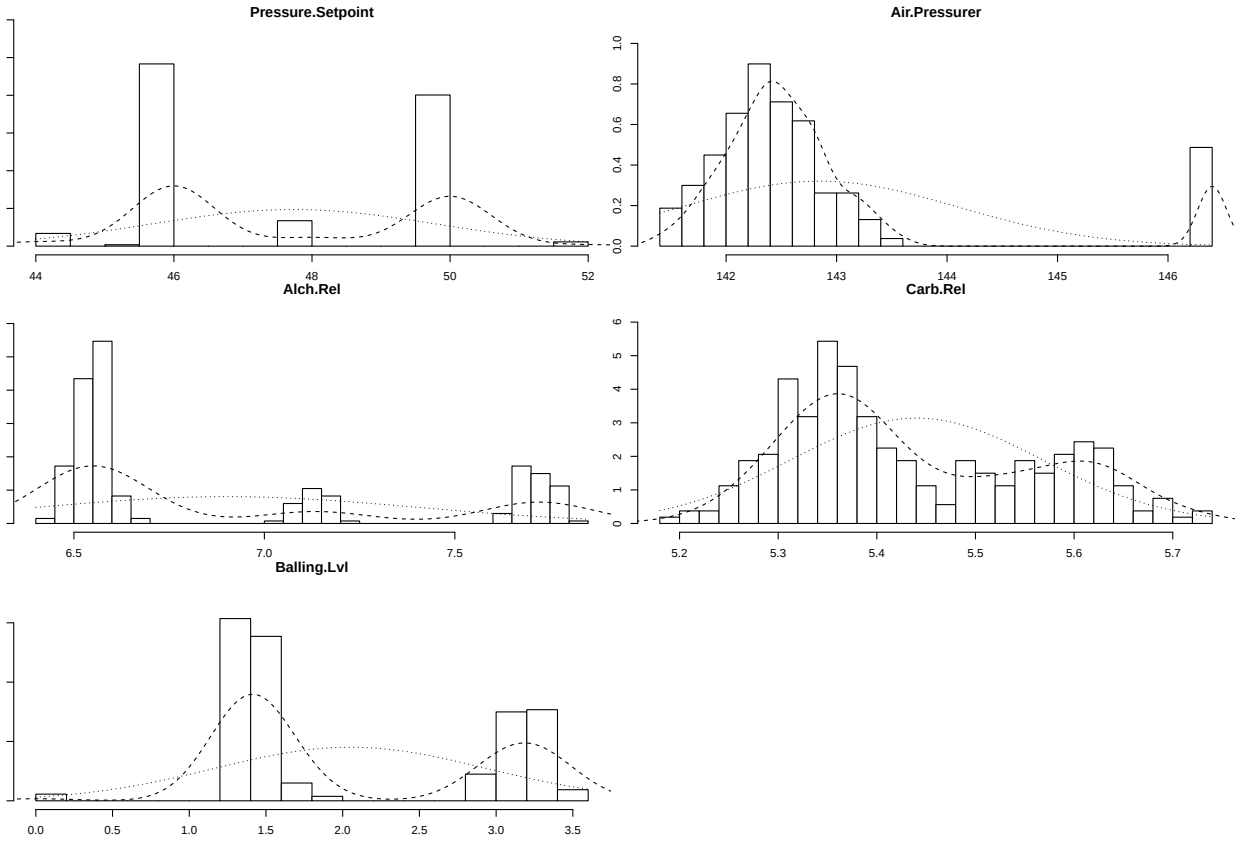
Histograms

Histograms tell us how the data is distributed in the dataset (numeric fields).

```
for(i in seq(from = 2, to = length(Ins_eval_cap), by = 9)) {
  if(i <= 27) {
    multi.hist(Ins_eval_cap[i:(i + 8)])
  } else {
    multi.hist(Ins_eval_cap[i:(i + 4)])
  }
}
```




Carb.Pressure1
Fill.Pressure
Hyd.Pressure1
Hyd.Pressure2
Hyd.Pressure3
Hyd.Pressure4
Filler.Level
Filler.Speed
Temperature
Usage.cont
Carb.Flow
Density
MFR
Balling
Pressure.Vacuum
PH
Oxygen.Filler
Bowl.Setpoint



We can ignore PH, which is target column, where zeros were forced in.

Observing the above histograms, I decided the critical skewness, needing BoxCox transformation, to be 0.75 or higher. Based on this critical value, I am creating a vector transform_cols2, which'll contain the column names of skewed columns.

The columns, whose skewness exceed the critical value of 0.75, are printed below.

```
transform_cols2 <- c()

for(i in seq(from = 2, to = length(Ins_eval_cap), by = 1)) {
  if(i == 26) next # skipping the PH column, which is a 26th column position.

  if(abs(skewness(Ins_eval_cap[, i])) >= 1) {
   transform_cols2 <- append(transform_cols2, names(Ins_eval_cap[i]))
   print(paste0(names(Ins_eval_cap[i]), ": ", skewness(Ins_eval_cap[, i])))
  }
}
```

```
## [1] "Filler.Speed: -1.76718799176986"
## [1] "MFR: -1.88411587418829"
## [1] "Oxygen.Filler: 1.49445726889183"
## [1] "Bowl.Setpoint: -1.1242132435293"
## [1] "Air.Pressurer: 2.13733821348594"
```

Many of these histograms are skewed. So, following the recommendations of "Applied Statistical Learning" (page 105, 2nd para), I'll apply Box-Cox transformation to remove the skewness.

```r
lambda <- NULL
data_imputed_3 <- Ins_eval_cap

for (i in 1:length(transform_cols2)) {
  lambda[transform_cols2[i]] <- BoxCox.lambda(abs(Ins_eval_cap[, transform_cols2[i]]))

  data_imputed_3[c(transform_cols2[i])] <- BoxCox(Ins_eval_cap[transform_cols2[i]], lambda[transform_co
}
```

Now, we don't need to observe the histograms all over again. It will suffice to see the skewness.

We observe that skewness of most or all of the columns reduced and some even reduced to less than 1.

```r
for(i in seq(from = 2, to = length(data_imputed_3), by = 1)) {
  if(i == 26) next # skipping the PH column, which is a 26th column position.

  if(abs(skewness(data_imputed_3[, i])) >= 1) {
    print(paste0(names(data_imputed_3[i]), ": ", skewness(data_imputed_3[, i])))
  }
}
```

```
## [1] "Filler.Speed: -1.72815857984234"
## [1] "MFR: -1.80898001927621"
## [1] "Air.Pressurer: 2.11075447000935"
```

Categorical variables

Now, we'll explore the Categorical variables.

```
cat('Brand.Code:')
```

```
## Brand.Code:
```

```
table(data_imputed_3$Brand.Code)
```

```
##
##       A   B   C   D
##   8  35 129  31  64
```

Observation: In Brand.Code column, 120 rows are empty. So, we'll impute them with "X".

```
Ins_eval_cap_imputed <- data_imputed_3 %>% mutate(Brand.Code = ifelse((Brand.Code == ""), "X", Brand.Co

cat("Brand.Code:")
```

```
## Brand.Code:
```

```
table(Ins_eval_cap_imputed$Brand.Code)
```

```
##
##   A   B   C   D   X
##  35 129  31  64   8
```

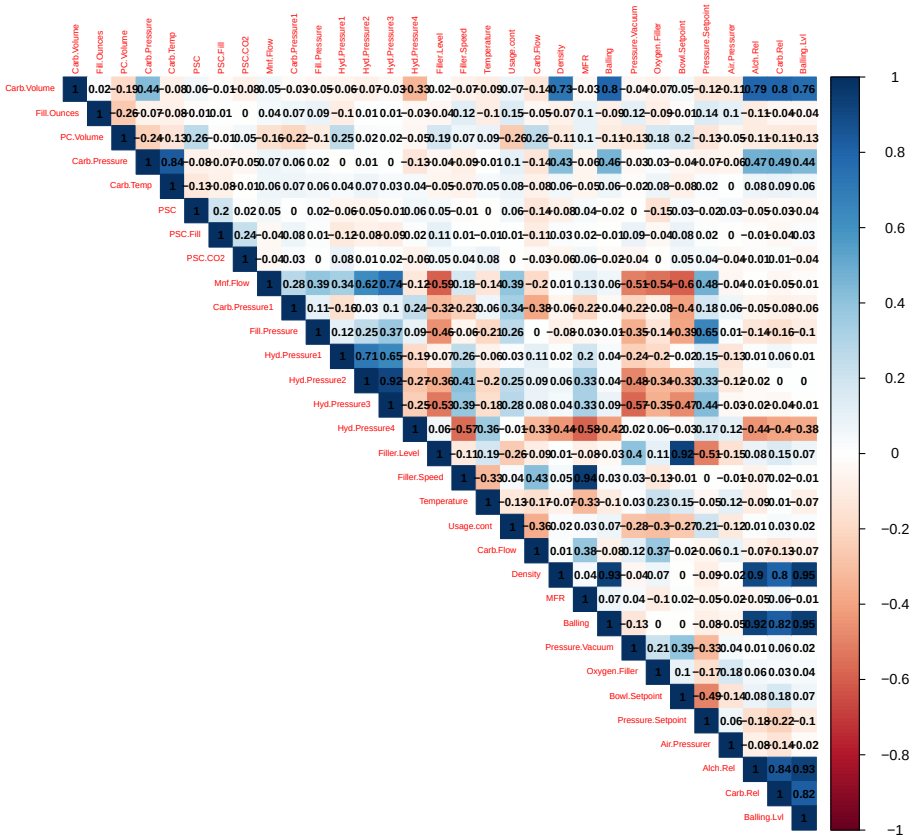
Correlations

At this point the data is prepared. So, we'll explore the top correlated variables.

For the purpose of correlation, we'll remove the only non-numeric field Brand.Code, out of the correlation.

Now, we'll look at the correlation matrix of the variables.

```
Ins_cap_corr <- subset(Ins_eval_cap_imputed, select = -c(Brand.Code, PH))

cor_mx = cor(Ins_cap_corr, use = "pairwise.complete.obs", method = "pearson")

corrplot(cor_mx, method = "color", type = "upper", order = "original", number.cex = .7, addCoef.col = "
                            tl.srt = 90,  # Text label color and rotation
                            diag = TRUE,  # hide correlation coefficient on the principal diagonal
                            tl.cex = 0.5)
```



At this point exploration, preparation and pair-wise correlations of **StudentEvaluation.csv** are done. So, I'll begin the building process.

Models

Splitting Test and Train

We will use 80/20 split to create Test and Train data from our Ins_train_cap_imputed file. Since our dataset is not that large, we want to have as much training data available for modeling as possible.

```
set.seed(300)
trainingRows <- createDataPartition(Ins_train_cap_imputed$PH, p = 0.8, list = FALSE)
Ins_train <- Ins_train_cap_imputed[trainingRows, ]
Ins_test <- Ins_train_cap_imputed[-trainingRows, ]

Ins_train_Y <- subset( Ins_train, select = PH )
Ins_train_X <- subset( Ins_train, select = -PH )
Ins_test_Y <- subset( Ins_test, select = PH )
Ins_test_X <- subset( Ins_test, select = -PH )
```

Linear Models

First we are going to try to use linear models to predict the relationship between our predictors and PH values, assuming that the relationship shows a constant rate of change. We do not have very high hopes for these models since there are a lot of limitations associated with Linear Models - in the real world, the data is rarely linearly separable.

GLM Model

First, we will try Generalized Linear model. The GLM generalizes linear regression by allowing the linear model to be related to the response variable via a link function and by allowing the magnitude of the variance of each measurement to be a function of its predicted value.

```
set.seed(300)
lmFit1 = train(PH ~ ., data = Ins_train,
                       metric = 'RMSE',
                       method = 'glm',
                       preProcess = c('center', 'scale'),
                       trControl = trainControl(method = 'cv', number = 5, savePredictions = TRUE)
)

lmFit1_pred <- predict(lmFit1, Ins_test_X)

lmFit1
```

```
## Generalized Linear Model
##
## 2058 samples
##   32 predictor
##
## Pre-processing: centered (35), scaled (35)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1647, 1646, 1647, 1647, 1645
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1301865  0.4060756  0.1022162
```

The GLM R-Squared value is not very high - 0.40, meaning that the model explains 40% of variability in the data. RMSE for GLM is 0.135.

PLS Model

Next, we will try Partial Least Squares model. PSL finds a linear regression model by projecting the predicted variables and the observable variables to a new space. If the correlation among predictors is high, then the partial least squares squares might be a better option. PSL might also be better is the number of predictors may be greater than the number of observations.

```
set.seed(300)
lmFit2 = train(PH ~ ., data = Ins_train,
                       metric = 'RMSE',
                       method = 'pls',
                       preProcess = c('center', 'scale'),
                       trControl = trainControl(method = 'cv', number = 5, savePredictions = TRUE)
)

lmFit2_pred <- predict(lmFit2, Ins_test_X)

lmFit2
```

```
## Partial Least Squares
##
## 2058 samples
##   32 predictor
##
## Pre-processing: centered (35), scaled (35)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1647, 1646, 1647, 1647, 1645
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##   1      0.1458734  0.2509994  0.1163688
##   2      0.1381529  0.3291329  0.1090211
##   3      0.1352101  0.3567068  0.1072727
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 3.
```

The PLS R-Squared value is not very high - 0.37, meaning that the model explains 37% of variability in the data. RMSE for PLS is 0.132.

Ridge Model

Next, we will try some penalized models, we will start with a Ridge model. Ridge regression adds a penalty on the sum of the squared regression parameters.

```
set.seed(300)
ridgeGrid <- data.frame(.lambda = seq(0, .1, length = 15))
ridgeRegFit <- train(x = Ins_train_X[,-1], y = Ins_train_Y$PH,
                       method = "ridge",
                       tuneGrid = ridgeGrid,
                       trControl = trainControl(method = "cv", number = 10),
                       preProc = c("center", "scale")
```

```
                      )
ridgeRegFit
```

```
## Ridge Regression
##
## 2058 samples
##   31 predictor
##
## Pre-processing: centered (31), scaled (31)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1852, 1852, 1852, 1853, 1852, 1853, ...
## Resampling results across tuning parameters:
##
##   lambda       RMSE       Rsquared   MAE
##   0.000000000  0.1354371  0.3560067  0.1059201
##   0.007142857  0.1353470  0.3565510  0.1059690
##   0.014285714  0.1353805  0.3560982  0.1060828
##   0.021428571  0.1354441  0.3554055  0.1061960
##   0.028571429  0.1355215  0.3546028  0.1063121
##   0.035714286  0.1356067  0.3537398  0.1064235
##   0.042857143  0.1356962  0.3528436  0.1065304
##   0.050000000  0.1357882  0.3519307  0.1066289
##   0.057142857  0.1358813  0.3510119  0.1067232
##   0.064285714  0.1359747  0.3500943  0.1068187
##   0.071428571  0.1360680  0.3491827  0.1069119
##   0.078571429  0.1361607  0.3482803  0.1070040
##   0.085714286  0.1362528  0.3473893  0.1070925
##   0.092857143  0.1363440  0.3465111  0.1071803
##   0.100000000  0.1364344  0.3456464  0.1072671
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 0.007142857.
```

```
ridge_pred <- predict(ridgeRegFit, Ins_test_X)
```

The Ridge R-Squared value is not very high - 0.376, meaning that the model explains 38% of variability in the data. RMSE for Ridge is 0.132.

ENET Model

Next, we will try ENET model. Elastic net model has both ridge penalties and lasso penalties.

```
df1_enet <-  train(x = as.matrix(Ins_train_X[,-1]),
                   y = Ins_train_Y$PH,
                   method='enet',
                   metric='RMSE',
                   trControl = trainControl(method = 'cv', number = 5, savePredictions = TRUE))

df1_enet
```

```
## Elasticnet
```

```
##
## 2058 samples
##   31 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1645, 1646, 1646, 1648, 1647
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE       Rsquared   MAE
##   0e+00   0.050     0.1561349  0.2015095  0.1261653
##   0e+00   0.525     0.1359674  0.3497534  0.1070352
##   0e+00   1.000     0.1353090  0.3552157  0.1058922
##   1e-04   0.050     0.1561613  0.2015095  0.1261896
##   1e-04   0.525     0.1359783  0.3496696  0.1070497
##   1e-04   1.000     0.1353032  0.3552711  0.1058916
##   1e-01   0.050     0.1603609  0.2015095  0.1301397
##   1e-01   0.525     0.1391188  0.3198614  0.1102796
##   1e-01   1.000     0.1363073  0.3460939  0.1073337
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were fraction = 1 and lambda = 1e-04.
```

```
enet_pred <- predict(df1_enet, Ins_test_X)
```

The ENet R-Squared value is not very high - 0.319, meaning that the model explains 32% of variability in the data. RMSE for Enet is 0.144.

Comparing Linear Models

As expected, it doesn't look like either of the linear models has a good performance based on their R-squared and RMSE values, but let's compare those and see which model performs best.

```
z<- rbind(
  postResample(pred = lmFit1_pred, obs = Ins_test_Y$PH),
  postResample(pred = lmFit2_pred, obs = Ins_test_Y$PH),
  postResample(pred = ridge_pred, obs = Ins_test_Y$PH),
  postResample(pred = enet_pred, obs = Ins_test_Y$PH)
)

data.frame(z,row.names = c('GLM', 'PLS', 'RIDGE', 'ENET'))
```

```
##            RMSE  Rsquared       MAE
## GLM   0.1270041 0.4235592 0.1007638
## PLS   0.1343780 0.3541478 0.1084992
## RIDGE 0.1308187 0.3878284 0.1016489
## ENET  0.1305934 0.3900558 0.1014632
```

The best linear model based on the highest R-Squared and lowest RSME value is GLM.

Non-Linear Models

Next we will try several Non-Linear models:multivariate adaptive regression splines (MARS), support vector machines (SVMs), and K-nearest neighbors (KNNs). We expect these models to perform better than Linear Models. We will look at Tree models separately.

MARS Model

We will continue modeling by tuning and evaluating a MARS model. MARS uses surrogate features instead of the original predictors.

```
set.seed(200)
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:20)
marsTune <- train(x = Ins_train_X,
                  y = Ins_train_Y$PH,
                  method = "earth",
                  preProc=c("center", "scale"),
                  tuneGrid= marsGrid,
                  trControl = trainControl(method = "cv"))
```

```
## Loading required package: earth
```

```
## Warning: package 'earth' was built under R version 3.6.3
```

```
## Loading required package: Formula
```

```
## Warning: package 'Formula' was built under R version 3.6.3
```

```
## Loading required package: plotmo
```

```
## Warning: package 'plotmo' was built under R version 3.6.3
```

```
## Loading required package: plotrix
```

```
## Warning: package 'plotrix' was built under R version 3.6.3
```

```
##
## Attaching package: 'plotrix'
```

```
## The following object is masked from 'package:psych':
##
##     rescale
```

```
## Loading required package: TeachingDemos
```

```
## Warning: package 'TeachingDemos' was built under R version 3.6.3
```

```
##
## Attaching package: 'plotmo'
```

```
## The following object is masked from 'package:urca':
##
##     plotres
```

Evaluating MARS model's performance:

```
marsPred = predict(marsTune, newdata = Ins_test_X)
postResample(pred = marsPred, obs = Ins_test_Y$PH)
```

```
##       RMSE   Rsquared        MAE
## 0.11687735 0.51394588 0.08898694
```

The MARS R-Squared value is 0.506, meaning that the model explains 51% of variability in the data. RMSE for MARS is 0.122.

SVM Model

The next model we will tune and evaluate is SVM model - I will use pre-process to center and scale the data and will use tune length of 10. The benefist of SVM are that, since the squared residuals are not used, large outliers have a limited effect on the regression equation. Second, samples that the model fits well have no effect on the regression equation.

```
set.seed(200)
svmTuned = train(x = Ins_train_X[,-1],
                 y = Ins_train_Y$PH,
                 method="svmRadial",
                 preProc=c("center", "scale"),
                 tuneLength=10,
                 trControl = trainControl(method = "repeatedcv"))

svmTuned
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2058 samples
##   31 predictor
##
## Pre-processing: centered (31), scaled (31)
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 1853, 1852, 1851, 1852, 1853, 1853, ...
## Resampling results across tuning parameters:
##
##   C       RMSE       Rsquared   MAE
##     0.25  0.1269786  0.4413714  0.09578562
##     0.50  0.1236352  0.4673079  0.09230783
##     1.00  0.1209517  0.4887998  0.08975190
##     2.00  0.1191973  0.5033202  0.08823290
##     4.00  0.1184197  0.5110685  0.08745544
##     8.00  0.1194729  0.5069260  0.08835530
##    16.00  0.1210187  0.5014635  0.08958548
##    32.00  0.1240928  0.4897067  0.09231411
##    64.00  0.1288739  0.4679796  0.09608113
```

```
##   128.00  0.1337186  0.4471950  0.10014367
##
## Tuning parameter 'sigma' was held constant at a value of 0.02221953
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.02221953 and C = 4.
```

```
SVMPred = predict(svmTuned, newdata = Ins_test_X[,-1])
postResample(pred = SVMPred, obs = Ins_test_Y$PH)
```

```
##      RMSE  Rsquared       MAE
## 0.1181665 0.5100057 0.0844474
```

The SVM R-Squared value is 0.432, meaning that the model explains 43% of variability in the data. RMSE for SVM is 0.133.

KNN Model

The next Non-Linear model we will tune and evaluate is KNN model. The KNN approach predicts a new sample using the K-closest samples from the training set.

```
set.seed(333)
knnModel <- train(x = Ins_train_X[,-1],
                  y = Ins_train_Y$PH,
                  method = "knn",
                  preProc = c("center", "scale"),
                  tuneLength = 10)
knnModel
```

```
## k-Nearest Neighbors
##
## 2058 samples
##   31 predictor
##
## Pre-processing: centered (31), scaled (31)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2058, 2058, 2058, 2058, 2058, 2058, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    5  0.1368120  0.3769043  0.10108189
##    7  0.1328200  0.3982213  0.09877433
##    9  0.1313671  0.4060634  0.09828788
##   11  0.1303900  0.4126733  0.09806581
##   13  0.1301276  0.4147379  0.09831833
##   15  0.1301600  0.4141980  0.09872267
##   17  0.1305650  0.4106228  0.09923355
##   19  0.1310104  0.4065626  0.09978585
##   21  0.1314057  0.4030552  0.10036307
##   23  0.1316947  0.4008629  0.10082642
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 13.
```

Evaluating the model's performance:

```
knnPred <- predict(knnModel, newdata = Ins_test_X[,-1])
postResample(pred = knnPred, obs = Ins_test_Y$PH)
```

```
##       RMSE   Rsquared        MAE
## 0.12506420 0.44893881 0.09301673
```

The SVM R-Squared value is 0.416, meaning that the model explains 42% of variability in the data. RMSE for SVM is 0.135.

Comparing Non-Linear Models

It looks like non-linear models are performing better than the linear models based on their R-squared values, but let's compare those and see which model performs best.

```
z<- rbind(
  postResample(pred = marsPred, obs = Ins_test_Y$PH),
  postResample(pred = SVMPred, obs = Ins_test_Y$PH),
  postResample(pred = knnPred, obs = Ins_test_Y$PH)
)

data.frame(z,row.names = c('MARS', 'SVM', 'KNN'))
```

```
##           RMSE  Rsquared        MAE
## MARS 0.1168773 0.5139459 0.08898694
## SVM  0.1181665 0.5100057 0.08444740
## KNN  0.1250642 0.4489388 0.09301673
```

The best non-linear model based on the highest R-Squared and lowest RSME value is MARS

Tree Models

We will now try some Tree models. Decision tree analysis involves making a tree-shaped diagram to chart out a course of action or a statistical probability analysis. It is used to break down complex problems or branches. Each branch of the decision tree could be a possible outcome.

Random Forest

First, we will try a Random Forest Model, these model achieves variance reduction by selecting strong, complex learners that exhibit low bias. Because each learner is selected independently of all previous learners, random forests is robust to a noisy response.

```
suppressWarnings(library(randomForest))
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:psych':
##
##     outlier

## The following object is masked from 'package:gridExtra':
##
##     combine

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(333)
RF_model <- randomForest(x = Ins_train_X[,-1],
                  y = Ins_train_Y$PH,
                  importance = TRUE,
                  ntree = 700
                  )

RFPred <- predict(RF_model, newdata = Ins_test_X[,-1])
postResample(pred = RFPred, obs = Ins_test_Y$PH)
```

```
##       RMSE   Rsquared        MAE
## 0.09718891 0.68219200 0.07064440
```

The Random Forest R-Squared value is 0.641, meaning that the model explains 64% of variability in the data. RMSE for Random Forest is 0.109.

Boosted trees

Next, we will try a Boosted Tree Model. The basic principles of gradient boosting are as follows: given a loss function (e.g., squared error for regression) and a weak learner (e.g., regression trees), the algorithm seeks to find an additive model that minimizes the loss function.

```
suppressWarnings(library(gbm))
```

```
## Loaded gbm 2.1.8
```

```
set.seed(333)
gbmGrid <- expand.grid(.interaction.depth = seq(1, 5, by = 2),
                       .n.trees = seq(300, 1000, by = 100),
                       .shrinkage = c(0.05, 0.1),
                       .n.minobsinnode = 5)
```

```
gbmTune <- suppressWarnings(train(Ins_train_X[,-1], Ins_train_Y$PH,
                 method = "gbm",
                 tuneGrid = gbmGrid,
                 verbose = FALSE)
                 )

GBM_Pred <- predict(gbmTune, newdata = Ins_test_X[,-1])
postResample(pred = GBM_Pred, obs = Ins_test_Y$PH)
```

```
##       RMSE   Rsquared        MAE
## 0.10310588 0.61984238 0.07703696
```

The Boosted Tree R-Squared value is 0.578, meaning that the model explains 58% of variability in the data. RMSE for Boosted Tree is 0.114.

Single Tree

Next, we will try a Single Tree Model. Basic regression trees partition the data into smaller groups that are more homogenous with respect to the response.

```
set.seed(333)
rpartTune <- train(Ins_train_X, Ins_train_Y$PH,
                   method = "rpart2",
                   tuneLength = 10,
                   trControl = trainControl(method = "cv"))

ST_Pred <- predict(rpartTune, newdata = Ins_test_X)
postResample(pred = ST_Pred, obs = Ins_test_Y$PH)
```

```
##       RMSE   Rsquared        MAE
## 0.12524172 0.43956631 0.09828361
```

The Basic Regression Tree R-Squared value is 0.459, meaning that the model explains 46% of variability in the data. RMSE for Basic Regression Tree is 0.129.

Cubist

Next, we will try a Cubist Model. Cubist is a rule–based model. A tree is grown where the terminal leaves contain linear regression models. These models are based on the predictors used in previous splits. Also, there are intermediate linear models at each step of the tree.

```
suppressWarnings(library(Cubist))
set.seed(333)

cubistMod <- cubist(Ins_train_X,
                    Ins_train_Y$PH,
                    committees = 6
)
```

```
## Warning in cubist.default(Ins_train_X, Ins_train_Y$PH, committees = 6): NAs
## introduced by coercion
```

```
cubistModPred <- predict(cubistMod, newdata = Ins_test_X)
postResample(pred = cubistModPred, obs = Ins_test_Y$PH)
```

```
##       RMSE   Rsquared        MAE
## 0.11080863 0.59902441 0.07735901
```

The Cubist R-Squared value is 0.671, meaning that the model explains 67% of variability in the data. RMSE for Cubist is 0.101.

Bagged Trees

Finally, we will try Bagged Trees Model. Bagging effectively reduces the variance of a prediction through its aggregation process.

```
set.seed(333)
suppressWarnings(library(ipred))

baggedTree <- ipredbagg(Ins_train_Y$PH, Ins_train_X)

baggedTreePred <- predict(baggedTree, newdata = Ins_test_X)
postResample(pred = baggedTreePred, obs = Ins_test_Y$PH)
```

```
##       RMSE   Rsquared        MAE
## 0.11717009 0.51039455 0.09120786
```

The Bagged R-Squared value is 0.523, meaning that the model explains 53% of variability in the data. RMSE for Bagged Tree is 0.122.

Comparing Tree Models

It looks like Tree Models are performing better than non-linear models and linear models based on their R-squared values, but let's compare those and see which model performs best.

```
z<- rbind(
  postResample(pred = RFPred, obs = Ins_test_Y$PH),
  postResample(pred = GBM_Pred, obs = Ins_test_Y$PH),
  postResample(pred = ST_Pred, obs = Ins_test_Y$PH),
  postResample(pred = cubistModPred, obs = Ins_test_Y$PH),
  postResample(pred = baggedTreePred, obs = Ins_test_Y$PH)
)

data.frame(z,row.names = c('Random Forrest', 'Boosted Trees', 'Single Tree', 'Cubist', 'Bagged Tree'))
```

```
##                      RMSE  Rsquared        MAE
## Random Forrest 0.09718891 0.6821920 0.07064440
## Boosted Trees  0.10310588 0.6198424 0.07703696
## Single Tree    0.12524172 0.4395663 0.09828361
## Cubist         0.11080863 0.5990244 0.07735901
## Bagged Tree    0.11717009 0.5103946 0.09120786
```

Based on the combination of R-Squared and RMSE values for all models we tried - the best Model is Cubist - that's what we will use for our predictions. Random Forest model also has vevry good performance compared to all the other models we tuned and evaluated. Overall, Tree models are performing better that Linear and other Non-Linear Models based on RMSE and R-Squared values.

Here is the list of most relevant variables in this Cubist model:

```
varImp(cubistMod)
```

```
##                   Overall
## Mnf.Flow             81.5
## Brand.Code           27.5
## Pressure.Vacuum      53.0
## Balling.Lvl          53.5
## Bowl.Setpoint        31.5
## Oxygen.Filler        38.0
## Filler.Speed         31.5
## Hyd.Pressure3        40.0
## Carb.Rel             34.0
## Alch.Rel             48.0
## Usage.cont           24.5
## Balling              43.5
## Density              47.5
## Carb.Pressure1       35.0
## Air.Pressurer         4.5
## Carb.Flow            23.0
## Filler.Level         20.5
## MFR                  16.5
## PC.Volume            11.0
## Hyd.Pressure2        24.0
## Hyd.Pressure1        20.5
## Hyd.Pressure4         6.0
## Temperature          31.0
## Carb.Volume          10.5
## Carb.Pressure        19.0
## Carb.Temp            16.5
## Pressure.Setpoint    13.5
## Fill.Pressure         9.5
## PSC.Fill              7.0
## Fill.Ounces           6.5
## PSC                   3.0
## PSC.CO2               1.0
```

Predictions

Now that we have identified the best model, we can use our evaluation data to make PH predictions and output predictions to an excel readable format. We are adding predicted PH values to our Evaluation data set.

```
final_predictions <- predict(cubistMod, newdata=Ins_eval_cap_imputed)
Ins_eval_cap_imputed$PH <- final_predictions
final_predictions_df <- data.frame(Ins_eval_cap_imputed)
head(final_predictions_df)
```

```
##   Brand.Code Carb.Volume Fill.Ounces PC.Volume Carb.Pressure Carb.Temp    PSC
## 1          D    5.480000    24.03333 0.2700000          65.4     134.6 0.1934
## 2          A    5.393333    23.95333 0.2266667          63.2     135.0 0.0420
## 3          B    5.293333    23.92000 0.3033333          66.4     140.4 0.0680
## 4          B    5.266667    23.94000 0.1860000          64.8     139.0 0.0040
## 5          B    5.406667    24.09333 0.1600000          69.4     142.2 0.0400
## 6          B    5.286667    24.10667 0.2120000          73.4     147.2 0.0780
##   PSC.Fill PSC.CO2 Mnf.Flow Carb.Pressure1 Fill.Pressure Hyd.Pressure1
## 1     0.40    0.04     -100          116.6          46.0             0
## 2     0.22    0.08     -100          118.8          46.2             0
## 3     0.10    0.02     -100          120.2          45.8             0
## 4     0.20    0.02     -100          124.8          40.0             0
## 5     0.30    0.06     -100          115.0          51.4             0
## 6     0.22    0.04     -100          118.6          46.4             0
##   Hyd.Pressure2 Hyd.Pressure3 Hyd.Pressure4 Filler.Level Filler.Speed
## 1             0             0          96.0        129.4    7939410.7
## 2             0             0         112.0        120.0    8043319.4
## 3             0             0          98.0        119.4    8035302.4
## 4             0             0         125.4        120.2     509801.6
## 5             0             0          94.0        116.0    8067394.3
## 6             0             0          94.0        120.4    8035302.4
##   Temperature Usage.cont Carb.Flow Density       MFR Balling Pressure.Vacuum
## 1        66.0      21.66      2950    0.88 264578.30   1.398            -3.8
## 2        65.6      17.60      2916    1.50 270575.24   2.942            -4.4
## 3        65.6      24.18      3056    0.90 269840.31   1.448            -4.2
## 4        69.0      18.12        28    0.74  26994.26   1.056            -4.0
## 5        66.4      21.32      3214    0.88 282620.39   1.398            -4.0
## 6        66.6      18.00      3064    0.84 267787.82   1.298            -3.8
##         PH Oxygen.Filler Bowl.Setpoint Pressure.Setpoint Air.Pressurer Alch.Rel
## 1 8.591923     -3.736578      8446.705              45.2     0.9930600     6.56
## 2 8.892403     -3.438840      7197.162              46.0     0.9932421     7.14
## 3 8.704253     -3.026817      7197.162              46.0     0.9932421     6.52
## 4 8.788146     -2.443021      7197.162              46.0     0.9932421     6.48
## 5 8.383476     -2.466458      7197.162              50.0     0.9932421     6.50
## 6 8.699192     -2.707141      7197.162              46.0     0.9932421     6.50
##   Carb.Rel Balling.Lvl
## 1     5.34        1.48
## 2     5.58        3.04
## 3     5.34        1.46
## 4     5.50        1.48
## 5     5.38        1.46
## 6     5.42        1.44
```

```
write_csv(final_predictions_df, "PH_Result.csv")
```