FALL 2019: DATA 624 PROJECT 2

Group 1

Angrand, Burke, Deboch, Groysman, Karr

Contents

Project2_Group1_Data624

Angrand, Burke, Deboch, Groysman, Karr

December 10, 2019

Data 624 Project 2

Prompt

You are given a simple data set from a beverage manufacturing company. It consists of 2,571 rows/cases of data and 33 columns / variables. **Your goal is to use this data to predict PH (a column in the set).** PH is a measure of acidity/alkalinity, it must conform in a critical range and therefore it is important to understand its influence and predict its values.This is production data. PH is a KPI, Key Performance Indicator. You are also given a scoring set (267 cases). All variables other than the dependent or target. You will use this data to score your model with your best predictions.

Provided Files

- Data Dictionary.xlsx *Provides a listing of the columns and their underlying data components*
- StudentData.xlsx *The training dataset for the exercise*
- StudentEvaluation- TO PREDICT.xlsx *The evaluation dataset for the exercise*

Required Packages

```
#Upload library
library(readxl)
library(dplyr)
library(tidyr)
library(stringr)
library(ggplot2)
library(ggcorrplot)
library(reshape2)
library(caret)
library(mice)
library(fastDummies)
library(MASS)
library(randomForest)
```

1. Read in Data

- Download from the group's github repository link to ensure stakeholders can reproduce easily.

- The file is downloaded from git to the user's default downloads location and read in with the read_excel function from the readxl module
- Print the dimensions to ensure that the data is consistent with promt specifications
- Train Data: 2,571 observations, 33 predictors
- Eval Data: 267 observations, 33 predictors

a) Training Dataset - Student Data

```
train.loc <- tempfile(fileext = ".xlsx")
train.dataURL <-
"https://raw.githubusercontent.com/mburke65/CUNY_Data624/master/Project2Folde
r/ProvidedFiles/StudentData.xlsx"
download.file(train.dataURL, destfile= train.loc, mode='wb')


train.data <- readxl::read_excel(train.loc,sheet =  1,  col_names =TRUE)
print(paste("Dimensions of the train.data:", list(dim(train.data))))

## [1] "Dimensions of the train.data: c(2571, 33)"
```

b) Evaluation Dataset - StudentEvaluation- TO PREDICT.xlsx

```
eval.loc <- tempfile(fileext = ".xlsx")
eval.dataURL <-
"https://raw.githubusercontent.com/mburke65/CUNY_Data624/master/Project2Folde
r/ProvidedFiles/StudentEvaluation-%20TO%20PREDICT.xlsx"
download.file(eval.dataURL, destfile= eval.loc, mode='wb')

eval.data <- read_excel(eval.loc,sheet =  1,  col_names =TRUE)
print(paste("Dimensions of the eval.data:", list(dim(eval.data))))

## [1] "Dimensions of the eval.data: c(267, 33)"
```

2. Exploratory Data Analysis

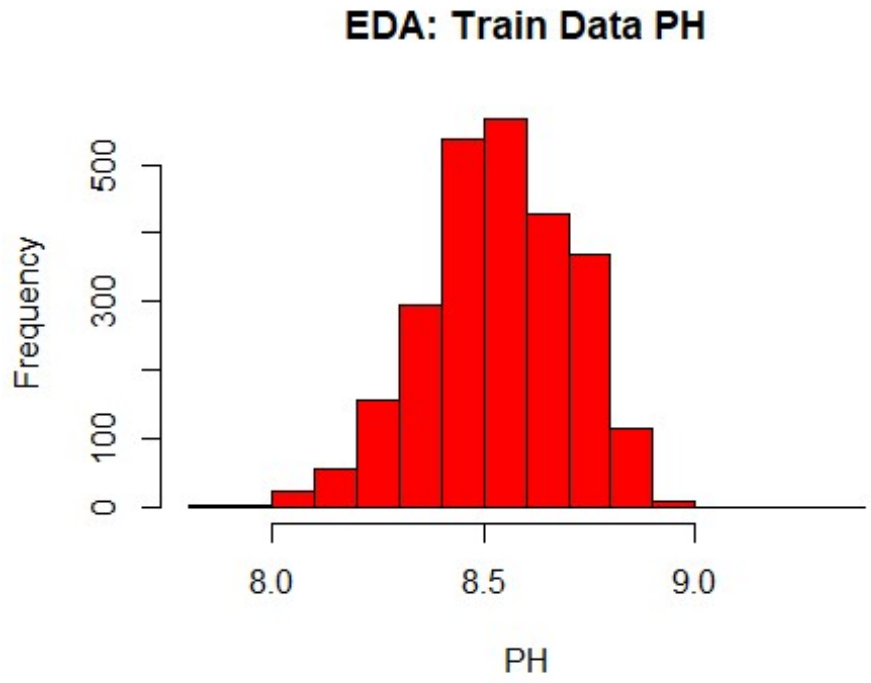
a. Explore PH
b. Identify the predictors (column names)
c. For numerical predictors (!= "Brand Code"), explore predictors summary statistics. The summary statistics provide a quick and simple description of the data which helps us have a better understanding of the data. The summary function in R provides the mean, median, number of nulls, min and max.
d. For categorical predictors (== "Brand Code"), explore the frequency of brands
e. Graphically highlight nulls
f. Investigate the correlation between predictors
g. histogram of all the predictors to better understand the shape and spread of the data

a. Explore PH

PH Definition In chemistry,PH is a scale used to specify how acidic or basic a water-based solution is. Acidic solutions have a lower pH, while basic solutions have a higher pH. At room temperature (25°C or 77°F), pure water is neither acidic nor basic and has a pH of 7.The pH value can be less than 0 for very strong acids, or greater than 14 for very strong bases.

- In the provided train.data, approximately 85% of the PH values were between 8.3 and 8.8. For reference, most of these observation would fall between PH values similar to sea water and baking soda, slightly more basic than water. The group expects that the predicted values will track against this range of values.
- Based on the distribution below, the data looks mostly normal

```
hist.ph<- hist(train.data$PH,
    main = 'EDA: Train Data PH',
    xlab = 'PH',
    col = 10)
```




```
breaks <- hist.ph$breaks

value_list <- c()
i <- 1
for (val in breaks){
  if(which(breaks == val) != 1){
    value_list[[i]] <- paste0(breaks[which(breaks == val)-1]," to
",breaks[which(breaks == val)] )
```

```
    i <- i +1
  }
}
#create a dateframe from the breaks and the counts
ph_df<- data.frame(breaks= value_list, counts = hist.ph$counts)
ph_df$percentage<-round((ph_df$counts/ sum(ph_df$counts)) *100,2)
ph_df
```

```
##        breaks counts percentage
## 1  7.8 to 7.9      2       0.08
## 2    7.9 to 8      3       0.12
## 3    8 to 8.1     23       0.90
## 4  8.1 to 8.2     57       2.22
## 5  8.2 to 8.3    157       6.12
## 6  8.3 to 8.4    296      11.53
## 7  8.4 to 8.5    538      20.96
## 8  8.5 to 8.6    567      22.09
## 9  8.6 to 8.7    429      16.71
## 10 8.7 to 8.8    369      14.37
## 11 8.8 to 8.9    116       4.52
## 12   8.9 to 9      9       0.35
## 13   9 to 9.1      0       0.00
## 14 9.1 to 9.2      0       0.00
## 15 9.2 to 9.3      0       0.00
## 16 9.3 to 9.4      1       0.04
```

b. Identify Predictors

```
#predictors
names(train.data)
```

```
##  [1] "Brand Code"        "Carb Volume"       "Fill Ounces"
##  [4] "PC Volume"         "Carb Pressure"     "Carb Temp"
##  [7] "PSC"               "PSC Fill"          "PSC CO2"
## [10] "Mnf Flow"          "Carb Pressure1"    "Fill Pressure"
## [13] "Hyd Pressure1"     "Hyd Pressure2"     "Hyd Pressure3"
## [16] "Hyd Pressure4"     "Filler Level"      "Filler Speed"
## [19] "Temperature"       "Usage cont"        "Carb Flow"
## [22] "Density"           "MFR"               "Balling"
## [25] "Pressure Vacuum"   "PH"                "Oxygen Filler"
## [28] "Bowl Setpoint"     "Pressure Setpoint" "Air Pressurer"
## [31] "Alch Rel"          "Carb Rel"          "Balling Lvl"
```

c. Summary Statistics Table

- Initial Findings: MFR and Brand Code have a significant percentage of null values and the scaling/range of each variable varies

```
#get the null values

#select(-c(Var1, Value))%>%
#%>%select(-c("1st Qu.","3rd Qu.","Class", "Length" , "NA's" ,'Mode' ))
null.values <-as.data.frame(sapply(train.data, function(x) sum(is.na(x))),
```

```
col.names = "null_values")%>%
  tibble::rownames_to_column("Predictors")%>%
  rename(nulls = 2)%>%
  mutate(Percentage_Missing = round((nulls/2571)*100,2))

#get the summary stats, restructure into a readable dataframe. merge on the
null.values
summary.stats <- as.data.frame(summary(train.data))%>%
  na.omit() %>%
  separate(Freq, c("Summary.Stat","Value"), sep = ":")%>%
  mutate(Var2 = as.character(Var2))%>%
  mutate_if(is.character, str_trim)%>%
  filter(Value !="character  " )%>%
  mutate(Value.Num = factor(Value))%>%
  dplyr::select(-Var1, -Value)%>%

  rename(Predictors = Var2)%>%
  spread(Summary.Stat, Value.Num)%>%
  dplyr::select(-c("1st Qu.","3rd Qu.","Class", "Length" , "NA's" ,'Mode'
))%>%
  left_join(null.values, by = "Predictors")%>%
  arrange(-nulls)


summary.stats
```

```
##           Predictors    Max.    Mean  Median    Min. nulls
## 1                MFR   868.6   704.0   724.0    31.4   212
## 2         Brand Code    <NA>    <NA>    <NA>    <NA>   120
## 3       Filler Speed    4030    3687    3982     998    57
## 4          PC Volume 0.47800 0.27712 0.27133 0.07933    39
## 5            PSC CO2 0.24000 0.05641 0.04000 0.00000    39
## 6        Fill Ounces   24.32   23.97   23.97   23.63    38
## 7                PSC 0.27000 0.08457 0.07600 0.00200    33
## 8     Carb Pressure1   140.2   122.6   123.2   105.6    32
## 9      Hyd Pressure4  142.00   96.29   96.00   52.00    30
## 10     Carb Pressure   79.40   68.19   68.20   57.00    27
## 11         Carb Temp   154.0   141.1   140.8   128.6    26
## 12          PSC Fill  0.6200  0.1954  0.1800  0.0000    23
## 13     Fill Pressure   60.40   47.92   46.40   34.60    22
## 14      Filler Level   161.2   109.3   118.4    55.8    20
## 15     Hyd Pressure2   59.40   20.96   28.60    0.00    15
## 16     Hyd Pressure3   50.00   20.46   27.60   -1.20    15
## 17       Temperature   76.20   65.97   65.60   63.60    14
## 18     Oxygen Filler 0.40000 0.04684 0.03340 0.00240    12
## 19 Pressure Setpoint   52.00   47.62   46.00   44.00    12
## 20     Hyd Pressure1   58.00   12.44   11.40   -0.80    11
## 21          Carb Rel   6.060   5.437   5.400   4.960    10
## 22       Carb Volume   5.700   5.370   5.347   5.040    10
## 23          Alch Rel   8.620   6.897   6.560   5.280     9
```

```
## 24        Usage cont   25.90   20.99   21.79   12.08     5
## 25                PH   9.360   8.546   8.540   7.880     4
## 26     Bowl Setpoint   140.0   109.3   120.0    70.0     2
## 27         Carb Flow    5104    2468    3028      26     2
## 28          Mnf Flow  229.40   24.57   65.20 -100.20     2
## 29           Balling   4.012   2.198   1.648  -0.170     1
## 30       Balling Lvl    3.66    2.05    1.48    0.00     1
## 31           Density   1.920   1.174   0.980   0.240     1
## 32     Air Pressurer   148.2   142.8   142.6   140.8     0
## 33   Pressure Vacuum  -3.600  -5.216  -5.400  -6.600     0
##    Percentage_Missing
## 1                8.25
## 2                4.67
## 3                2.22
## 4                1.52
## 5                1.52
## 6                1.48
## 7                1.28
## 8                1.24
## 9                1.17
## 10               1.05
## 11               1.01
## 12               0.89
## 13               0.86
## 14               0.78
## 15               0.58
## 16               0.58
## 17               0.54
## 18               0.47
## 19               0.47
## 20               0.43
## 21               0.39
## 22               0.39
## 23               0.35
## 24               0.19
## 25               0.16
## 26               0.08
## 27               0.08
## 28               0.08
## 29               0.04
## 30               0.04
## 31               0.04
## 32               0.00
## 33               0.00
```

d. Categorical Frequency - Brand Code

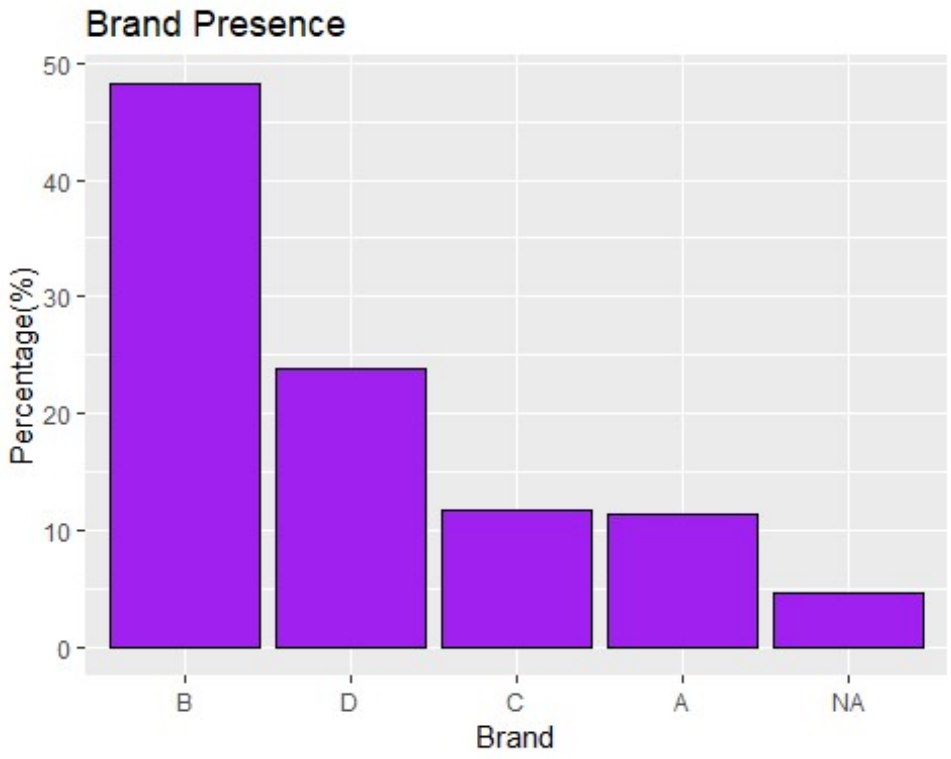
- Brand Code B is the most popular brand produced by the factory by a significant margin

- Approximately 5% of the Brands dropped identification (null), future null analysis in (d)
- Being the one categorical variable, the group will need to convert brand code to dummy variables in the modeling phase

```
brand <- train.data%>%
  rename(Brand_Code = 1)%>%
  group_by(Brand_Code)%>% summarise(n = n())%>%
  arrange(-n)%>%
  mutate(Presence_Precentage = round((n/sum(n))*100,1))
brand
```

```
## # A tibble: 5 x 3
##   Brand_Code     n Presence_Precentage
##   <chr>      <int>               <dbl>
## 1 B           1239                48.2
## 2 D            615                23.9
## 3 C            304                11.8
## 4 A            293                11.4
## 5 <NA>         120                 4.7
```

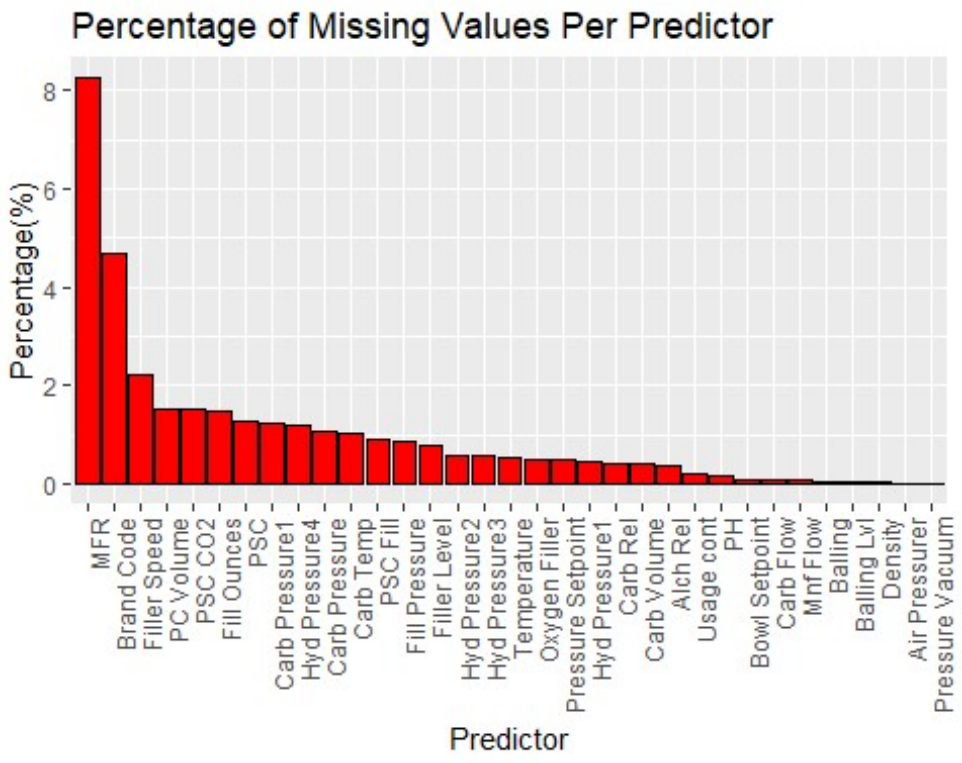
```
brand%>%

  ggplot(aes(x= reorder(Brand_Code,-Presence_Precentage), y =
Presence_Precentage))+
  geom_bar(stat = "identity", fill = "purple", color = "black")+
  ggtitle("Brand Presence")+
  xlab("Brand")+
  ylab("Percentage(%)")
```



Null Display - All Predictors

- This graph displays the missing values across diverse predictors. In subsequent sections, the group will determine an approach to deal with missing values (i.e. random generation of values, mean replacement or k-neighbor, mice, etc.)

```
summary.stats%>%

  ggplot(aes(x= reorder(Predictors,-Percentage_Missing), y =
Percentage_Missing ))+
  geom_bar(stat = "identity",fill = "red", color = "black")+
  theme(axis.text.x=element_text(angle=90, hjust=1))+
  ggtitle("Percentage of Missing Values Per Predictor")+
  xlab("Predictor")+
  ylab("Percentage(%)")
```

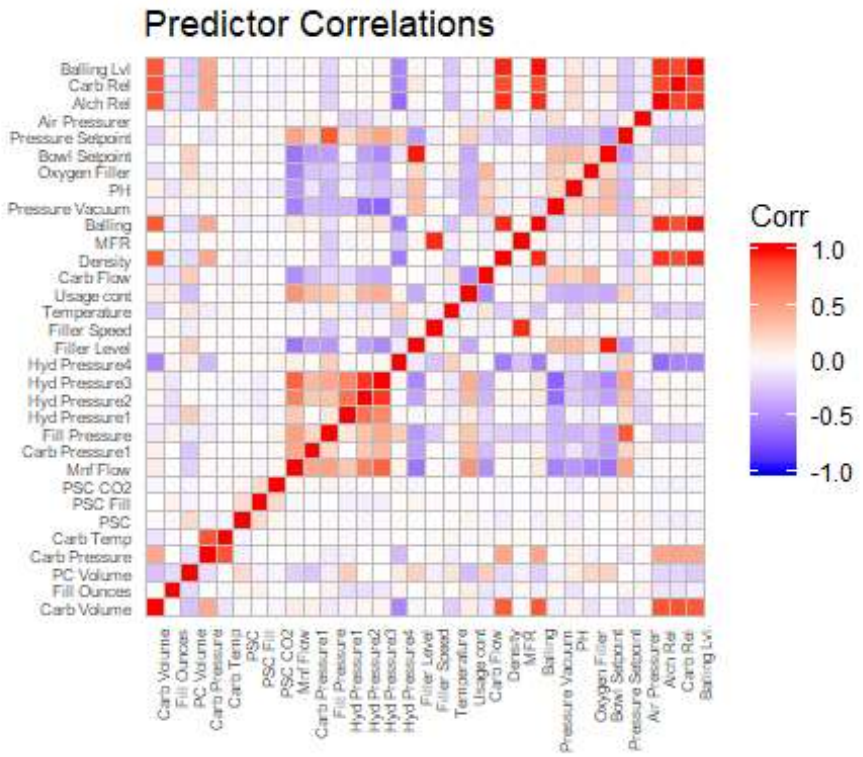


e) Data Correlations Display - All Predictors

- Compute the correlation matrix using the cor() function. Override the "use" parameter, a optional parameter method for computing covariances in the presence of missing values which the PH dataset has.
- Display the correlation matrix ggcorrplot()
- Extract the unique predictor combinations with the highest correlation values (>= .85)
- Balling Lvl is highly correlated to Balling, Density, Alch Rel, Carb Rel
- Carb Rel is highly correlated to Alch Rel, Balling, Density
- Bowl Setpoint is highly correlated to Filler Level
- Balling is highly correlated to Density
- MFR is highly correlated to Filler Speed
- Hyd Pressure3 is highly correlated to Hyd Pressure2
- In subsequent steps, the group will likely remove the highly correlated variables to avoid multicollinearity in our modeling
- use the findCorrelation() with the .85 threshold to identify which predictors should be removed from the subset previously identified. The function finds absolute values of pair-wise correlations are considered. If two variables have a high correlation, the function looks at the mean absolute correlation of each variable and removes the variable with the largest mean absolute correlation.

```
cor.matrix <-  cor(train.data[,-1], use = "na.or.complete")
ggcorrplot(cor.matrix)+
  theme(axis.text.x=element_text(size=rel(.7), angle=90, hjust=1),
```

```
        axis.text.y = element_text(size=rel(.7), hjust=1))+
  ggtitle("Predictor Correlations")
```




```
#reshape the correlation matrix and identify the correlation pairs above 85%
reshape2::melt(cor.matrix)%>%
  rename(Predictor1 = Var1, Predictor2 = Var2, CorrelationValue = value)%>%
  filter(CorrelationValue != 1)%>%
  arrange(-CorrelationValue) %>%
  filter(CorrelationValue >= 0.85)%>%
 filter (! duplicated(CorrelationValue))%>%
  arrange(Predictor1, -CorrelationValue)
```

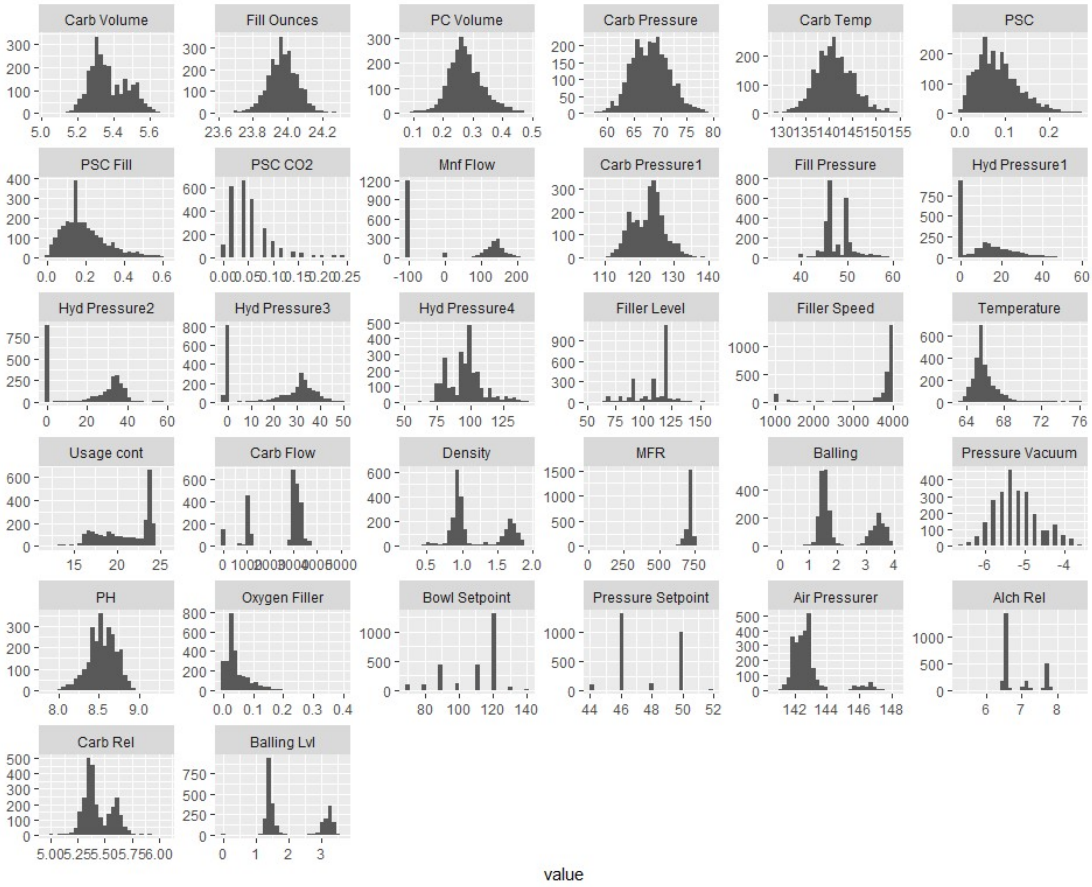
```
##       Predictor1    Predictor2 CorrelationValue
## 1  Hyd Pressure3 Hyd Pressure2        0.9176010
## 2            MFR  Filler Speed        0.9514224
## 3        Balling       Density        0.9523125
## 4  Bowl Setpoint  Filler Level        0.9773811
## 5       Alch Rel       Balling        0.9412251
## 6       Alch Rel       Density        0.9157798
## 7       Carb Rel      Alch Rel        0.8768039
## 8       Carb Rel       Balling        0.8542582
## 9       Carb Rel       Density        0.8526890
## 10   Balling Lvl       Balling        0.9876727
## 11   Balling Lvl       Density        0.9550900
## 12   Balling Lvl      Alch Rel        0.9429801
## 13   Balling Lvl      Carb Rel        0.8682872
```

```
#identify which columns will later be removed
remove.cor.cols <- findCorrelation(cor.matrix, cutoff= .85,verbose = TRUE,
                                   names = TRUE)

## Compare row 23  and column  30 with corr  0.941
##   Means:  0.251 vs 0.158 so flagging column 23
## Compare row 14  and column  13 with corr  0.918
##   Means:  0.245 vs 0.152 so flagging column 14
## Compare row 30  and column  32 with corr  0.943
##   Means:  0.219 vs 0.145 so flagging column 30
## Compare row 32  and column  31 with corr  0.868
##   Means:  0.191 vs 0.141 so flagging column 32
## Compare row 31  and column  21 with corr  0.853
##   Means:  0.169 vs 0.139 so flagging column 31
## Compare row 16  and column  27 with corr  0.977
##   Means:  0.209 vs 0.134 so flagging column 16
## Compare row 22  and column  17 with corr  0.951
##   Means:  0.089 vs 0.133 so flagging column 17
## All correlations <= 0.85

remove.cor.cols

## [1] "Balling"       "Hyd Pressure3" "Alch Rel"      "Balling Lvl"
## [5] "Carb Rel"      "Filler Level"  "Filler Speed"
```

f.1) Scatter Plots - All Predictors

- Convert the train.data from wide to long foramt
- Use qplot to generate histograms for each variable
- Initial Findings (NORMAL?): Fill Ounces, PC Volume, Carb Pressure, Carb Temp, Carb Pressure1, PH follow somewhat normal distributions
- Initial Findings (CLUSTER?): Mnf Flow, Hyd Pressure1, Hyd Pressure2, Hyd Pressure3, Filler Speed, and Carb flow values appear more clustered and may need to be examined more closely
- Initial Findings (CATEGORICAL?): Pressure Setpoint, Bowl Setpoint appear to be more categorical as there is very little variety/distribution in the resulting data

```
melted.train <- melt(train.data[,-1])
qplot(value, data=melted.train) + facet_wrap(~variable, scales="free")

## Warning: Removed 724 rows containing non-finite values (stat_bin).
```



f.2) Scatter Plots For Identified Clustered Predictors

- Use plot() to test the PH levels based on each predictor identified to assess if any values are problematic and should be removed in the modeling step
- Identified Clusters: Mnf Flow, Hyd Pressure1, Hyd Pressure2, and Hyd Pressure3
- Find the counts at each bin to assess if the distribution makes sense

Function to generate the scatter plot of PH vs. the inputted predictor

```
scatter_analysis <- function(predictor) {
  subset<- train.data %>%
  dplyr::select(c(predictor, "PH"))%>%
  filter(complete.cases(predictor, "PH"))

  plotgraphic<- plot(subset, aes(PH, predictor))+
    title(paste0("PH Levels Based on: ", predictor))

  return(plotgraphic)
}
```

Function to generate the counts at each bin

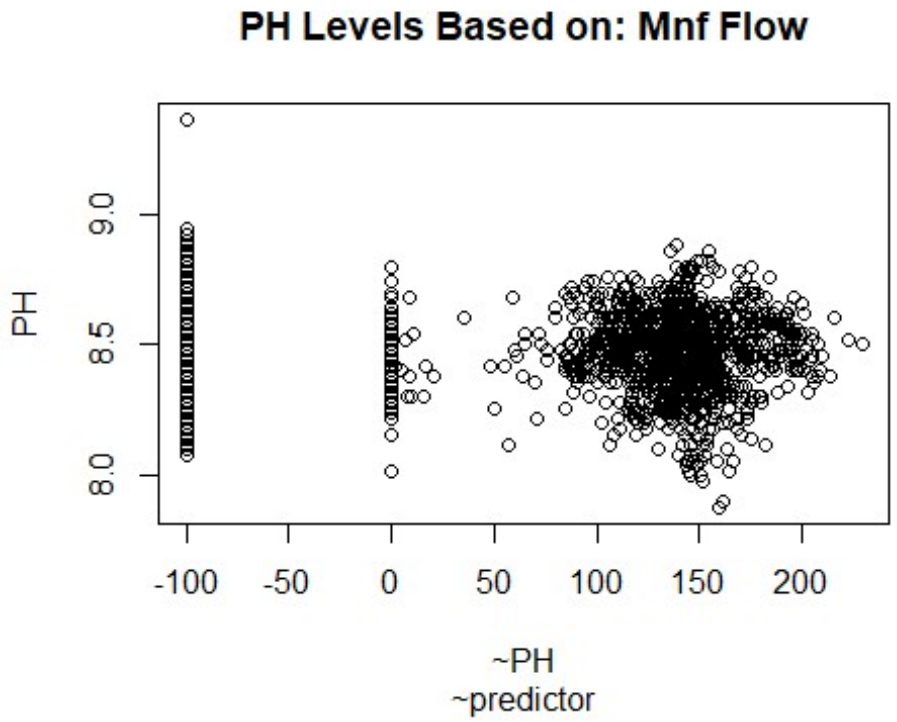
```
distribution_cluster <- function(predictor) {
  test <- melted.train%>%
```

```
    filter(variable ==predictor )%>%
    dplyr::select(c(value))

  histrv<- hist(test$value)
  breaks <-   histrv$breaks

  #loop through the breaks to generate the string range
  value_list <- c()
  i <- 1
  for (val in breaks){
    if(which(breaks == val) != 1){
      value_list[[i]] <- paste0(breaks[which(breaks == val)-1]," to
",breaks[which(breaks == val)] )
      i <- i +1
    }
  }
#create a dateframe from the breaks and the counts
return_df<- data.frame(breaks= value_list, counts = histrv$counts)
#add the percentage at each break point
return_df <- return_df %>%
            mutate(Percentage_Break = round((counts/sum(counts))*100,2))
return(return_df)
}
```

f.2) Mnf Flow

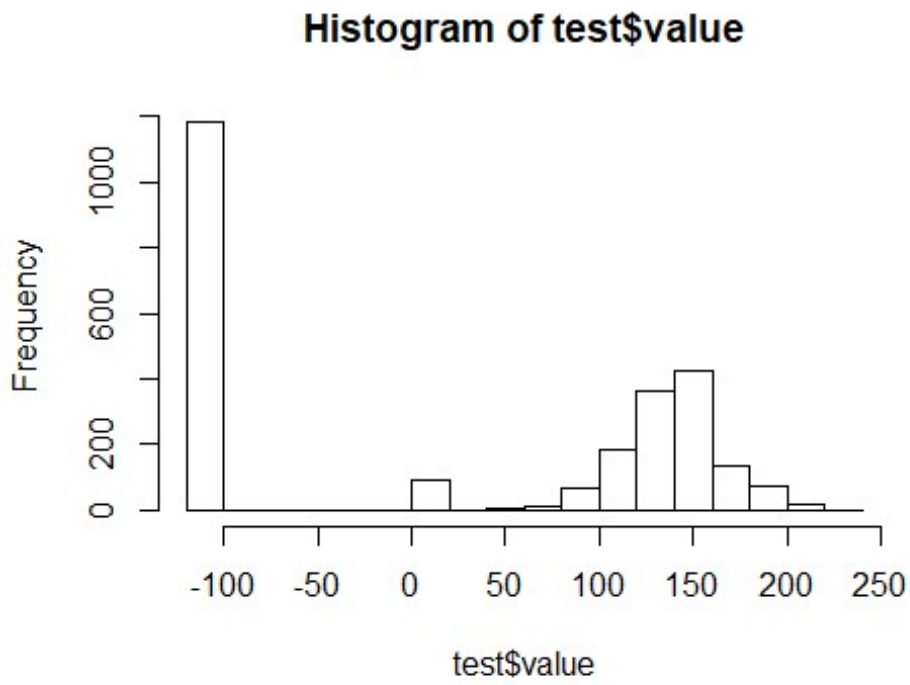
- The group selected MnF Flow to further analyze because the histogram presented odd behavior with a sizable concentration of negative 100 values when the rest of the values are positive and are concentrated around 100 to 160. It is likely that these values were misattributed with a negative value or it is a placeholder for null values.
- Using the distribution_cluster function, it was discovered that the negative 100 concentration makes up approximately 46.1% of the observations. Given the high concentration of seemingly misattributed values, the group will not consider Mnf Flow as a model predictor.

```
scatter_analysis("Mnf Flow")
```




```
## integer(0)
```

```
distribution_cluster("Mnf Flow")
```



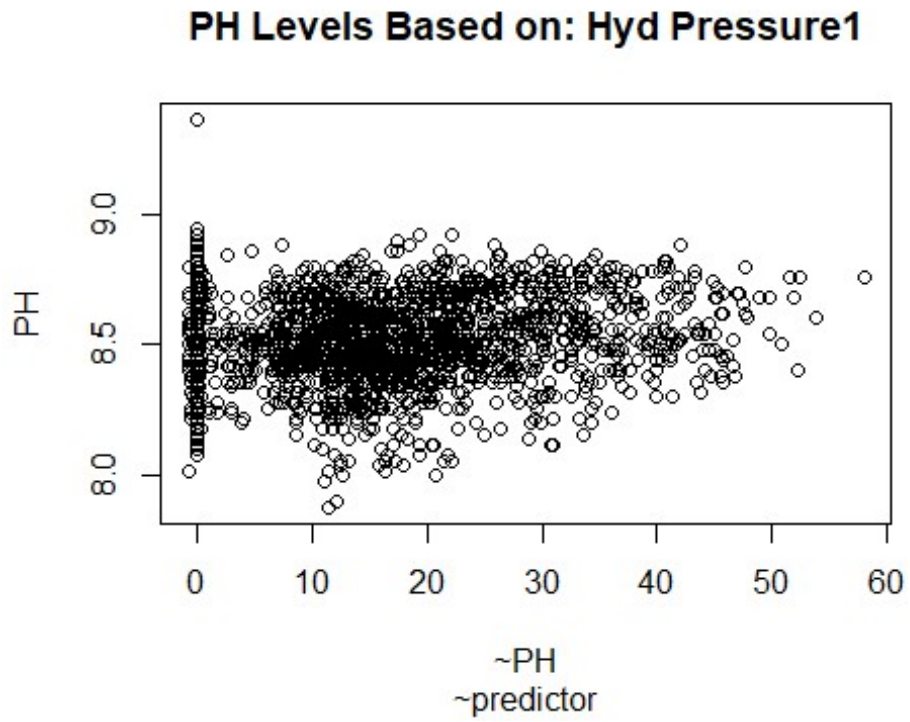
```
##          breaks counts Percentage_Break
## 1  -120 to -100   1184            46.09
## 2   -100 to -80      0             0.00
## 3    -80 to -60      0             0.00
## 4    -60 to -40      0             0.00
## 5    -40 to -20      0             0.00
## 6      -20 to 0      0             0.00
## 7       0 to 20     89             3.46
## 8      20 to 40      2             0.08
## 9      40 to 60      5             0.19
## 10     60 to 80     13             0.51
## 11    80 to 100     65             2.53
## 12   100 to 120    187             7.28
## 13   120 to 140    363            14.13
## 14   140 to 160    427            16.62
## 15   160 to 180    138             5.37
## 16   180 to 200     75             2.92
## 17   200 to 220     19             0.74
## 18   220 to 240      2             0.08
```

f.2) Hyd Pressure1, Hyd Pressure2, Hyd Pressure3

- Similar to Mnf Flow, Hyd Pressure1, Hyd Pressure2, and Hyd Pressure3 presented an odd distribution pattern with a relvatively high concentration of observations clustered around zero and the rest of the observations following a somewhat normal distributions around different ranges. It is the groups suspision that the high concentration of zeros are imputed NaN values.

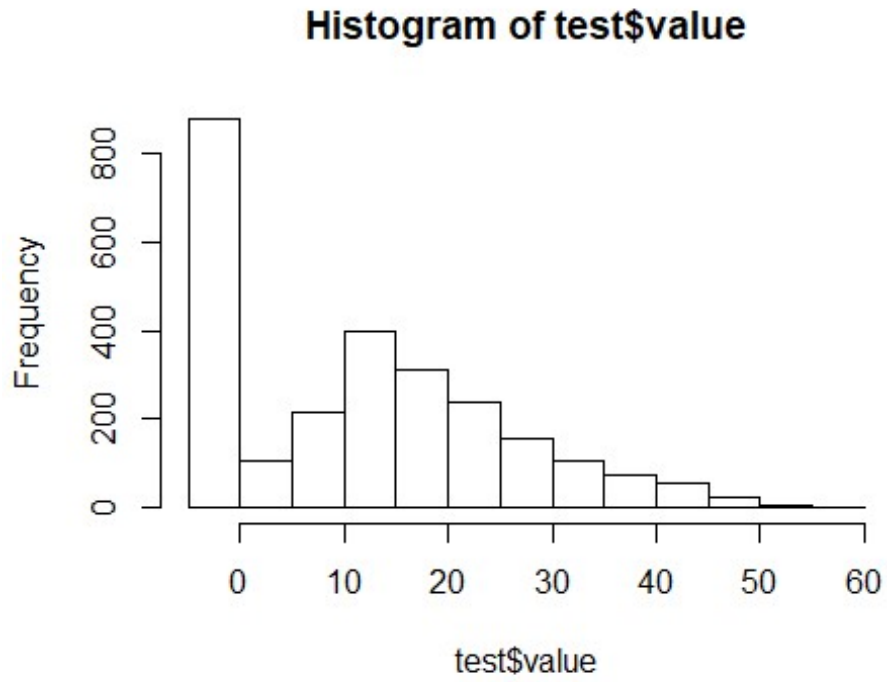
- The distribution_cluster() function notes that these zero observations make up approximately 34.3%, 35.1%, and 34.6% of the total observations for Hyd Pressure1, Hyd Pressure2, and Hyd Pressure3, respectively .Given the high concentration of seemingly misattributed values, the group will not consider Hyd Pressure1, Hyd Pressure2, Hyd Pressure3 as model predictors.

```
#Hyd Pressure1
scatter_analysis("Hyd Pressure1")
```



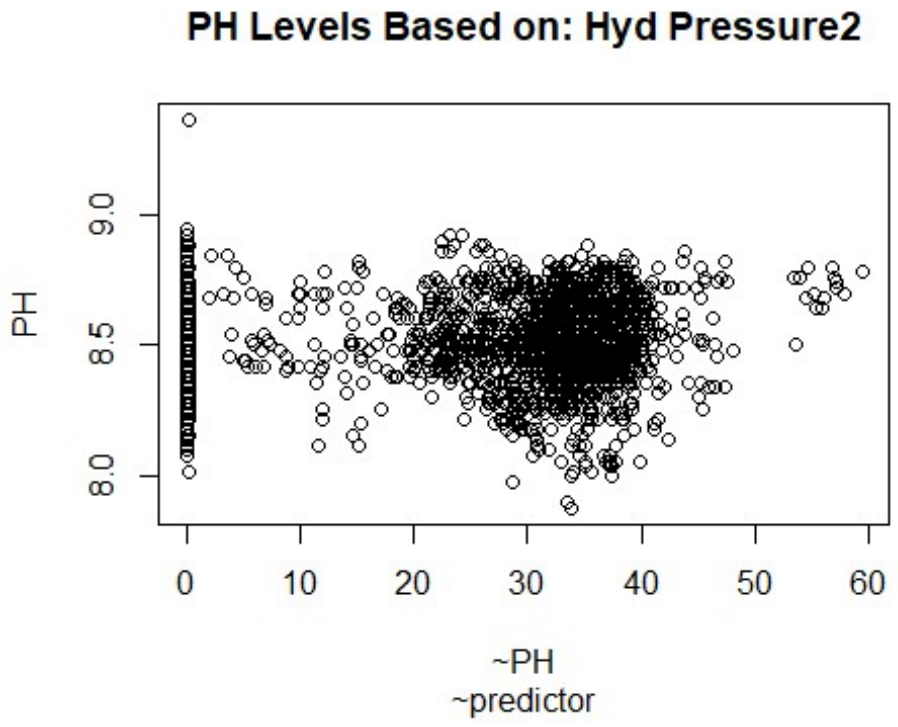

```
## integer(0)
```

```
distribution_cluster("Hyd Pressure1")
```



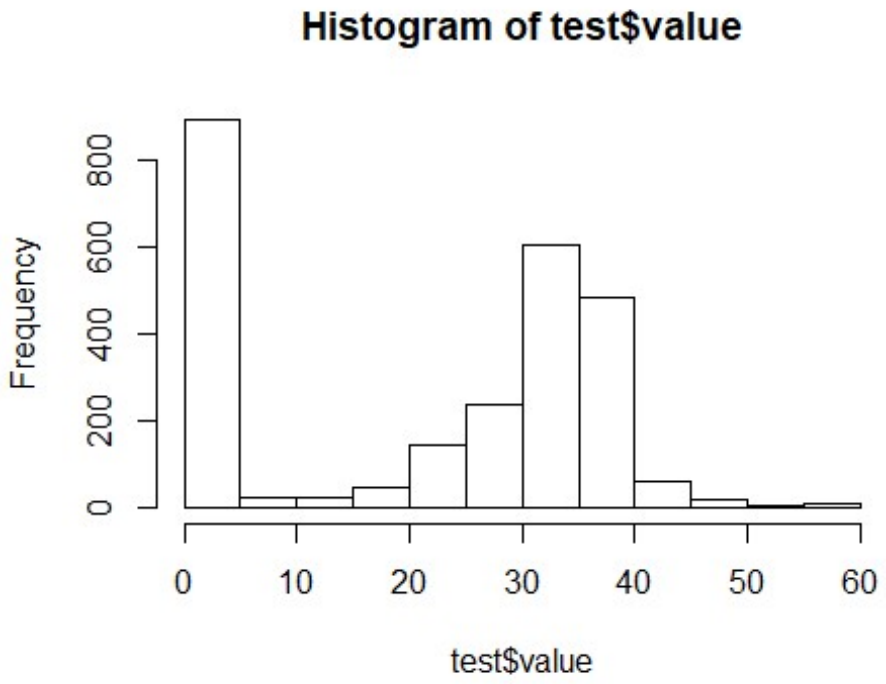
```
##       breaks counts Percentage_Break
## 1    -5 to 0    879            34.34
## 2     0 to 5    107             4.18
## 3    5 to 10    213             8.32
## 4   10 to 15    396            15.47
## 5   15 to 20    311            12.15
## 6   20 to 25    236             9.22
## 7   25 to 30    156             6.09
## 8   30 to 35    107             4.18
## 9   35 to 40     71             2.77
## 10 40 to 45     54             2.11
## 11 45 to 50     23             0.90
## 12 50 to 55      6             0.23
## 13 55 to 60      1             0.04
```

```
#Hyd Pressure2
scatter_analysis("Hyd Pressure2")
```



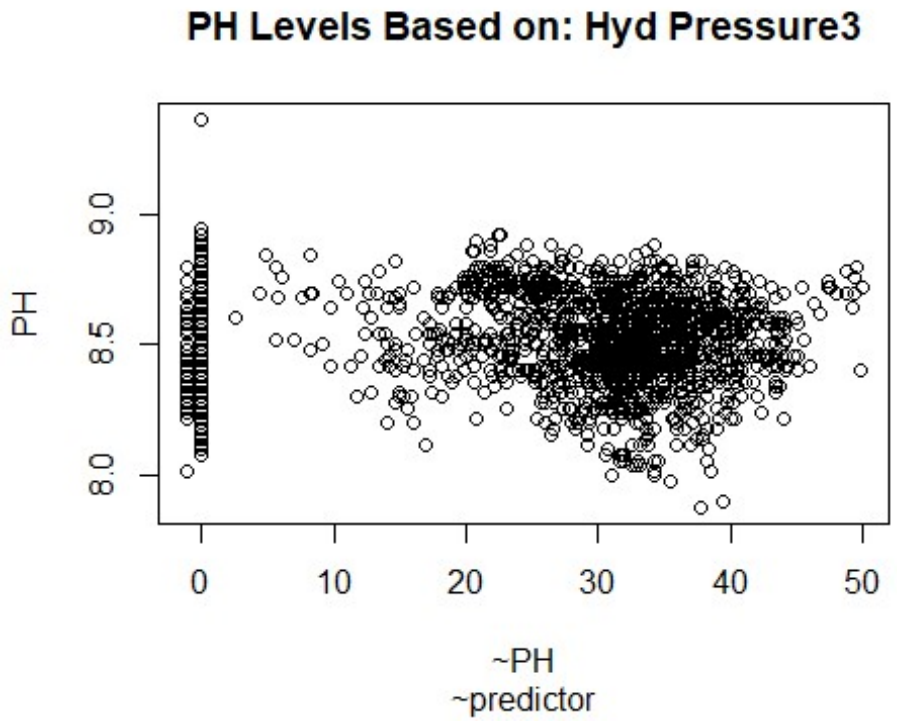

```
## integer(0)
```

```
distribution_cluster("Hyd Pressure2")
```



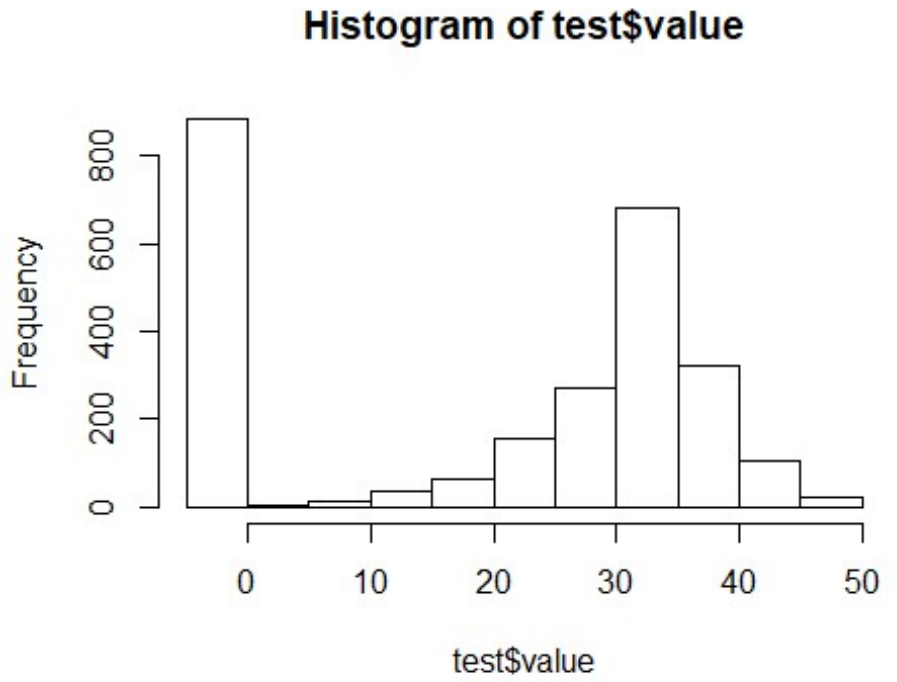

```
##       breaks counts Percentage_Break
## 1     0 to 5    896             35.05
## 2    5 to 10     24              0.94
## 3   10 to 15     25              0.98
## 4   15 to 20     45              1.76
## 5   20 to 25    142              5.56
## 6   25 to 30    239              9.35
## 7   30 to 35    604             23.63
## 8   35 to 40    483             18.90
## 9   40 to 45     62              2.43
## 10 45 to 50     20              0.78
## 11 50 to 55      5              0.20
## 12 55 to 60     11              0.43
```

```
#Hyd Pressure3
scatter_analysis("Hyd Pressure3")
```




```
## integer(0)
```

```
distribution_cluster("Hyd Pressure3")
```



```
##       breaks counts Percentage_Break
## 1   -5 to 0    884            34.59
## 2    0 to 5      4             0.16
## 3   5 to 10     13             0.51
## 4  10 to 15     35             1.37
## 5  15 to 20     63             2.46
## 6  20 to 25    157             6.14
## 7  25 to 30    269            10.52
## 8  30 to 35    683            26.72
## 9  35 to 40    320            12.52
## 10 40 to 45    107             4.19
## 11 45 to 50     21             0.82
```

3. Data Transformation

a. Remove problematic predictors
b. Impute null values
c. Create Dummy cols for the Brand Codes

a) Predictor Reduction

- remove the highly correlated predictors identified above and stored as "remove.cor.cols"
- remove the predictors with the problematic distribution noted above
- Following the removal of the identified predictors, there are 22 predictor columns to be considered

```
#full identified list between correlation and problematic clusters
remove.var <- c("Mnf Flow", "Hyd Pressure1", "Hyd Pressure2", "Hyd
Pressure3","Balling",  "Alch Rel", "Balling Lvl", "Carb Rel","Filler Level",
"Filler Speed")

#function to remove columns
remove <- function(dataframe, remove.var){
  return.df <- dataframe %>%
               dplyr::select(-remove.var)

  return(return.df)
}

#remove from train and test

train.sub<- remove(train.data,remove.var )
eval.sub<- remove(eval.data,remove.var )


print(paste("Dimensions of the train.sub:", list(dim(train.sub))))

## [1] "Dimensions of the train.sub: c(2571, 23)"

print(paste("Dimensions of the eval.sub:", list(dim(eval.sub))))
```

```
## [1] "Dimensions of the eval.sub: c(267, 23)"
```

b) Impute null values

- Fill in the missing Brand Code as "U" for unknown
- Since we have a limited amount of missing values across predictors, we will impute the data. We will use the mice package. The mice package implements a method to deal with missing data. The package creates multiple imputations (replacement values) for multivariate missing data. The method is based on Fully Conditional Specification, where each incomplete variable is imputed by a separate model. *Please note the mice package requires a transformation on col names as it does not accept spaces*
- The density plots below display the imputed density distribution in red

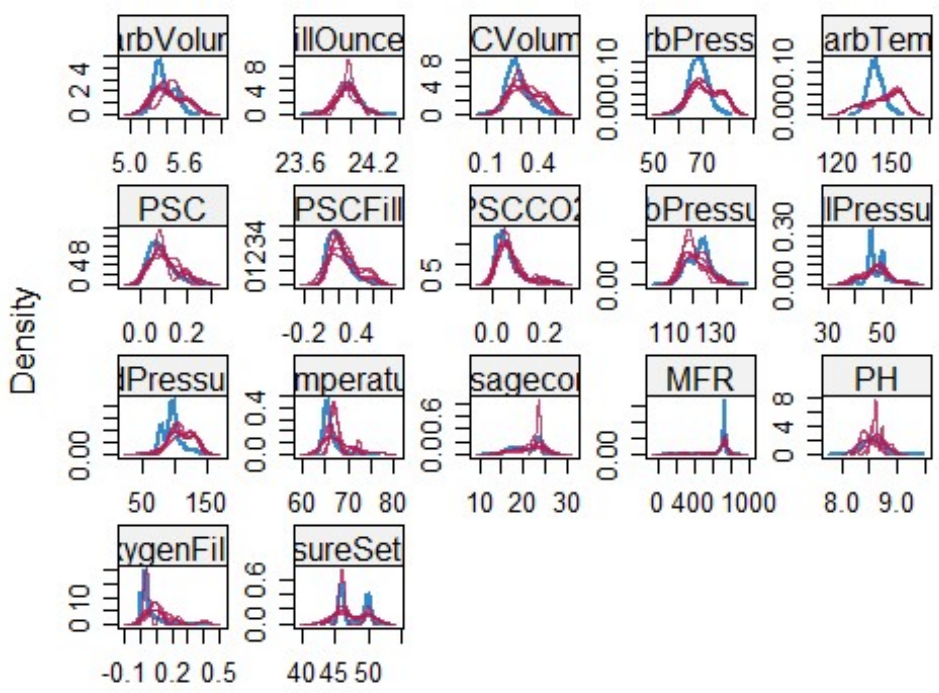
```
#fill in the Brand Code with "U" for unknown
train.sub$`Brand Code`[is.na(train.sub$`Brand Code`)]= "U"
eval.sub$`Brand Code`[is.na(eval.sub$`Brand Code`)]= "U"
train.sub%>%
  group_by(`Brand Code`)%>%
  summarise(n = n())
```

```
## # A tibble: 5 x 2
##   `Brand Code`     n
##   <chr>        <int>
## 1 A              293
## 2 B             1239
## 3 C              304
## 4 D              615
## 5 U              120
```

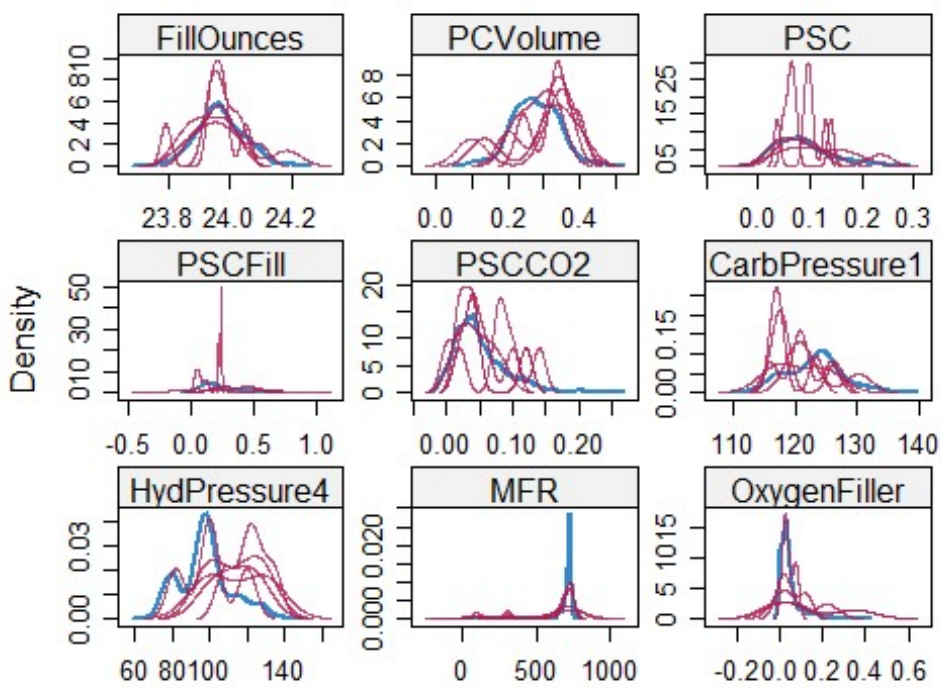
```
#mice impute function, call on eval and train data
impute_mice <- function(dataframe){
  #fix the column names so that the mice package can be used
  cols <- str_replace(colnames(dataframe), '\\s', '')
  colnames(dataframe) <- cols

  #call the mice function on the train data

  mic.imputes <- mice(dataframe, print = FALSE,seed = 123)



  return(mic.imputes)

}

#apply to train

train.imputed <- impute_mice(train.sub)
```

```
## Warning: Number of logged events: 1
```

```
densityplot(train.imputed)
```




```
train.imputed <- complete(train.imputed)

#apply to eval
eval.imputed <- impute_mice(eval.sub)

## Warning: Number of logged events: 2

densityplot(eval.imputed)
```




```
eval.imputed <- complete(eval.imputed)
```

c) Brand Code Dummy Columns

- Hot encode the Brand Code categorical variable into factors. This removes any issues with the modeling.

```
dummy_creation <- function(dataframe){
  df <- cbind(dataframe, dummy_cols(dataframe['BrandCode'])) %>%
        dplyr::select(-c('BrandCode','.data'))%>%
        rename(BrandB =`.data_B` , BrandA = `.data_A`, BrandC = `.data_C`,
BrandD =`.data_D`, BrandU =`.data_U` )

  return(df)


}
train.imputed.dum<- dummy_creation(train.imputed)
eval.imputed.dum<- dummy_creation(eval.imputed)
```

4. Modeling Building

- The group will explore and build various model to most accurately predict PH and identify the most significant variables that influence PH.

    a) Split the data into train/test
    b) Multiple Linear Regression:
    c) Stepwise - Both, Forward, Backward

d) Partial Least Square model
e) Random Forest
f) KNN

a) Create Train & Test Datasets

- The group split the data into train/test sets before running the models. This will allow us to make sure that predictions are consistent at all levels of PH. The train/test split is at 80/20% levels, respectively.

```
## set the seed to make the partition reproducible
set.seed(143)
# Procedure to create a train control data-set.
train.set <- createDataPartition(train.imputed.dum$PH, p = 0.80, list=FALSE)

train.df <- train.imputed.dum[train.set,]

test.df <- train.imputed.dum[-train.set,]


train.control <-trainControl(method = 'cv', number = 10,
  verboseIter = FALSE, savePredictions = TRUE,allowParallel = T)
```

b) Multiple Linear Regression

- The multiple linear regression model is an extension of simple linear regression used to predict an outcome variable (y) on the basis of multiple distinct predictor variables source. The caret package train() function was used to tune the model, parameter 'lm'.
- display the variable importance using the varImp Function

```
set.seed(143)
linear <- train(PH ~ ., data = train.df , metric = 'RMSE', method = 'lm',
trControl = train.control,
           tuneGrid  = expand.grid(intercept = FALSE))
linear$finalModel

##
## Call:
## lm(formula = .outcome ~ 0 + ., data = dat)
##
## Coefficients:
##       CarbVolume        FillOunces          PCVolume      CarbPressure
##       -2.281e-01        -8.263e-02        -1.813e-01         7.869e-03
##         CarbTemp               PSC           PSCFill            PSCCO2
##       -5.851e-03        -1.366e-01        -3.404e-02        -9.633e-02
##    CarbPressure1      FillPressure      HydPressure4       Temperature
##        7.077e-03         2.275e-03         3.303e-04        -1.464e-02
##        Usagecont          CarbFlow           Density               MFR
##       -1.025e-02         2.466e-05        -3.515e-02        -8.403e-05
##   PressureVacuum      OxygenFiller      BowlSetpoint  PressureSetpoint
##        4.981e-03         9.010e-02         3.989e-03        -1.088e-02
##     AirPressurer             BrandB            BrandA            BrandC
```

```
##       -1.929e-03         1.272e+01         1.268e+01         1.258e+01
##           BrandD            BrandU
##        1.277e+01         1.263e+01
```

linear

```
## Linear Regression
##
## 2059 samples
##   26 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1853, 1854, 1853, 1853, 1853, 1853, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1390388  0.3502282  0.1106914
##
## Tuning parameter 'intercept' was held constant at a value of FALSE
```

```
#variable importance
varImp( linear$finalModel)
```

```
##                     Overall
## CarbVolume        2.2862607
## FillOunces        2.2655367
## PCVolume          2.9718421
## CarbPressure      1.6503560
## CarbTemp          1.5618292
## PSC               2.1013428
## PSCFill           1.2645029
## PSCCO2            1.3464366
## CarbPressure1     8.8775348
## FillPressure      1.5904365
## HydPressure4      0.9738716
## Temperature       5.8590174
## Usagecont         8.1681713
## CarbFlow          6.6574016
## Density           1.4927533
## MFR               1.9758209
## PressureVacuum    0.7750041
## OxygenFiller      1.2752314
## BowlSetpoint     14.9724282
## PressureSetpoint  4.8730647
## AirPressurer      0.7119170
## BrandB           10.8882932
## BrandA           10.8432836
## BrandC           10.7608394
## BrandD           10.9301632
## BrandU           10.8019944
```

c) Stepwise

- An extension of the multiple linear regression. The stepwise model iteratively searches for the best model by dropping one variable at a time. The default is to remove in a backwards direction. The caret train() function was used to tune the model with method indicator parameter of 'lmStepAIC'

```
set.seed(143)
step.linear <- train(PH ~ ., data = train.df , metric = 'RMSE', method =
"lmStepAIC", trControl = train.control, trace = FALSE)
step.linear

## Linear Regression with Stepwise Selection
##
## 2059 samples
##   26 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1853, 1854, 1853, 1853, 1853, 1853, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1397659  0.3436441  0.1111799

step.linear$finalModel

##
## Call:
## lm(formula = .outcome ~ CarbVolume + FillOunces + PCVolume +
##     CarbPressure + CarbTemp + PSC + PSCCO2 + CarbPressure1 +
##     Temperature + Usagecont + CarbFlow + Density + MFR + BowlSetpoint +
##     PressureSetpoint + BrandB + BrandA + BrandC + BrandD, data = dat)
##
## Coefficients:
##      (Intercept)        CarbVolume        FillOunces          PCVolume
##        1.251e+01        -2.243e-01        -8.800e-02        -1.909e-01
##     CarbPressure          CarbTemp               PSC           PSCCO2
##        7.422e-03        -5.509e-03        -1.513e-01        -1.053e-01
##    CarbPressure1       Temperature         Usagecont          CarbFlow
##        6.994e-03        -1.451e-02        -1.029e-02         2.515e-05
##          Density               MFR      BowlSetpoint  PressureSetpoint
##       -4.074e-02        -1.193e-04         4.024e-03        -9.217e-03
##           BrandB            BrandA            BrandC            BrandD
##        8.991e-02         5.011e-02        -5.210e-02         1.377e-01

#variable importance
varImp( step.linear$finalModel)

##                   Overall
## CarbVolume        2.257470
## FillOunces        2.426564
```

```
## PCVolume           3.296571
## CarbPressure       1.562810
## CarbTemp           1.476080
## PSC                2.398979
## PSCCO2             1.494280
## CarbPressure1      8.852715
## Temperature        6.020287
## Usagecont          8.486585
## CarbFlow           7.225758
## Density            1.768013
## MFR                3.194434
## BowlSetpoint      15.798601
## PressureSetpoint   5.347868
## BrandB             5.984269
## BrandA             2.241030
## BrandC             3.148685
## BrandD             5.549640
```

d) Partial Least Square model

- Partial least squares (PLS) regression is a technique that reduces the predictors to a smaller set of uncorrelated components and performs least squares regression on these components, instead of on the original data source. The caret train() function was used to tune the model with method indicator parameter of 'pls'.

```
set.seed(143)
pls <- train(PH ~ ., data = train.df, metric = 'RMSE', method ='pls',
preProcess = c('center', 'scale'), tunelength = 15, trControl =
train.control)
pls$finalModel

## Partial least squares regression , fitted with the orthogonal scores
algorithm.
## Call:
## plsr(formula = .outcome ~ ., ncomp = param$ncomp, data = dat,     method =
"oscorespls", tunelength = 15)

pls$results

##   ncomp      RMSE  Rsquared       MAE      RMSESD RsquaredSD       MAESD
## 1     1 0.1477605 0.2648669 0.1173026 0.005684922 0.04500596 0.005159802
## 2     2 0.1423809 0.3168928 0.1131319 0.005807653 0.04832799 0.005066523
## 3     3 0.1406837 0.3337172 0.1124199 0.005556905 0.05289861 0.005433135

#variable importance
varImp( pls$finalModel)

##                     Overall
## CarbVolume       0.004181108
## FillOunces       0.005559869
## PCVolume         0.003045400
## CarbPressure     0.003533676
```

```
## CarbTemp         0.001050275
## PSC              0.005279005
## PSCFill          0.001574585
## PSCCO2           0.003066963
## CarbPressure1    0.006880636
## FillPressure     0.011899611
## HydPressure4     0.008126951
## Temperature      0.009844618
## Usagecont        0.017550618
## CarbFlow         0.008699394
## Density          0.005751894
## MFR              0.002601945
## PressureVacuum   0.012031799
## OxygenFiller     0.007730633
## BowlSetpoint     0.020857642
## PressureSetpoint 0.015808299
## AirPressurer     0.001134145
## BrandB           0.009088624
## BrandA           0.006408373
## BrandC           0.017058194
## BrandD           0.009983744
## BrandU           0.004619746
```

e) Random Forest

- The random forest algorithm works by aggregating the predictions made by multiple decision trees of varying depth. Every decision tree in the forest is trained on a subset of the dataset called the bootstrapped dataset source.The caret train() function was used to tune the model with method indicator parameter of 'rf'. *Note the random forest takes a good amount of time to load, the group decided to utilize the default tunelength of 3 due to these constraints*

```
# Create model with default paramters
rf.control <- trainControl(method="repeatedcv", number=5, repeats=3,
search="random")
set.seed(143)
#mtry: Number of variables randomly sampled as candidates at each split.
mtry <- sqrt(ncol(train.df))

random.forest<- train(PH~., data=train.df, metric = 'RMSE' , method="rf",
trControl= rf.control,importance =T)
random.forest
```

```
## Random Forest
##
## 2059 samples
##   26 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 1646, 1648, 1648, 1648, 1646, 1646, ...
```

```
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##    1    0.1351406  0.4728258  0.10709948
##   11    0.1108986  0.5978037  0.08358820
##   20    0.1106735  0.5936839  0.08266726
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 20.
```

```
random.forest$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, importance = ..1)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 20
##
##           Mean of squared residuals: 0.01169679
##                     % Var explained: 60.44
```

```
random.forest$results
```

```
##   mtry      RMSE  Rsquared        MAE      RMSESD RsquaredSD       MAESD
## 1    1 0.1351406 0.4728258 0.10709948 0.004972765 0.04763032 0.003482229
## 2   11 0.1108986 0.5978037 0.08358820 0.006625957 0.04818090 0.004491333
## 3   20 0.1106735 0.5936839 0.08266726 0.006820566 0.04888639 0.004465864
```

```
#variable importance
varImp( random.forest$finalModel)
```

```
##                   Overall
## CarbVolume       20.760455
## FillOunces        5.016696
## PCVolume         19.847999
## CarbPressure      5.068488
## CarbTemp          1.113976
## PSC               2.240945
## PSCFill           4.943980
## PSCCO2           -1.978690
## CarbPressure1    32.171068
## FillPressure     14.603928
## HydPressure4     11.169990
## Temperature      43.294757
## Usagecont        69.351649
## CarbFlow         30.094404
## Density          31.074656
## MFR              17.513866
## PressureVacuum   44.796522
## OxygenFiller     48.155774
```

```
## BowlSetpoint     49.383898
## PressureSetpoint 16.268037
## AirPressurer     36.851501
## BrandB           14.133281
## BrandA           20.263780
## BrandC           53.788034
## BrandD           21.194957
## BrandU           17.168681
```

f) KNN

- The knn() function identifies the k-nearest neighbors using Euclidean distance where k is a user-specified number. The value for k is generally chosen as the square root of the number of observations. knn classifies new cases by a majority vote of its k neighbors. This algorithms segregates unlabeled data points into well defined groups source.The caret train() function was used to tune the model with method indicator parameter of 'knn'.

```
set.seed(143)
knn <- train(PH ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = 1:10),
             trControl  = train.control,
             metric     = "RMSE",
             data       = train.df,preProc =c("center", "scale"))
knn
```

```
## k-Nearest Neighbors
##
## 2059 samples
##   26 predictor
##
## Pre-processing: centered (26), scaled (26)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1853, 1854, 1853, 1853, 1853, 1853, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    1  0.1609965  0.3192015  0.11327855
##    2  0.1398724  0.3961582  0.10307060
##    3  0.1341496  0.4212506  0.10052946
##    4  0.1297281  0.4466161  0.09809383
##    5  0.1284165  0.4534512  0.09731105
##    6  0.1281554  0.4525795  0.09628552
##    7  0.1279291  0.4528643  0.09691187
##    8  0.1279979  0.4519583  0.09709169
##    9  0.1271743  0.4579162  0.09685967
##   10  0.1267458  0.4613414  0.09690715
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 10.
```

```
knn$finalModel

## 10-nearest neighbor regression model

knn$results

##     k      RMSE  Rsquared        MAE      RMSESD RsquaredSD       MAESD
## 1   1 0.1609965 0.3192015 0.11327855 0.010857391 0.07773374 0.008844057
## 2   2 0.1398724 0.3961582 0.10307060 0.008326285 0.06404515 0.007135962
## 3   3 0.1341496 0.4212506 0.10052946 0.008601719 0.07078500 0.005718208
## 4   4 0.1297281 0.4466161 0.09809383 0.007068029 0.06492708 0.004752165
## 5   5 0.1284165 0.4534512 0.09731105 0.005603217 0.05489998 0.003744808
## 6   6 0.1281554 0.4525795 0.09628552 0.005526775 0.05367217 0.003837955
## 7   7 0.1279291 0.4528643 0.09691187 0.005390734 0.05378057 0.004258764
## 8   8 0.1279979 0.4519583 0.09709169 0.005542693 0.05236798 0.004281751
## 9   9 0.1271743 0.4579162 0.09685967 0.005435037 0.05389345 0.004471553
## 10 10 0.1267458 0.4613414 0.09690715 0.005163525 0.05162884 0.004279798
```

5. Model Evaluation & Selection

a) Create evaluations table for each model (train and test data)
b) Select and evaluate model
c) Predict PH using evaluation dataset
d) Export Results to excel

a. Create evaluations table for each model (train and test data)

- RMSE, MAE, and R^2 based on train data

```
models <- list(linear,step.linear, glm, pls, random.forest,knn)


cols <- c('linear.model',
'stepwise.linear.model','pls.model','random.forest.model','knn.model')

rmse <- data.frame(cbind(min(linear$results$RMSE),
min(step.linear$results$RMSE), min(pls$results$RMSE),
min(random.forest$results$RMSE), min(knn$results$RMSE)), row.names =
'rsme.models')
colnames(rmse)<-cols

rsquared <- data.frame(cbind(max(linear$results$Rsquared),
max(step.linear$results$Rsquared), max(pls$results$Rsquared),
max(random.forest$results$Rsquared), max(knn$results$Rsquared)), row.names =
'rsquared.models')
colnames(rsquared)<-cols

mae <- data.frame(cbind(min(linear$results$MAE),
min(step.linear$results$MAE),  min(pls$results$MAE),
min(random.forest$results$MAE), min(knn$results$MAE)), row.names =
'mae.models')
colnames(mae)<-cols
```

```
rbind(rmse,
rsquared,
mae)
```

```
##                 linear.model stepwise.linear.model pls.model
## rsme.models        0.1390388             0.1397659 0.1406837
## rsquared.models    0.3502282             0.3436441 0.3337172
## mae.models         0.1106914             0.1111799 0.1124199
##                 random.forest.model  knn.model
## rsme.models              0.11067351 0.12674581
## rsquared.models          0.59780369 0.46134136
## mae.models               0.08266726 0.09628552
```

- RMSE, MAE, and R^2 based on test data

```
pred.test <- function(model, model.label, testData, ytest) {

 cols <- c('Model.Name', 'RMSE.Test','RSquared.Test','MAE.Test')

  # Predict Model on test.df
  pred <- predict(model, testData)


#https://www.rdocumentation.org/packages/caret/versions/2.27/topics/postResam
ple
  post.resample<- postResample(pred = pred, obs = ytest)

  rmse <- c(post.resample[[1]])
  r2 <- c(post.resample[[2]])
  mae <- c(post.resample[[3]])

  return.df <- data.frame(cbind(model.label, round(rmse,3), round(r2,3),
round(mae,3)))

  colnames(return.df)<-cols
  return(return.df)
}


#Prediction based on test.df

test.performance <- rbind(pred.test(linear, "linear", test.df, test.df$PH),
                          pred.test(step.linear, "step.linear", test.df,
test.df$PH),
                          pred.test(pls, "pls", test.df, test.df$PH),
                          pred.test(random.forest, "random.forest", test.df,
test.df$PH),
                          pred.test(knn, "knn", test.df, test.df$PH))
```

```
test.performance
```

```
##      Model.Name RMSE.Test RSquared.Test MAE.Test
## 1        linear     0.144         0.321    0.112
## 2   step.linear     0.144         0.321    0.112
## 3           pls     0.144         0.322    0.112
## 4 random.forest     0.108         0.623    0.078
## 5           knn     0.128         0.466    0.094
```

- From the summary tables, it can be observed that the random forest model is the best model based on the RSME, R^2 and MAE calculations.

b) Select and evaluate model

Display the caret VarImp() to assess the predictors' impact in the random forest final model - Usagecount, BowlSetpoint, and Temperature appear to be the chief driving factors influencing PH.

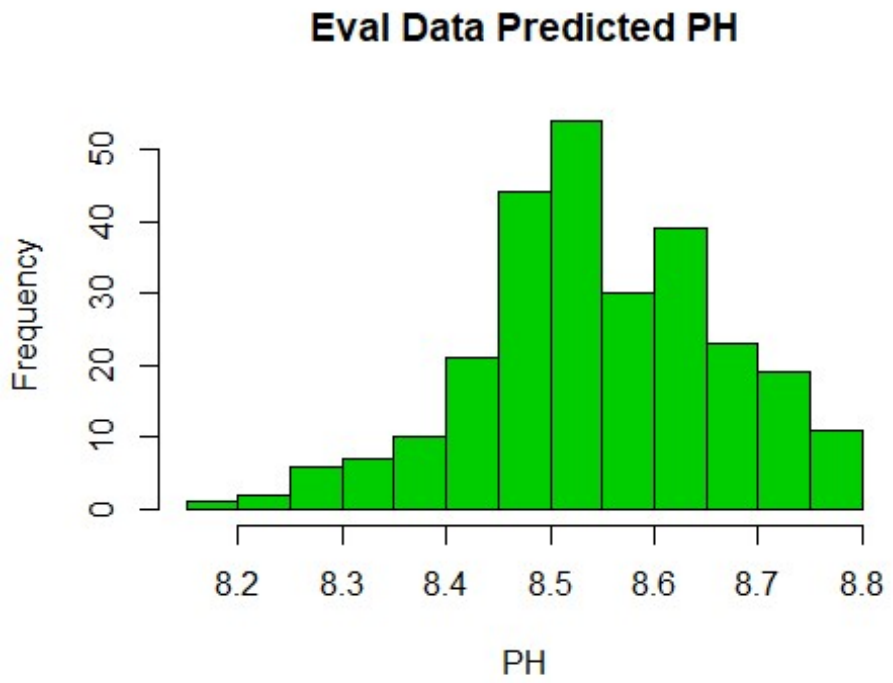
```
as.data.frame(varImp( random.forest$finalModel))%>%
  tibble::rownames_to_column("Predictors")%>%
  arrange(-Overall)
```

```
##          Predictors   Overall
## 1         Usagecont 69.351649
## 2            BrandC 53.788034
## 3      BowlSetpoint 49.383898
## 4      OxygenFiller 48.155774
## 5    PressureVacuum 44.796522
## 6       Temperature 43.294757
## 7      AirPressurer 36.851501
## 8     CarbPressure1 32.171068
## 9           Density 31.074656
## 10         CarbFlow 30.094404
## 11           BrandD 21.194957
## 12       CarbVolume 20.760455
## 13           BrandA 20.263780
## 14         PCVolume 19.847999
## 15              MFR 17.513866
## 16           BrandU 17.168681
## 17 PressureSetpoint 16.268037
## 18     FillPressure 14.603928
## 19           BrandB 14.133281
## 20     HydPressure4 11.169990
## 21     CarbPressure  5.068488
## 22       FillOunces  5.016696
## 23          PSCFill  4.943980
## 24              PSC  2.240945
## 25         CarbTemp  1.113976
## 26           PSCCO2 -1.978690
```

c) Predict PH using evaluation dataset

- The group will predict PH using the random.forest model and summarize the results in a histogram
- The evaluation distribution of PH is consistent with the original train values. Recall, 85% of the train data set PH values were between 8.3 and 8.8. Based of the predicted PH values, approximately 75% are between 8.45 and 8.7.

```
validations.df <- eval.imputed.dum
validations.df$PH<- NULL


prediction.rf <-as.data.frame( predict(random.forest, validations.df))
summary(prediction.rf)

##  predict(random.forest, validations.df)
##  Min.   :8.168
##  1st Qu.:8.477
##  Median :8.542
##  Mean   :8.547
##  3rd Qu.:8.633
##  Max.   :8.798

hist.ph.eval<- hist(predict(random.forest, validations.df),
    main = 'Eval Data Predicted PH',
    xlab = 'PH',
    col = 11)
```



```
breaks <- hist.ph.eval$breaks

value_list <- c()
i <- 1
for (val in breaks){
  if(which(breaks == val) != 1){
    value_list[[i]] <- paste0(breaks[which(breaks == val)-1]," to
",breaks[which(breaks == val)] )
    i <- i +1
  }
}
#create a dateframe from the breaks and the counts
ph_df_eval<- data.frame(breaks= value_list, counts = hist.ph.eval$counts)
ph_df_eval$percentage<-round((ph_df_eval$counts/ sum(ph_df_eval$counts))
*100,2)
ph_df_eval
```

```
##         breaks counts percentage
## 1  8.15 to 8.2      1       0.37
## 2  8.2 to 8.25      2       0.75
## 3  8.25 to 8.3      6       2.25
## 4  8.3 to 8.35      7       2.62
## 5  8.35 to 8.4     10       3.75
## 6  8.4 to 8.45     21       7.87
## 7  8.45 to 8.5     44      16.48
## 8  8.5 to 8.55     54      20.22
## 9  8.55 to 8.6     30      11.24
## 10 8.6 to 8.65     39      14.61
## 11 8.65 to 8.7     23       8.61
## 12 8.7 to 8.75     19       7.12
## 13 8.75 to 8.8     11       4.12
```

d) Export results to excel

- The results are written to “Group1_Project2_Results.csv”

```
write.csv(prediction.rf, file = "Group1_Project2_Results.csv")
```