Burk Soda Company pH Predictive Modeling

July 18th 2020

Data Analytics Division: Group 4

Layla Quinones

Sergio Ortega

Jack Russo

William Outcalt

Neil Shah

Table of Contents

Executive Summary

Potential for Hydrogen (or pH) is an important Key Performance Indicator (KPI) in Burk Soda Company production. Group 4, of the Data Analytics division, used live production data generated on July 2nd 2020 consisting of 33 variables [1 categorical and 32 numerical] to develop a predictive model for pH. After preprocessing the data was split and 80% was used to train models, while the remaining 20% was used to measure accuracy. The data was fit to an Multiple Linear Regression model used as a baseline, and Random Forest and Gradient Boosting models. Accuracy for all models were calculated using **MAPE** and **RMSE** error metrics and compared to select the best model to predict pH.

Group 4 concluded and recommends:

- A Random Forest model performed better against a Multiple Linear Regression model, and a Gradient Boosting model. Due to inherent trade-offs between computational time and training, Gradient Boosting can be used for precision applications or paired with Random Forest.
- Some predictor variables offered little insight into pH so resources for measuring said metrics can be optimized to the more germane variables; specifically focus data aqui data to fill gaps.
- Based on analysis of the relationship between variables, we have identified that beverages with **Brand Code** B and C were indistinguishable from each other and therefore may be consolidated into one common manufacturing process for cost saving and efficiencies.

Technical Terminology

Dimensionality Reduction: Method to reduce a higher dimensional dataset into a lower dimensional one. These lower dimensions contain data that is either more relevant, independent, or complete than the dropped dimensions. PCA is one common technique. [7]

Elbow method: A method to find the optimum number of clusters in a K-Means algorithm by identifying the point of diminishing utility (visually resembling an elbow) on a plot of a cost function vs k (number of clusters). [8]

Github Repository: Online version control and code base management platform.

Gradient Boosted Model: Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models [9]

Heat map: Visualization that uses a color scale to show the relationship (typically correlation) between variables.

Hot-encoding: The transformation of categorical variables into many vectors of ones and zeros. Each one represents “is that category” and each zero represents “is not that category”. This representation of categorical data better translates into model building. [10]

K-Means: Unsupervised machine learning technique that attempts to partition data into *k* number of clusters that minimize a specified metric, commonly WCSS. [11]

K-NN: Classification technique that attempts to identify variables based on surrounding (or neighbor) data points. [12]

MAPE: Mean Absolute Percent Error or the deviation in percent terms of a prediction versus the actual value. [13]

Multicollinearity: When many predictors are correlated to each other. This does not reduce the predictive power of a multiple regression model on the training data but rather the test data. [14]

Multimodal Shape: Description of a histogram with multiple peaks or modes. [15]

OLS: Ordinary Least Squares or linear regression in which a target variable is estimated using a linear combination of predictor variables which minimize the sum of squared errors between estimated and actual data. [16]

Overfitting: Common pitfall where a model learns the exact nuance and has a high accuracy on the training set but low accuracy on unseen or test data - model is too specific and has high variance. [17]

PCA: Principal Component Analysis--dimensionality reduction technique that deconstructs a data set into a smaller set of “principal components” which capture the majority of variability in the original set. [18]

pH: A measure of potential for Hydrogen in a liquid solution on a log scale from 0 to 14, with less than 7 implying acidity and greater than 7 implying alkalinity.

Random Forest: Regression technique that uses a set of decision trees defined by predictor variables to predict a value (typically mean) of a target variable. [19]

Random Sampling: a sampling technique in which each sample selected has an equal probability of being chosen

Skew: Also referred to as kurtosis. Description of a distribution in which one tail is longer than the other. **Left skew** or negative skew implies mode is larger than median which is larger than mean. **Right skew** or positive skew implies mean is larger than median which is larger than mode. [20]

Stepwise Regression: Regression technique that methodically adds (**forward**) or subtract (**backwards**) predictor variables until the best fit (usually defined by R^2) is reached. [21]

Testing set: Out of sample data used to test models on untrained data.

Training set: In sample data used to train the model.

XgBoost: Common framework for Gradient Boosting.

Underfitting: when a model fails to model the training data or generalize features for new data - model is over simplified and has high bias

Introduction

With the success in sales of Burk Cola's "Cream-ARIMA-Soda" and "White Noise Tea" executive management is considering capital projects to improve production lines, specifically through pH modeling. Potential for Hydrogen, or pH, is a scale that defines the acidity or alkalinity of a solution. The pH scale [Figure 1] ranges from 0 to 14, and is log based, with 7 being defined as neutral or theoretical pure water. A pH less than 7 indicates a higher ratio of Hydrogen ions or acidity and a pH greater than 7 indicates a higher ratio of Hydroxyl ions and thus alkalinity or basicity.

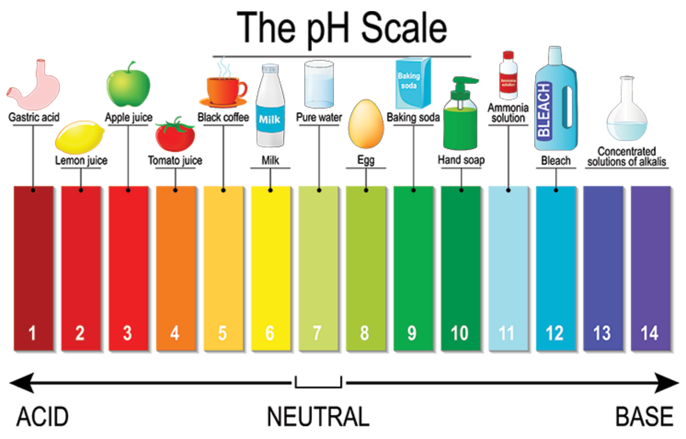


Figure 1: pH Scale [1]

As pH affects taste, stability and composition of beverages [2], there is special interest to predict and model pH for quality control and beverage development. Data Analytics Division Group 4 was tasked with developing a predictive model [3] based on production data from the New York City facility dated July 2nd 2020. This report lays out the analysis, best practices, lessons learned, and recommendations for future improvements .

Methodology

Group 4 followed a multi-step methodology [4] to perform predictive analysis:

1. **Exploratory Data Analysis:** Ingest data into statistical software (R), produce visualizations, generate descriptive statistics and gain understanding of the behavior of each predictor included in raw data.
2. **Pre-Processing and Dimensionality Reduction :** Impute missing values, normalize and scale predictors, address outliers and identify predictors to use during model training.
3. **Model Development:** Split data into train and test sets; use these sets to train and evaluate model performance.
4. **Analysis**: Determine best performing model using Mean Absolute Percent Error (MAPE) and use this model to make recommendations to the company.

Packages Used

The following R packages were used in this project:

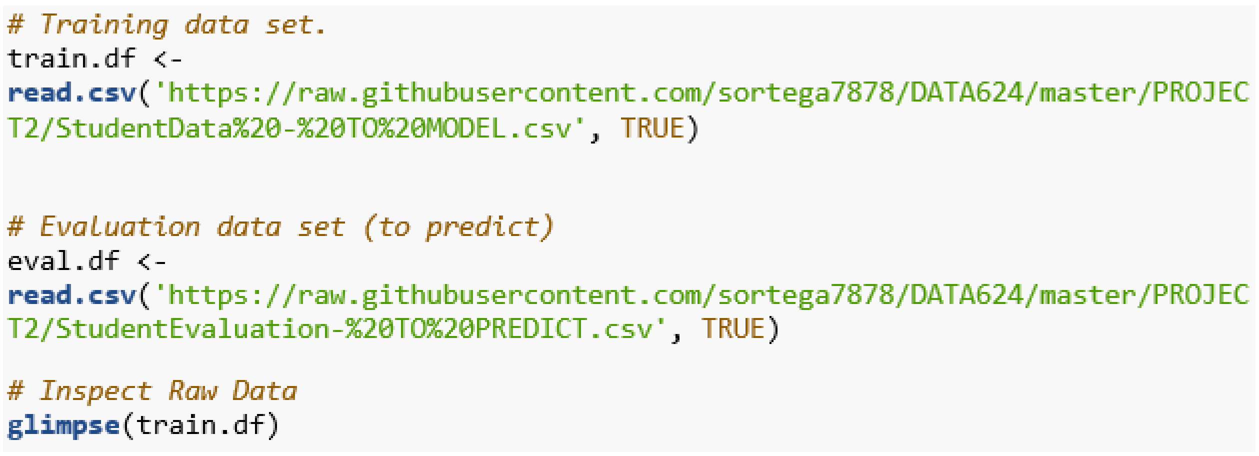
```
"parallel", "MASS", "dplyr", "tidyr", "reshape2", "funModeling", "naniar", "visdat", "corrplot",
"lattice", "caret",
"psych","ggplot2","prettyR","ggbiplot","nnet","randomForest","xgboost","fastDummies", "mlMetrics")
```

Exploratory Data Analysis and Preprocessing

Group 4 was given two comma separated value (.csv) datasets, specifically a **training set** with 2571 cases and a prediction set with 267 cases, both of which were uploaded to a shared Github repository and then ingested into statistical software R [Fig 2].

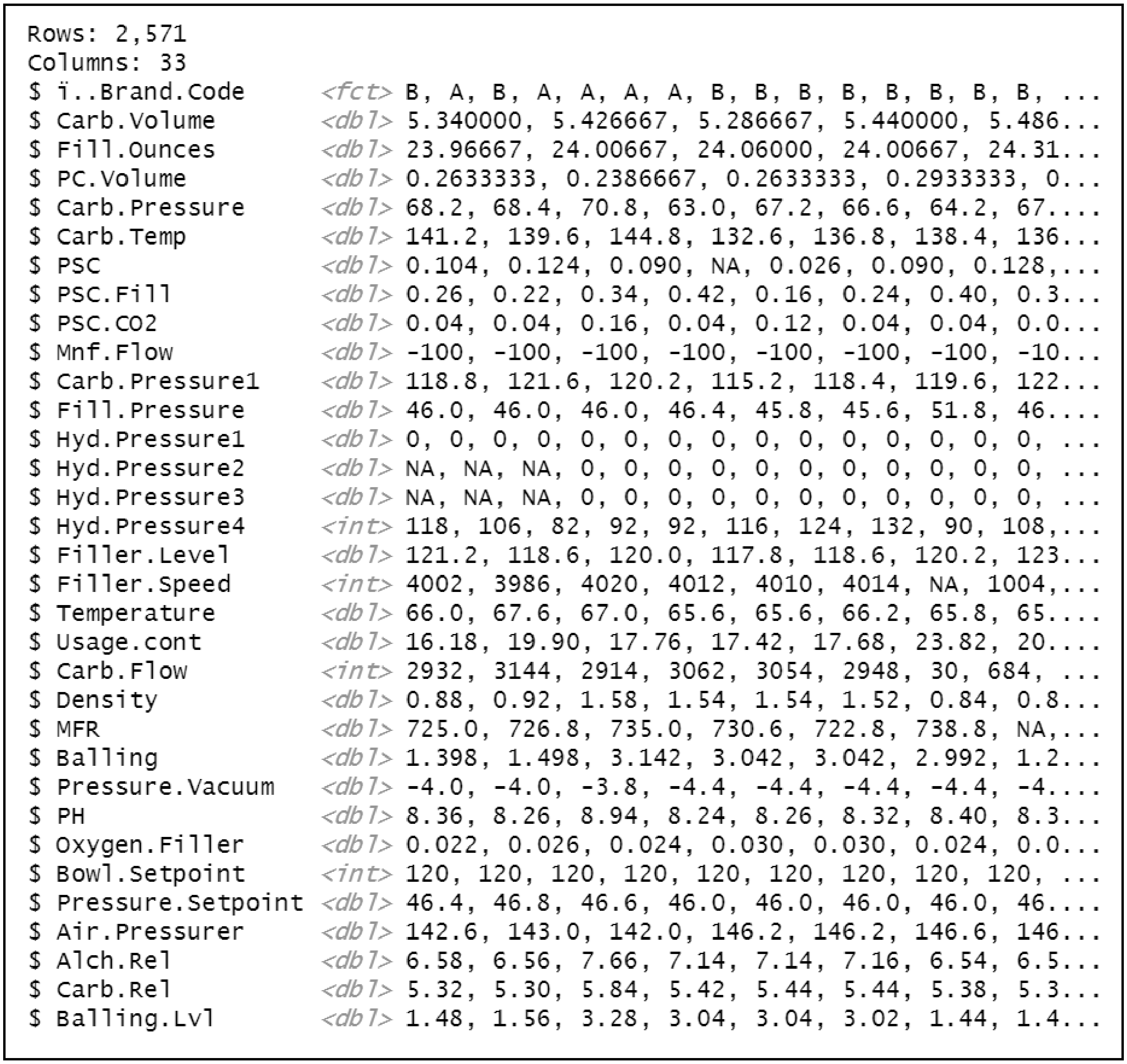
Exploratory data analysis was conducted on both data sets to gather information about the behavior of predictors, determine appropriate data preprocessing steps and gain insight into which models we should train for the most accurate predictions. Each data set contained 32 predictors; the training data set contained values for the target variable (pH) which were used to assess accuracy and choose the best model. After loading the .csv files from our **Github repository** into our R environment [Fig 2], we inspected the data to gain some initial insights into its behavior.

Code



```
# Training data set.
train.df <-
read.csv('https://raw.githubusercontent.com/sortega7878/DATA624/master/PROJEC
T2/StudentData%20-%20TO%20MODEL.csv', TRUE)


# Evaluation data set (to predict)
eval.df <-
read.csv('https://raw.githubusercontent.com/sortega7878/DATA624/master/PROJEC
T2/StudentEvaluation-%20TO%20PREDICT.csv', TRUE)

# Inspect Raw Data
glimpse(train.df)
```

Output (training data set)



```
Rows: 2,571
Columns: 33
$ ï..Brand.Code     <fct> B, A, B, A, A, A, A, B, B, B, B, B, B, B, B, ...
$ Carb.Volume       <dbl> 5.340000, 5.426667, 5.286667, 5.440000, 5.486...
$ Fill.Ounces       <dbl> 23.96667, 24.00667, 24.06000, 24.00667, 24.31...
$ PC.Volume         <dbl> 0.2633333, 0.2386667, 0.2633333, 0.2933333, 0...
$ Carb.Pressure     <dbl> 68.2, 68.4, 70.8, 63.0, 67.2, 66.6, 64.2, 67....
$ Carb.Temp         <dbl> 141.2, 139.6, 144.8, 132.6, 136.8, 138.4, 136...
$ PSC               <dbl> 0.104, 0.124, 0.090, NA, 0.026, 0.090, 0.128,...
$ PSC.Fill          <dbl> 0.26, 0.22, 0.34, 0.42, 0.16, 0.24, 0.40, 0.3...
$ PSC.CO2           <dbl> 0.04, 0.04, 0.16, 0.04, 0.12, 0.04, 0.04, 0.0...
$ Mnf.Flow          <dbl> -100, -100, -100, -100, -100, -100, -100, -10...
$ Carb.Pressure1    <dbl> 118.8, 121.6, 120.2, 115.2, 118.4, 119.6, 122...
$ Fill.Pressure     <dbl> 46.0, 46.0, 46.0, 46.4, 45.8, 45.6, 51.8, 46....
$ Hyd.Pressure1     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ Hyd.Pressure2     <dbl> NA, NA, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ Hyd.Pressure3     <dbl> NA, NA, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ Hyd.Pressure4     <int> 118, 106, 82, 92, 92, 116, 124, 132, 90, 108,...
$ Filler.Level      <dbl> 121.2, 118.6, 120.0, 117.8, 118.6, 120.2, 123...
$ Filler.Speed      <int> 4002, 3986, 4020, 4012, 4010, 4014, NA, 1004,...
$ Temperature       <dbl> 66.0, 67.6, 67.0, 65.6, 65.6, 66.2, 65.8, 65....
$ Usage.cont        <dbl> 16.18, 19.90, 17.76, 17.42, 17.68, 23.82, 20....
$ Carb.Flow         <int> 2932, 3144, 2914, 3062, 3054, 2948, 30, 684, ...
$ Density           <dbl> 0.88, 0.92, 1.58, 1.54, 1.54, 1.52, 0.84, 0.8...
$ MFR               <dbl> 725.0, 726.8, 735.0, 730.6, 722.8, 738.8, NA,...
$ Balling           <dbl> 1.398, 1.498, 3.142, 3.042, 3.042, 2.992, 1.2...
$ Pressure.Vacuum   <dbl> -4.0, -4.0, -3.8, -4.4, -4.4, -4.4, -4.4, -4....
$ PH                <dbl> 8.36, 8.26, 8.94, 8.24, 8.26, 8.32, 8.40, 8.3...
$ Oxygen.Filler     <dbl> 0.022, 0.026, 0.024, 0.030, 0.030, 0.024, 0.0...
$ Bowl.Setpoint     <int> 120, 120, 120, 120, 120, 120, 120, 120, 120, ...
$ Pressure.Setpoint <dbl> 46.4, 46.8, 46.6, 46.0, 46.0, 46.0, 46.0, 46....
$ Air.Pressurer     <dbl> 142.6, 143.0, 142.0, 146.2, 146.2, 146.6, 146...
$ Alch.Rel          <dbl> 6.58, 6.56, 7.66, 7.14, 7.14, 7.16, 6.54, 6.5...
$ Carb.Rel          <dbl> 5.32, 5.30, 5.84, 5.42, 5.44, 5.44, 5.38, 5.3...
$ Balling.Lvl       <dbl> 1.48, 1.56, 3.28, 3.04, 3.04, 3.02, 1.44, 1.4...
```

Fig 2 : R Code for loading train set, and evaluation set

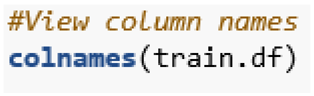
Interpretation

Inspecting the data revealed a total of 33 columns consisting of 32 predictor variable columns, of which 31 were numerical and one categorical, and a numerical target variable pH**.** There was missing data for various predictors and many column names were improperly formatted. We continued our analysis by renaming these columns and inspecting missing values for each column.

Cleaning Column Names

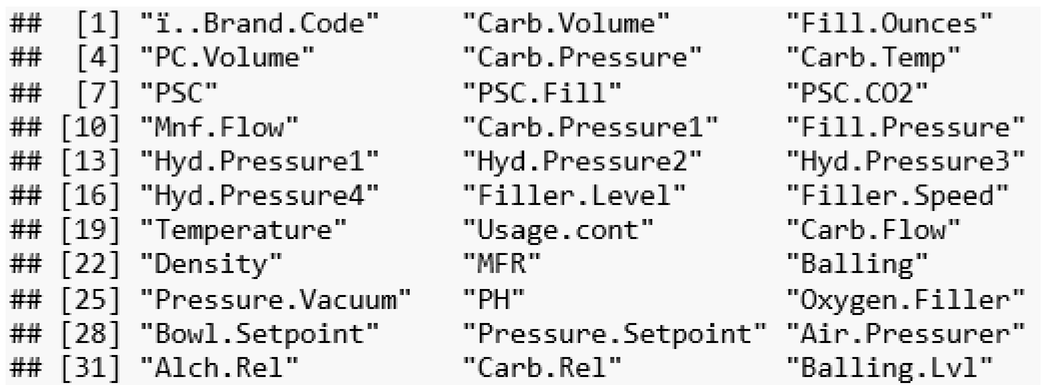
Upon inspection of predictor column names [Fig 3], we elected to rename columns so that they were easier to read and interpret [Fig 4]. This approach makes visualizations and results easier to interpret downstream.

Code



```
#View column names
colnames(train.df)
```

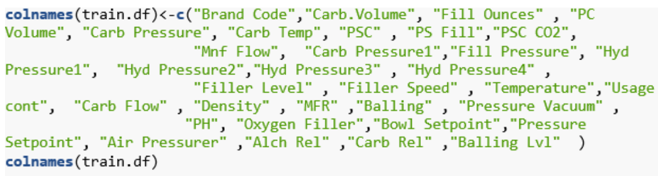
Output



```
##  [1] "ï..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"
```

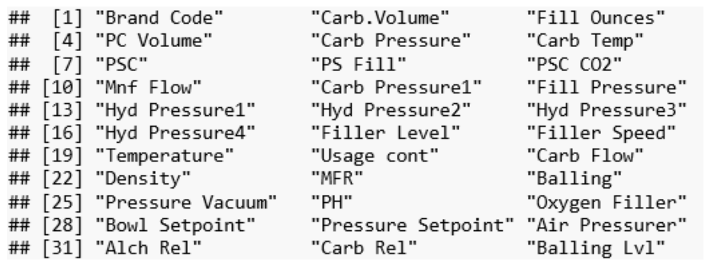
Fig 3 : R Code for identifying column names

Code



```
colnames(train.df)<-c("Brand Code","Carb.Volume", "Fill Ounces" , "PC
Volume", "Carb Pressure", "Carb Temp", "PSC" , "PS Fill","PSC CO2",
                      "Mnf Flow",  "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","Pressure
Setpoint", "Air Pressurer" ,"Alch Rel" ,"Carb Rel" ,"Balling Lvl"  )
colnames(train.df)
```

Output



```
##  [1] "Brand Code"        "Carb.Volume"       "Fill Ounces"
##  [4] "PC Volume"         "Carb Pressure"     "Carb Temp"
##  [7] "PSC"               "PS 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"
```

Fig 4 : R Code for renaming columns

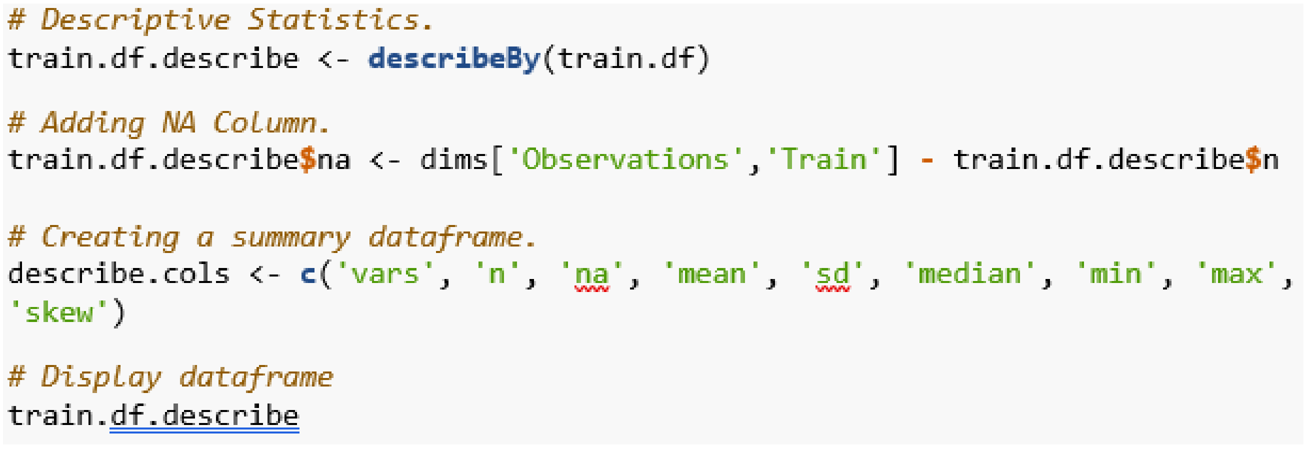
Interpretation

This reassignment permits us to quickly identify which naming conventions were used and how each variable may be related to the target quantity(pH). This will be useful later as we analyze our model to identify which predictors influence models the most.

Summary Statistics By Predictor

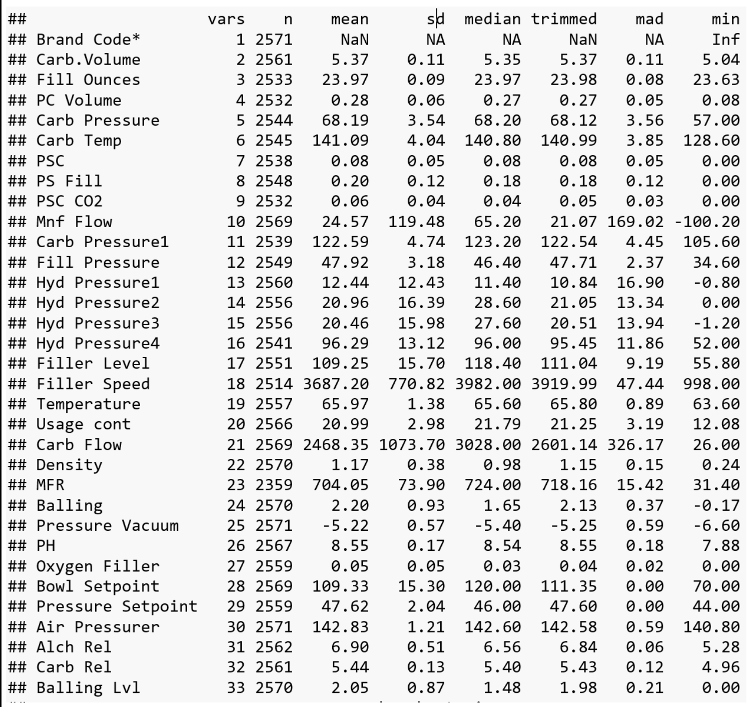
Summary statistics were calculated to understand the distribution and characteristics of the predictor variables.

Code

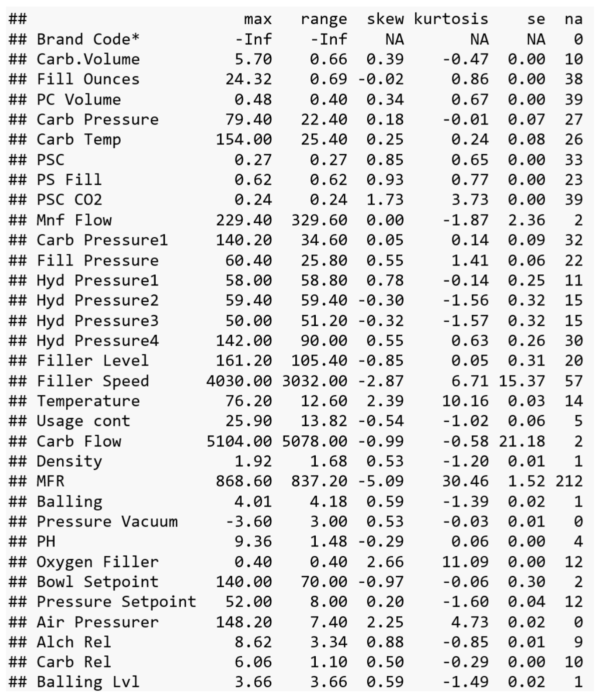


```r
# Descriptive Statistics.
train.df.describe <- describeBy(train.df)

# Adding NA Column.
train.df.describe$na <- dims['Observations','Train'] - train.df.describe$n

# Creating a summary dataframe.
describe.cols <- c('vars', 'n', 'na', 'mean', 'sd', 'median', 'min', 'max',
'skew')

# Display dataframe
train.df.describe
```

Output



```
##                    vars    n    mean      sd  median trimmed    mad     min
## Brand Code*           1 2571     NaN      NA      NA     NaN     NA     Inf
## Carb.Volume           2 2561    5.37    0.11    5.35    5.37   0.11    5.04
## Fill Ounces           3 2533   23.97    0.09   23.97   23.98   0.08   23.63
## PC Volume             4 2532    0.28    0.06    0.27    0.27   0.05    0.08
## Carb Pressure         5 2544   68.19    3.54   68.20   68.12   3.56   57.00
## Carb Temp             6 2545  141.09    4.04  140.80  140.99   3.85  128.60
## PSC                   7 2538    0.08    0.05    0.08    0.08   0.05    0.00
## PS Fill               8 2548    0.20    0.12    0.18    0.18   0.12    0.00
## PSC CO2               9 2532    0.06    0.04    0.04    0.05   0.03    0.00
## Mnf Flow             10 2569   24.57  119.48   65.20   21.07 169.02 -100.20
## Carb Pressure1       11 2539  122.59    4.74  123.20  122.54   4.45  105.60
## Fill Pressure        12 2549   47.92    3.18   46.40   47.71   2.37   34.60
## Hyd Pressure1        13 2560   12.44   12.43   11.40   10.84  16.90   -0.80
## Hyd Pressure2        14 2556   20.96   16.39   28.60   21.05  13.34    0.00
## Hyd Pressure3        15 2556   20.46   15.98   27.60   20.51  13.94   -1.20
## Hyd Pressure4        16 2541   96.29   13.12   96.00   95.45  11.86   52.00
## Filler Level         17 2551  109.25   15.70  118.40  111.04   9.19   55.80
## Filler Speed         18 2514 3687.20  770.82 3982.00 3919.99  47.44  998.00
## Temperature          19 2557   65.97    1.38   65.60   65.80   0.89   63.60
## Usage cont           20 2566   20.99    2.98   21.79   21.25   3.19   12.08
## Carb Flow            21 2569 2468.35 1073.70 3028.00 2601.14 326.17   26.00
## Density              22 2570    1.17    0.38    0.98    1.15   0.15    0.24
## MFR                  23 2359  704.05   73.90  724.00  718.16  15.42   31.40
## Balling              24 2570    2.20    0.93    1.65    2.13   0.37   -0.17
## Pressure Vacuum      25 2571   -5.22    0.57   -5.40   -5.25   0.59   -6.60
## PH                   26 2567    8.55    0.17    8.54    8.55   0.18    7.88
## Oxygen Filler        27 2559    0.05    0.05    0.03    0.04   0.02    0.00
## Bowl Setpoint        28 2569  109.33   15.30  120.00  111.35   0.00   70.00
## Pressure Setpoint    29 2559   47.62    2.04   46.00   47.60   0.00   44.00
## Air Pressurer        30 2571  142.83    1.21  142.60  142.58   0.59  140.80
## Alch Rel             31 2562    6.90    0.51    6.56    6.84   0.06    5.28
## Carb Rel             32 2561    5.44    0.13    5.40    5.43   0.12    4.96
## Balling Lvl          33 2570    2.05    0.87    1.48    1.98   0.21    0.00
```



```
##                       max   range  skew kurtosis    se  na
## Brand Code*          -Inf    -Inf    NA       NA    NA   0
## Carb.Volume          5.70    0.66  0.39    -0.47  0.00  10
## Fill Ounces         24.32    0.69 -0.02     0.86  0.00  38
## PC Volume            0.48    0.40  0.34     0.67  0.00  39
## Carb Pressure       79.40   22.40  0.18    -0.01  0.07  27
## Carb Temp          154.00   25.40  0.25     0.24  0.08  26
## PSC                  0.27    0.27  0.85     0.65  0.00  33
## PS Fill              0.62    0.62  0.93     0.77  0.00  23
## PSC CO2              0.24    0.24  1.73     3.73  0.00  39
## Mnf Flow           229.40  329.60  0.00    -1.87  2.36   2
## Carb Pressure1     140.20   34.60  0.05     0.14  0.09  32
## Fill Pressure       60.40   25.80  0.55     1.41  0.06  22
## Hyd Pressure1       58.00   58.80  0.78    -0.14  0.25  11
## Hyd Pressure2       59.40   59.40 -0.30    -1.56  0.32  15
## Hyd Pressure3       50.00   51.20 -0.32    -1.57  0.32  15
## Hyd Pressure4      142.00   90.00  0.55     0.63  0.26  30
## Filler Level       161.20  105.40 -0.85     0.05  0.31  20
## Filler Speed      4030.00 3032.00 -2.87     6.71 15.37  57
## Temperature         76.20   12.60  2.39    10.16  0.03  14
## Usage cont          25.90   13.82 -0.54    -1.02  0.06   5
## Carb Flow         5104.00 5078.00 -0.99    -0.58 21.18   2
## Density              1.92    1.68  0.53    -1.20  0.01   1
## MFR                868.60  837.20 -5.09    30.46  1.52 212
## Balling              4.01    4.18  0.59    -1.39  0.02   1
## Pressure Vacuum     -3.60    3.00  0.53    -0.03  0.01   0
## PH                   9.36    1.48 -0.29     0.06  0.00   4
## Oxygen Filler        0.40    0.40  2.66    11.09  0.00  12
## Bowl Setpoint      140.00   70.00 -0.97    -0.06  0.30   2
## Pressure Setpoint   52.00    8.00  0.20    -1.60  0.04  12
## Air Pressurer      148.20    7.40  2.25     4.73  0.02   0
## Alch Rel             8.62    3.34  0.88    -0.85  0.01   9
## Carb Rel             6.06    1.10  0.50    -0.29  0.00  10
## Balling Lvl          3.66    3.66  0.59    -1.49  0.02   1
```

Fig 5 : R Code for generating summary statistics for columns

Interpretation

Fig 5 shows summary statistics for the predictor variables revealing non-zero **kurtosis** or **skew** for some of the predictor variables, and quantified ranges for each numerical predictor. **Brand Code** has an undefined range due to its categorical nature.

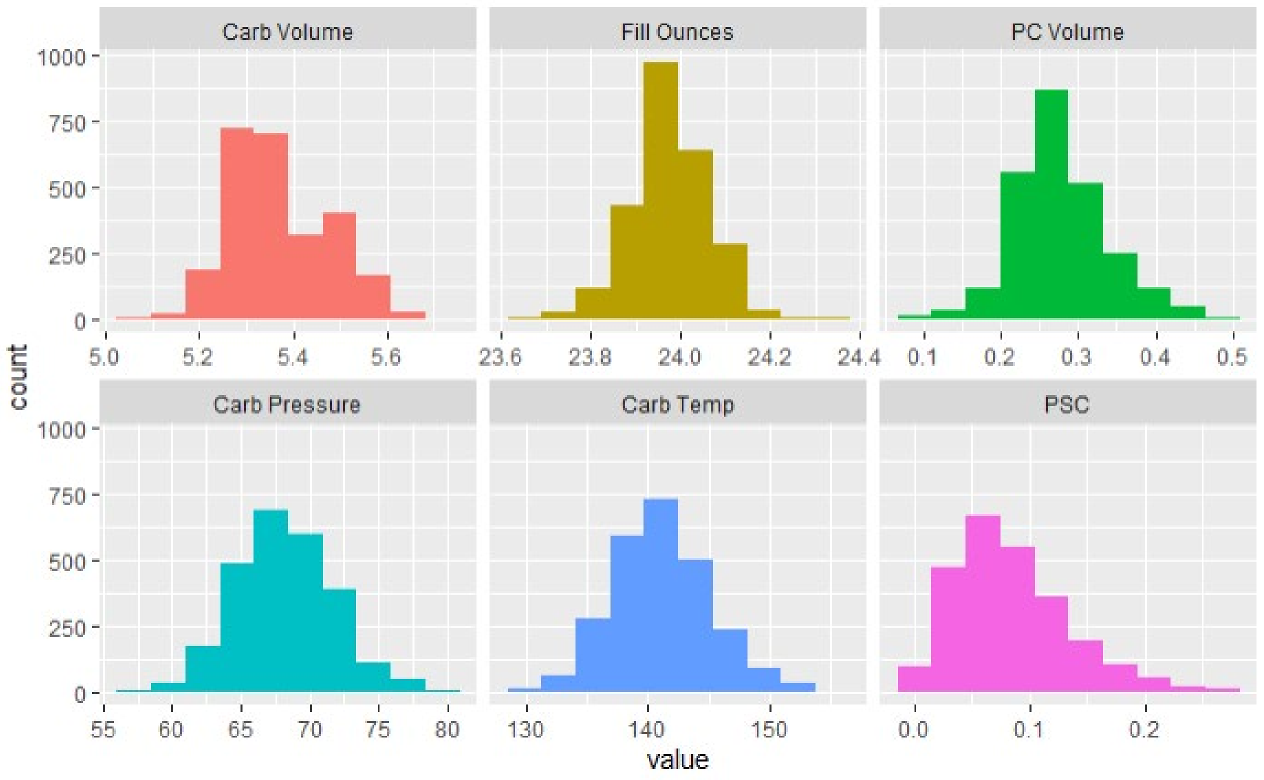
Visualizing Summary Statistics

Using the **plot_num()** function from the **funModeling** library, we created histograms for predictor variables to visualize these statistics, and gain more insight into the shape of each predictor [Fig 6].

Code

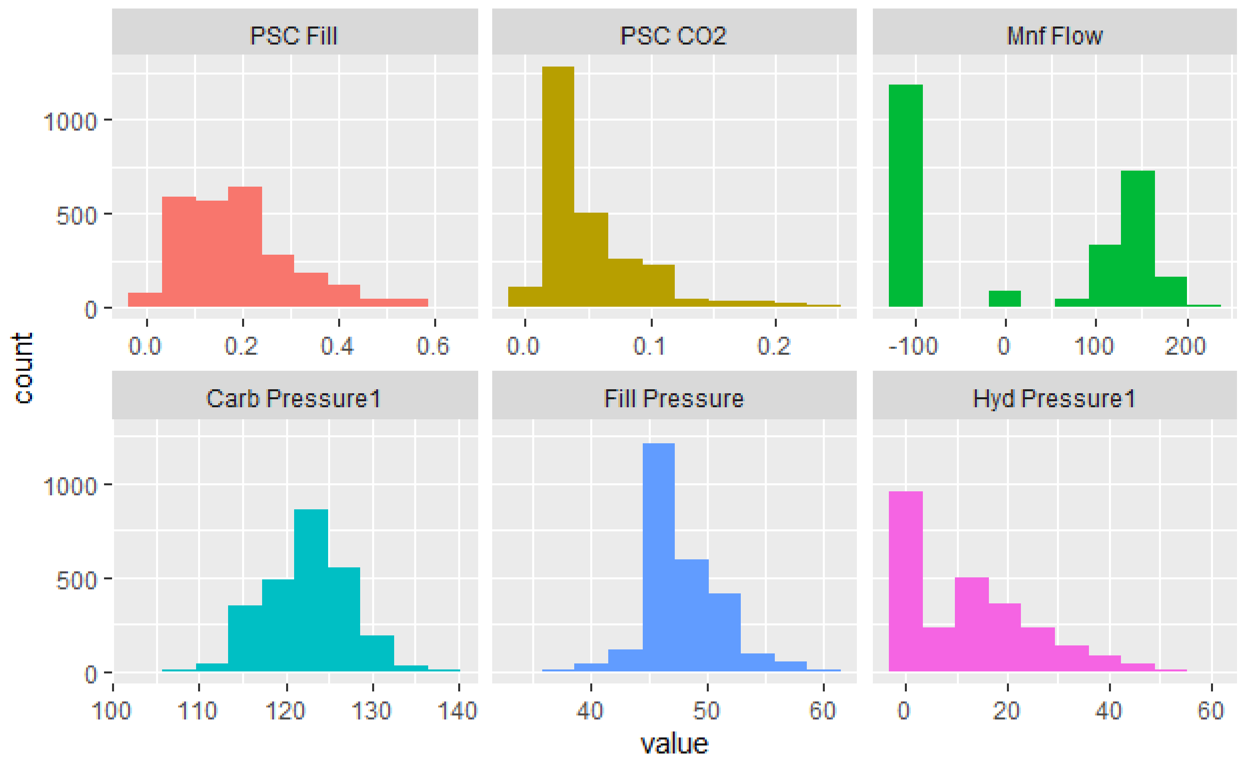
```
# Data visualization procedure
plot_num(train.df, bins = 10)
```

Output



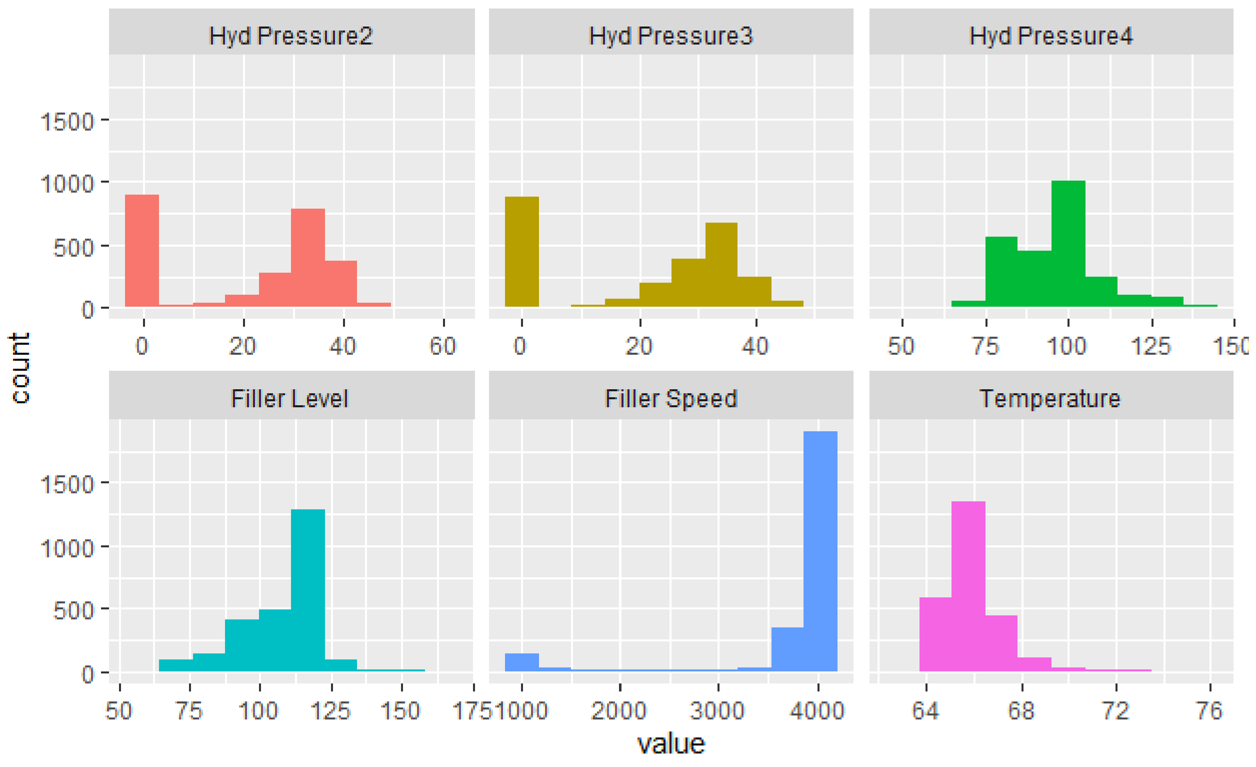
Interpretation

PSC has a **right skew;** all others in this group resembles a symmetrical distribution



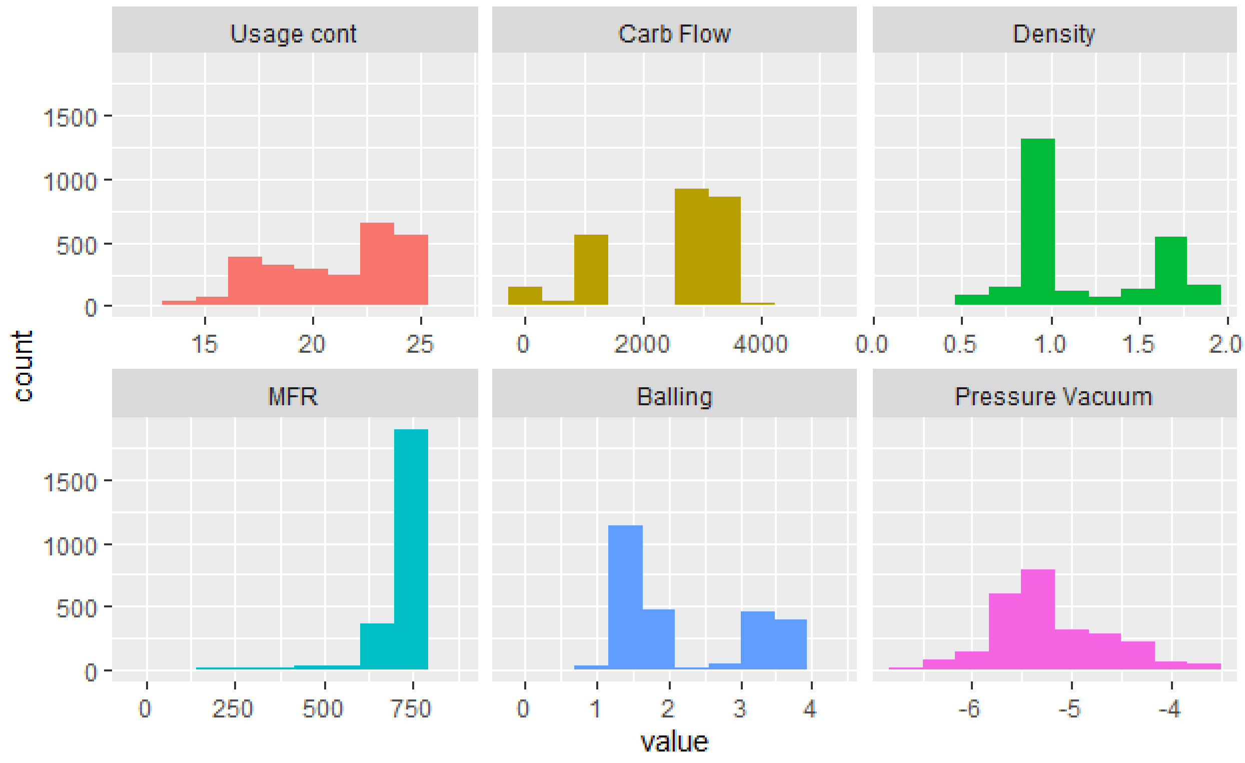
Interpretation

PSC Fill, PSC CO2 Hyd Pressure1 has a **right skew; Mnf Flow** seem to have some irregularities with large amounts of data points appearing to the left of zero, and a variable distribution to the right of zero (this is also known as a **multimodal shape**); **Carb Pressure1** and **Fill Pressure** both resemble a symmetrical distribution.



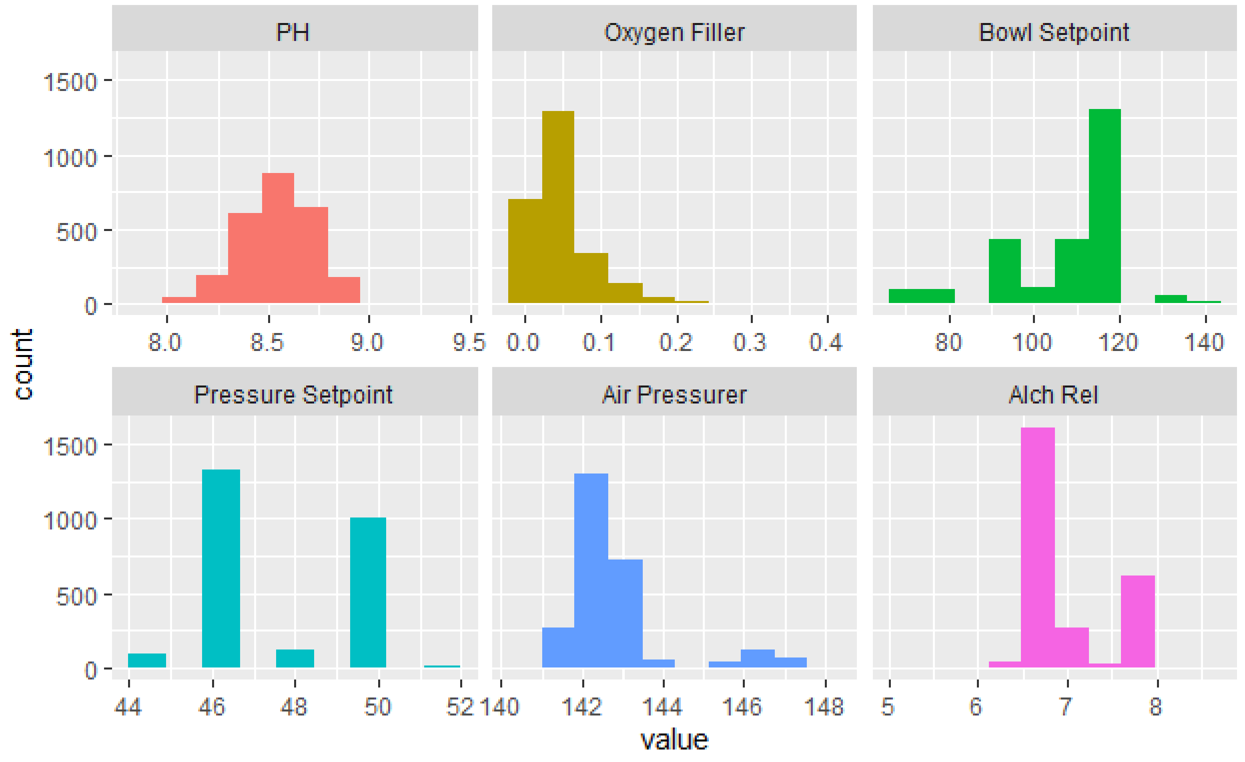
Interpretation

Hyd Pressure2 and **Hyd Pressure3** have a **multimodal** shape with a large percent of zero values, and a range of positive values between 0 and 50; **Hyd Pressure4** and **Temperature** seem to have a **right skew**; **Filter Level** and **Filter Speed** appear to have a skewed shape, but high variability makes this unclear.



Interpretation

Skew is present in **Usage cont (left skew), MFR** (**left skew**) and **Pressure Vacuum** (**right skew**). **Balling, Density** and **Carb Flow** appear to be **multimodal** and highly irregular.



Interpretation

pH (our target variable) appears to have a symmetrical distribution; **Oxygen Filler** and **Air Pressure** both exhibit a **right skew** and discrete values**; Pressure Seapoint** also contains discrete values that vary; **Bowl Setpoint** and **Alch Rel** show **left skew** but further analysis is needed to confirm this.

Fig 6 : R Code and output for generating histograms of columns in training set

Predictor Analysis: Brand Code

We generated a frequency table and histogram for the categorical variable **Brand Code** separately to better understand its distribution**.**

Code

```r
# Categorical variable visualizations.
ggplot(train.df, aes(x = `Brand Code`)) +
        geom_bar() +
        xlab("Brand Code") +
        ggtitle("Categorical variable analysis", subtitle = "Training data-
set") +
        theme(axis.text.x = element_text(angle = 45, hjust = 1),
             panel.background = element_rect(fill = "white", colour =
"grey50"))
```

Output

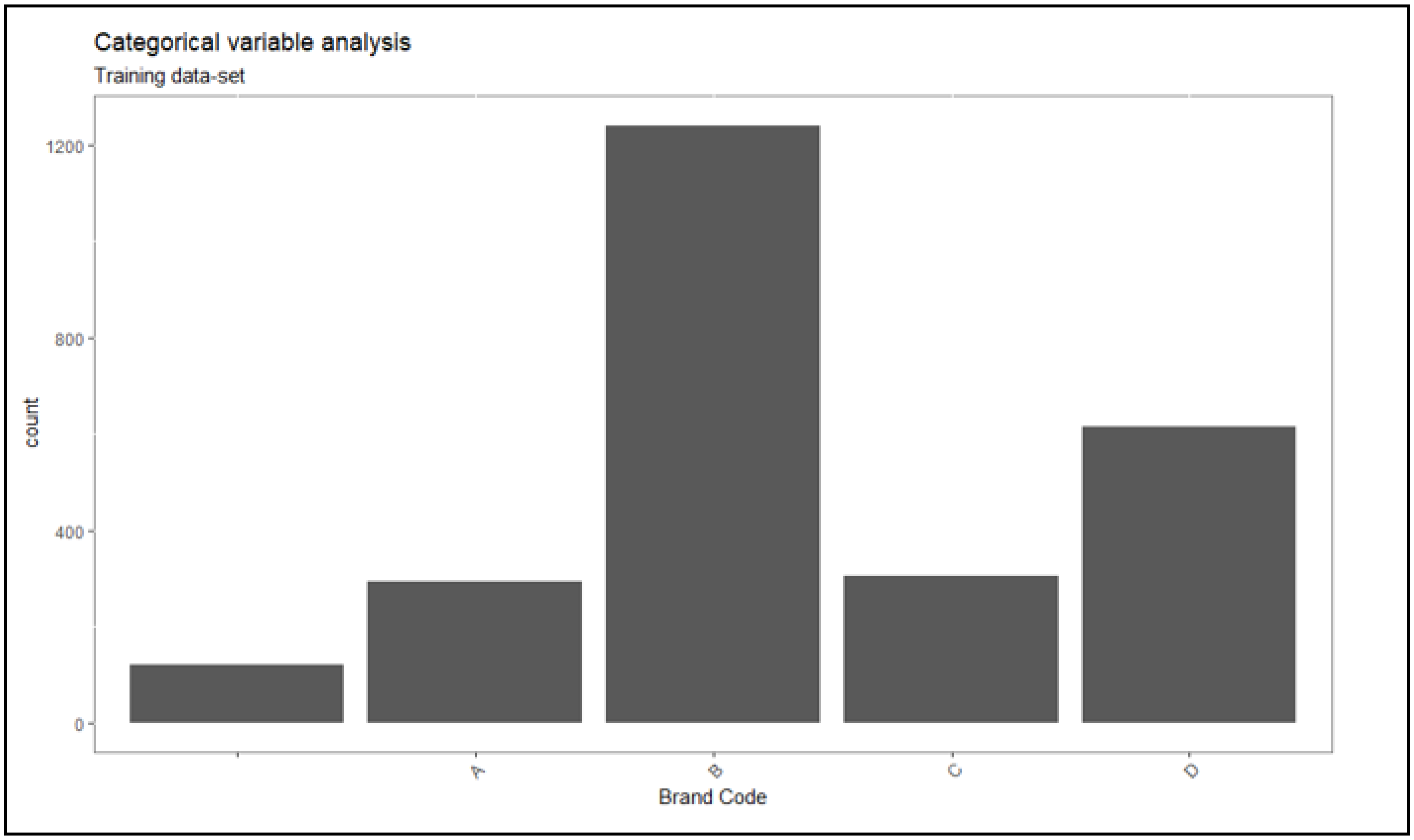


Fig 7 : R Code for generating histogram for Brand Code column

Interpretation

The histogram above [Fig 7] gives insight into the number of cases that fall into each **Brand Code.** We can see that most of the cases appear to fall in **Brand Code B** while fewest cases fall into the NA category (leftmost bar). These missing values will be imputed in our preprocessing stage.

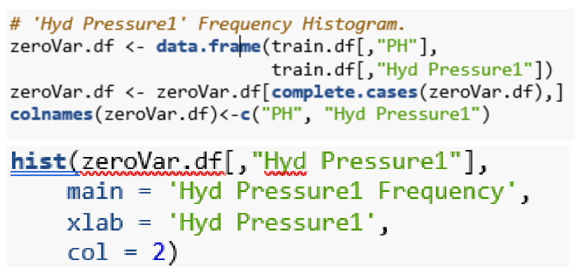
Non Zero Variance Predictors

During initial exploratory data analysis [Fig 6], we suspected that there may be predictors that have little or no variability, and thus provide no meaningful information about our target variable **(pH)**. We tested each predictor [Fig 8] and found a variable with near zero variance.

Code	**Output**

```
# Procedure to find near zero variance.
train.df.ZeroVar <- nearZeroVar(train.df, names = TRUE)
train.df.ZeroVar
```
	`## [1] "Hyd Pressure1"`

Code	**Output**



```
# 'Hyd Pressure1' Frequency Histogram.
zeroVar.df <- data.frame(train.df[,"PH"],
                         train.df[,"Hyd Pressure1"])
zeroVar.df <- zeroVar.df[complete.cases(zeroVar.df),]
colnames(zeroVar.df)<-c("PH", "Hyd Pressure1")
```

```
hist(zeroVar.df[,"Hyd Pressure1"],
    main = 'Hyd Pressure1 Frequency',
    xlab = 'Hyd Pressure1',
    col = 2)
```

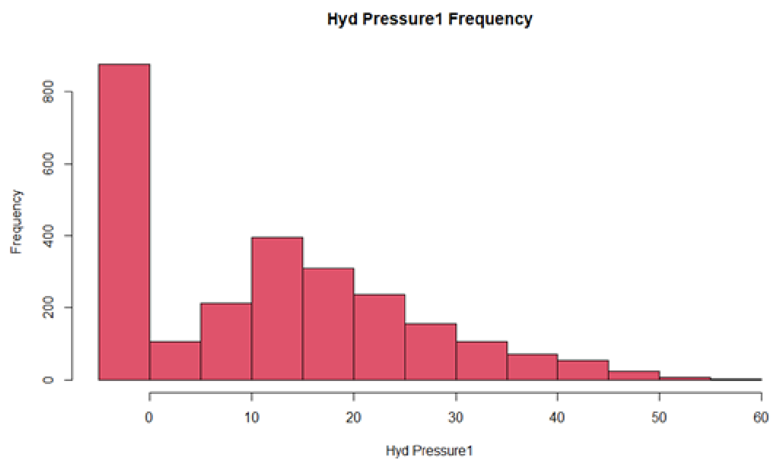


Fig 8 : R Code for identifying near zero predictors and plotting them

Interpretation

Hyd Pressure1 exhibits behavior of a predictor with a variance that is nearly zero. This means that many of the observations have the same value and therefore, do not provide meaningful information about our target variable. We elected to eliminate this variable from our data set.

Missing Value Analysis

Each column was analyzed for missing values by first calling the **aggr()** function from the **VIM** library. This allows us to visualize missing values via a histogram and aggregated plot as seen in Fig 9.

Code

```
#plot missing values using VIM package
aggr(train.df , col=c('navyblue','red'), numbers=TRUE, sortVars=TRUE,
labels=names(rawCSV), cex.axis=.7, gap=3, ylab=c("Histogram of missing
data","Pattern"))
```

Output

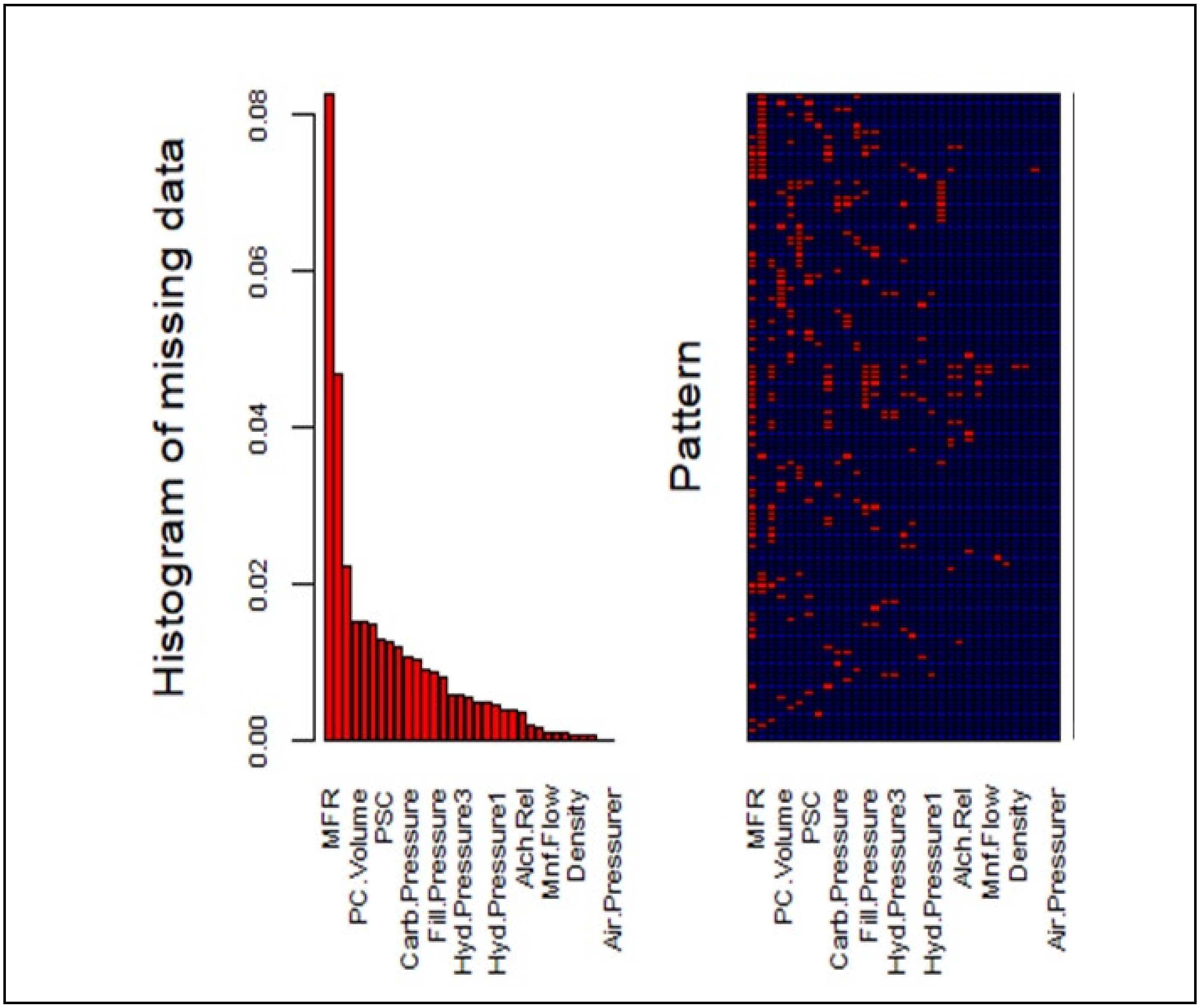


Fig 9 : R Code for generating histogram and plot for missing values

We also identified the fraction of missing values found in each predictor and placed them in a table.

Second Output
Fraction Missing Values (Sorted by Variable)

Variable	Fraction
MFR	0.0824581875
Brand Code	0.0466744457
Filler Speed	0.0221703617
PC Volume	0.0151691949
PSC CO2	0.0151691949
Fill Ounces	0.0147802412
PSC	0.0128354726
Carb Pressure1	0.0124465189
Hyd Pressure4	0.0116686114
Carb Pressure	0.0105017503
Carb Temp	0.0101127966
PSC Fill	0.0089459354
Fill Pressure	0.0085569817
Filler Level	0.0077790743
Hyd Pressure2	0.0058343057
Hyd Pressure3	0.0058343057
Temperature	0.0054453520

Variable	Fraction
Oxygen Filler	0.0046674446
Pressure Setpoint	0.0046674446
Hyd Pressure1	0.0042784909
Carb Volume	0.0038895371
Carb Rel	0.0038895371
Alch Rel	0.0035005834
Usage cont	0.0019447686
PH	0.0015558149
Mnf Flow	0.0007779074
Carb Flow	0.0007779074
Bowl.Setpoint	0.0007779074
Density	0.0003889537
Balling	0.0003889537
Balling Lvl	0.0003889537
Pressure Vacuum	0
Air Pressure	0

Fig 10 : Table of fraction of missing values for each column in training data set

Interpretation

From outputs in Fig 9 and Fig 10 we can clearly see that more than 5% of our total data set has missing values, with **MFR** missing the most values (approximately 8.25%), followed by the previously inspected categorical variable **Brand Code**. Imputation alternatives explored were random generation of values, mean replacement or k-neighbors.

Correlation Analysis

In order to gain some insight into the relationship between predictors, we performed correlation analysis between predictors, and between predictors and the target variable. While the basis of predictive analysis is to find correlated variables with the target variables, when multiple predictors are correlated --or multicollinear-- the relationship between individual predictors and the target become unclear. Furthermore the absence of multicollinearity is required for some models, such as OLS Regression. One standard technique to visualize linear correlations is through a heat map.

Correlation Between Predictors

A **heat map** was generated via the **vis_cor()** function from the **visdat** library to visualize correlations between predictors in the training set.

Code

```
# Procedure to visualize correlations from the training data-set.
vis_cor(train.df[,-1])
```

Output

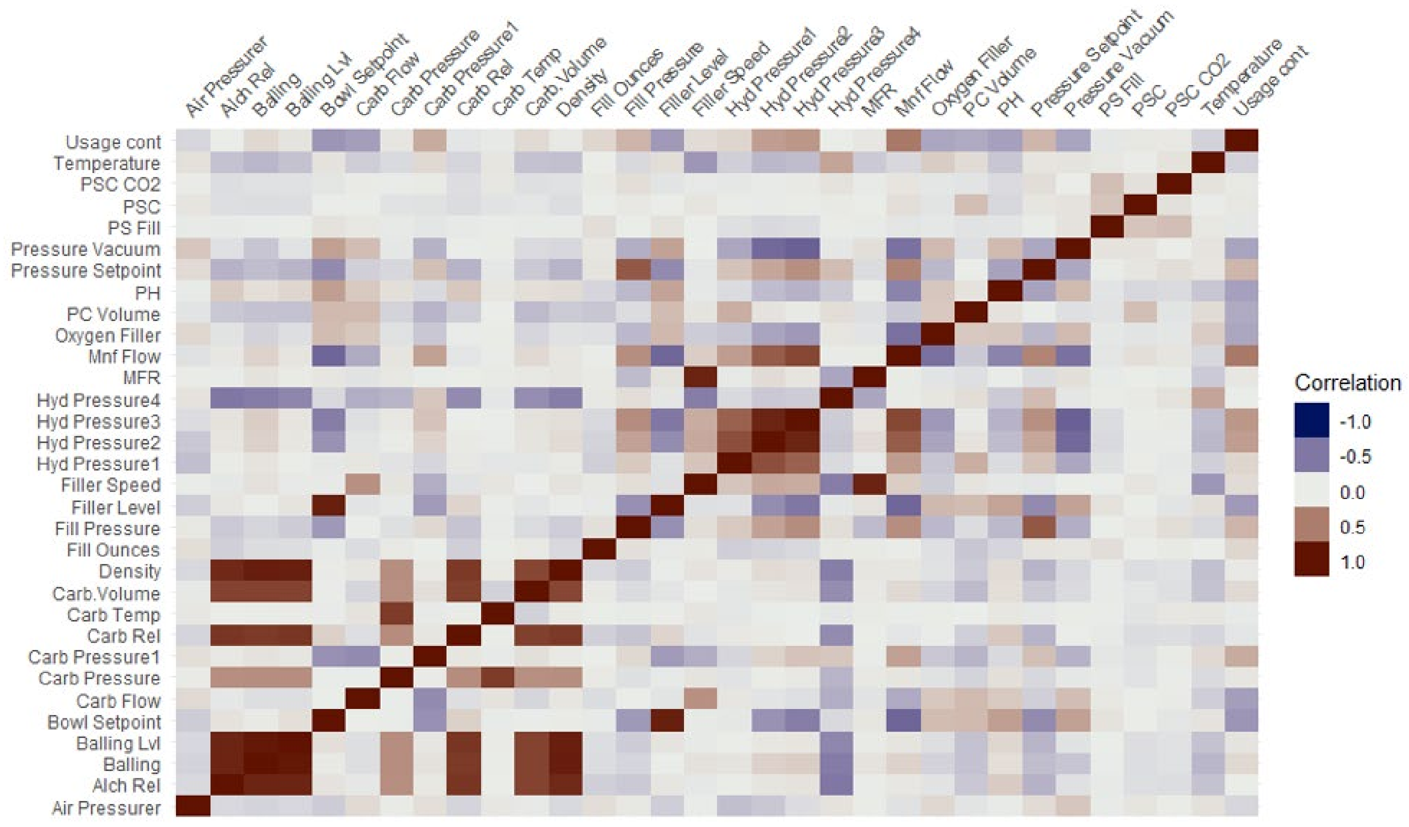


Fig 11 : R Code for generating heat map

Interpretation

The prevalence of the red blocks in Fig 11 tells us that many predictors are highly correlated. Below we identified highly correlated values for further analysis by using the **findCorrelation()** function from the **caret** library.

Code

```
# Finding correlation values.
train.df.correlations <- cor(train.df[,-1], use = "na.or.complete")
train.df.highlyCorrelated <- findCorrelation(train.df.correlations, .9,
                                             names = TRUE)
train.df.highlyCorrelated
```

Output

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

Fig 12 : R Code for identifying highly correlated predictors

Interpretation

The predictors that were identified in the output in Fig 12 are highly correlated to other predictors. Each of these were inspected more closely to identify which specific predictors they are correlated to [Fig 13]. We will inspect these further to determine how to address correlations before modeling.

Predictor Name	Highly Correlated to...
Balling	Density, Alch Rel & Balling Lvl
Hyd Pressure3	Hyd Pressure2
Alch Rel	Density, Balling & Balling Lvl
Balling Lvl	Density, Balling & Alch Rel
Filler Level	Bowl Setpoint
Filler Speed	MFR

Fig 13 : Table of highly correlated values

Correlation With Target Variable (pH)

Some predictors exhibited interesting behavior in their distributions to warrant further analysis. To get a better sense of how we should incorporate specific variables in training our model, we analyze their behavior with respect to the target variable (pH). Below are some interesting findings and assumptions we will be making as a result of this analysis.

Predictor Analysis: Mnf Flow

During the initial EDA [Fig 6], **Mnf Flow** exhibited irregularities with a high number of data points appearing to the left of zero, and a variable distribution to the right of zero (this is also known as a **multimodal shape**). This predictor was plotted against the target variable (pH) in Fig. 14.

Code

```r
# Select variable and target.
temp.df <- data.frame(train.df[,"PH"],
                      train.df[,"Mnf Flow"])

# Set column names
temp.df <- temp.df[complete.cases(temp.df),]
colnames(temp.df)<-c("PH", "Mnf Flow")

# Plot patterns.
plot(temp.df[,"PH"] ~ temp.df[,"Mnf Flow"],
    main = "'PH' levels based on 'Mnf Flow'",
    xlab = "Mnf Flow",
    ylab = "PH",
    col = 4)
```

Output

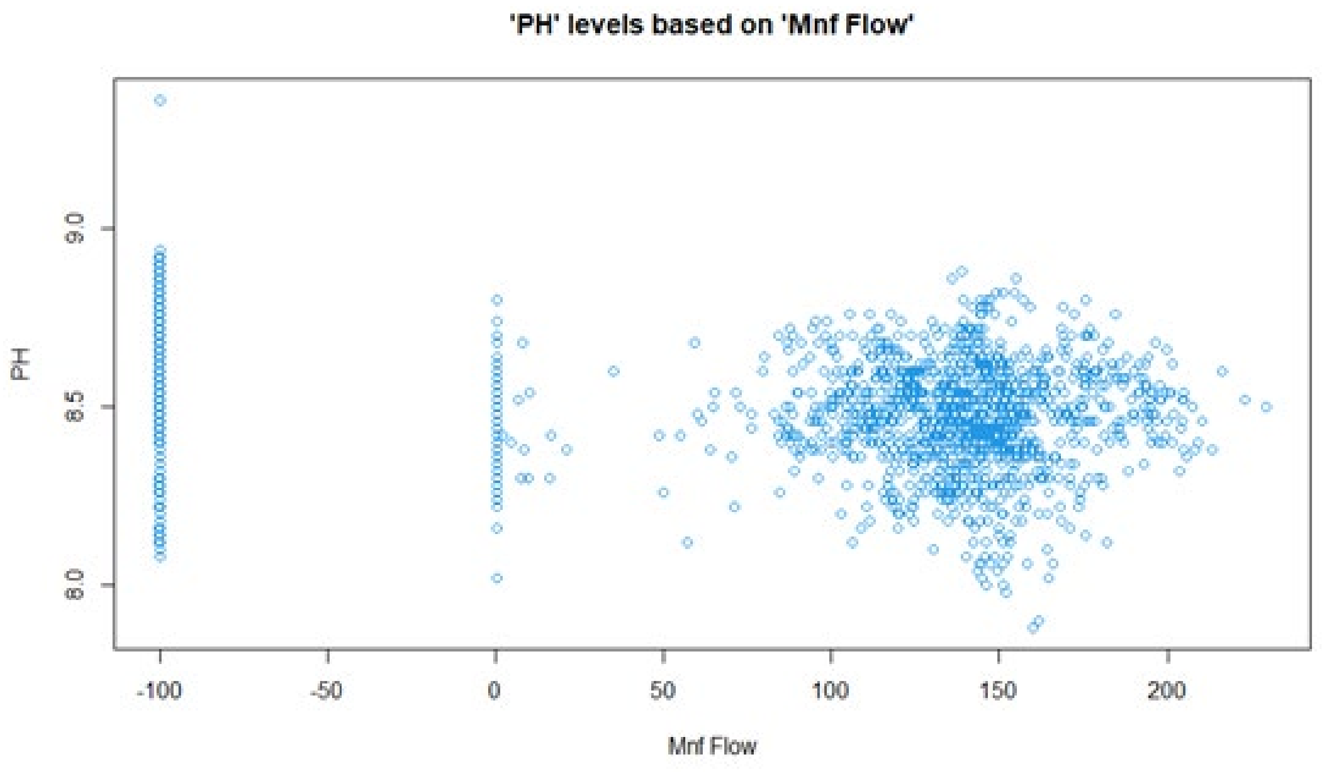


Fig 14 : R Code for plotting Mnf Flow vs pH

Interpretation

It is interesting to note negative values as seen in Fig 14; it does not seem to match the behavior when Mnf > 0 . We speculated that this odd behavior could have been due to NA values that were imputed as -100 in some other pre-processing step before we received this data. In the context of this problem, a negative flow rate has no practical significance and therefore, can present misleading information within this predictor. In this particular case, more than 46% of the data in this variable has problematic behavior that we cannot resolve without more information about how the data was gathered. For this reason we elected to remove this predictor from our training set.

Predictor Analysis:Hyd Pressure2 & Hyd Pressure3

During the initial EDA Fig 6, **Hyd Pressure2** exhibited irregularities with a large number of data points appearing at zero, and a variable distribution for values greater than zero (**multimodal shape**). **Hyd Pressure3** exhibited similar behavior and was highly correlated to **Hyd Pressure2** Fig 13. These predictors were plotted against the target variable (pH) to better understand it's behavior. Note: The code to generate the following visual is similar to the code presented in Fig 14 above. We have omitted it in the Fig 15 below.

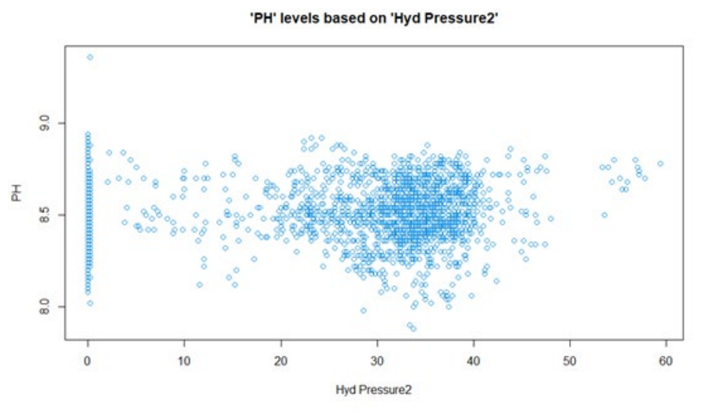
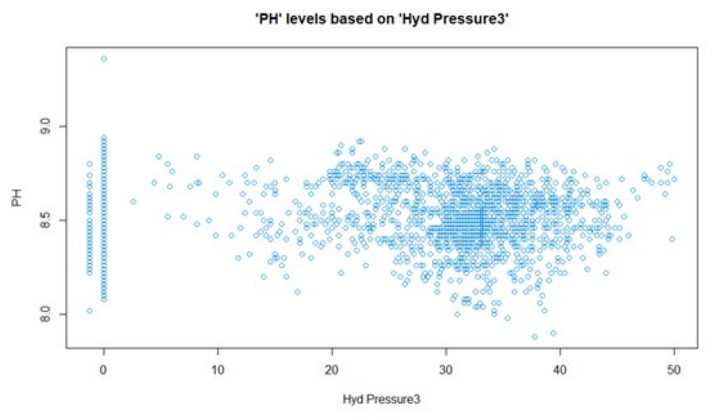
Output	
	

Fig 15 :Plotting Hyd Pressure2 vs pH & Hyd Pressure3 vs pH

Interpretation

Fig 15 shows that 34.3% of observations in the **Hyd Pressure2** and **Hyd Pressure3** predictors ranged from zero to other close values(0.2 and -1.2 respectively), which is noticeably different from the behavior of the rest of the data. Based on this evidence we determined that these predictors demonstrate problematic behavior and therefore cannot accurately explain one third of the pH values. We elect to remove them from the training data.

Predictor Analysis: Filler Speed & Carb Flow

During the initial EDA [Fig 6], **Filler Speed** exhibited irregularities and correlation with other predictors [Fig 13]; **Carb Flow** and **Pressure Setpoint** also showed some irregularities. Note: The code to generate the following visual is similar to the code presented in Fig 14 above. We have omitted it in the Fig 16 below.

Output

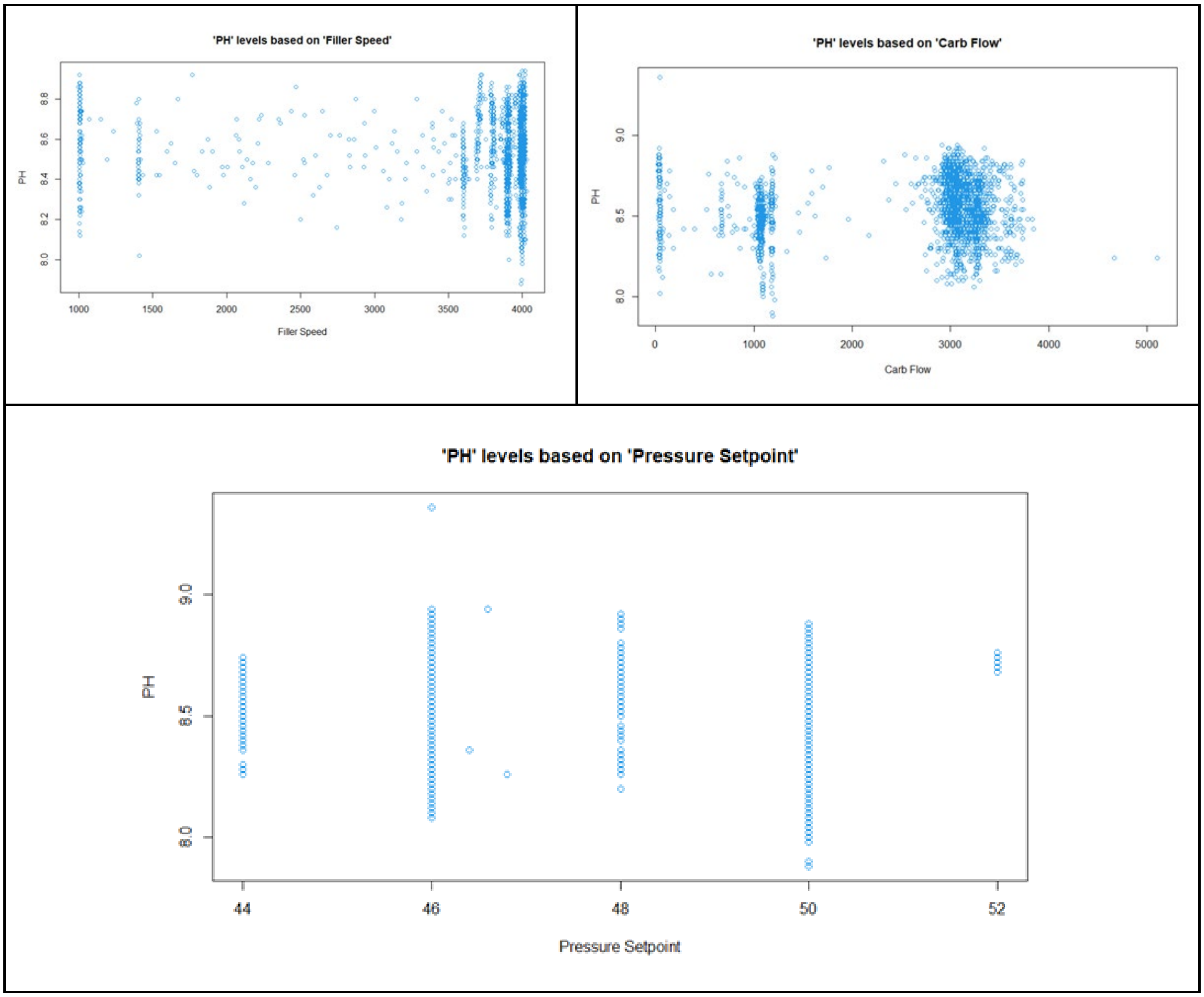


Fig 16 :Plotting Filler Speed vs pH, Carb Flow vs. pH and Pressure Setpoint

Interpretation

Fig 16 shows that **Filler Speed, Carb Flow** and **Pressure Setpoint** do not seem to follow any particular pattern with respect to the target variable (pH). It is interesting to note that **Pressure Setpoint** behaves in a similar manner to a discrete categorical variable (not continuous) which will help inform our imputation strategy for missing values. We assume irregularities are due to this random behavior and will include **Carb Flow** and **Pressure Setpoint** predictors in our training set; **Filler Speed** will be removed due to correlation with other predictors described previously.

Preprocessing

Now that we have a better understanding of the irregularities in expected behavior for each predictor, we can begin to preprocess or "prep" our data for training models. Preprocessing is necessary because the quality of the data used impacts how our model "learns" various features found in predictors that inform the target variable. Therefore, poor data quality will result in poor model quality and impact accuracy in a negative way. We begin our pre-processing by first eliminating predictors that exhibit irregular behavior, then we impute missing values using a **random sampling** approach, and end with scaling and centering each predictor. All preprocessing steps identified on the training data set will also be applied to the test data set.

Dimensionality Reduction

From the above analysis, we chose to remove the highly correlated predictors, near zero variance predictors, and predictors with significant irregularities as described above. This is called dimensionality reduction: reducing the number of predictors used to train models. The main goal of this preprocessing step is to reduce the complexity of our dataset, thereby reducing chances of overfitting. Reducing the number of dimensions used to train the model may also improve accuracy, reduce the time it takes to train models and the space required for computations, and remove redundant features. As Burk Soda has a limited operational budget for the fiscal year, capital allocation for computing needs to be justified.

Below is a table that summarizes the predictors that were removed from the data set along with rationale for removal:

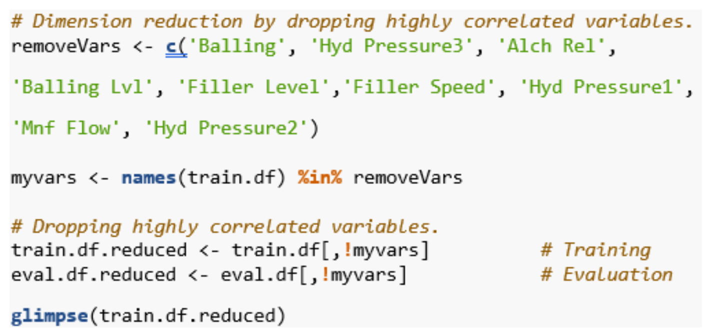
Variable (s)	**Rationale**
Mnf Flow	Unresolvable problematic behavior in 46% of data
Hyd Pressure2 & Hyd Pressure3	Unresolvable problematic behavior in 34.4% of data & Highly correlated
Hyd Pressure1	Non-zero variance
Balling, Alch Rel, Balling Lvl , Filler Level, Filler Speed	Highly correlated

Fig 17 : Predictors removed from training set

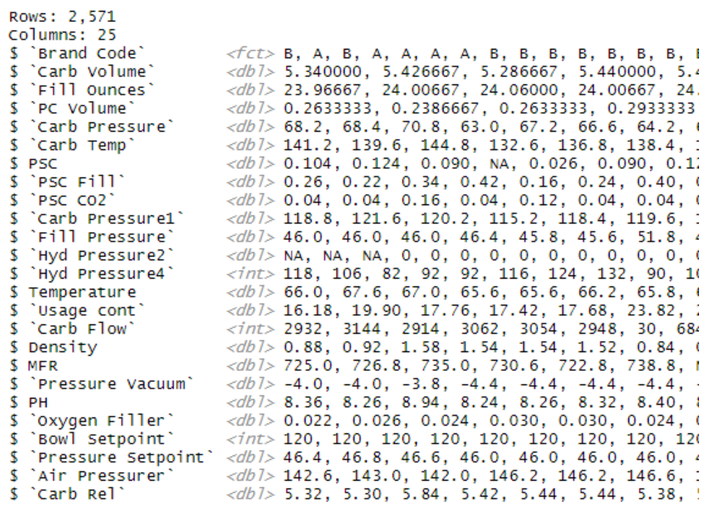
Approach

We removed these predictors by selecting column names and dropping them from our data set as seen in [Fig 18] below.

Code	**Output**



```r
# Dimension reduction by dropping highly correlated variables.
removeVars <- c('Balling', 'Hyd Pressure3', 'Alch Rel',

'Balling Lvl', 'Filler Level','Filler Speed', 'Hyd Pressure1',

'Mnf Flow', 'Hyd Pressure2')

myvars <- names(train.df) %in% removeVars

# Dropping highly correlated variables.
train.df.reduced <- train.df[,!myvars]          # Training
eval.df.reduced <- eval.df[,!myvars]            # Evaluation

glimpse(train.df.reduced)
```



```
Rows: 2,571
Columns: 25
$ `Brand Code`        <fct> B, A, B, A, A, A, A, B, B, B, B, B, B, B,
$ `Carb Volume`       <dbl> 5.340000, 5.426667, 5.286667, 5.440000, 5.
$ `Fill Ounces`       <dbl> 23.96667, 24.00667, 24.06000, 24.00667, 24.
$ `PC Volume`         <dbl> 0.2633333, 0.2386667, 0.2633333, 0.2933333
$ `Carb Pressure`     <dbl> 68.2, 68.4, 70.8, 63.0, 67.2, 66.6, 64.2,
$ `Carb Temp`         <dbl> 141.2, 139.6, 144.8, 132.6, 136.8, 138.4,
$ PSC                 <dbl> 0.104, 0.124, 0.090, NA, 0.026, 0.090, 0.1
$ `PSC Fill`          <dbl> 0.26, 0.22, 0.34, 0.42, 0.16, 0.24, 0.40,
$ `PSC CO2`           <dbl> 0.04, 0.04, 0.16, 0.04, 0.12, 0.04, 0.04,
$ `Carb Pressure1`    <dbl> 118.8, 121.6, 120.2, 115.2, 118.4, 119.6,
$ `Fill Pressure`     <dbl> 46.0, 46.0, 46.0, 46.4, 45.8, 45.6, 51.8,
$ `Hyd Pressure2`     <dbl> NA, NA, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
$ `Hyd Pressure4`     <int> 118, 106, 82, 92, 92, 116, 124, 132, 90, 1
$ Temperature         <dbl> 66.0, 67.6, 67.0, 65.6, 65.6, 66.2, 65.8,
$ `Usage cont`        <dbl> 16.18, 19.90, 17.76, 17.42, 17.68, 23.82,
$ `Carb Flow`         <int> 2932, 3144, 2914, 3062, 3054, 2948, 30, 68
$ Density             <dbl> 0.88, 0.92, 1.58, 1.54, 1.54, 1.52, 0.84,
$ MFR                 <dbl> 725.0, 726.8, 735.0, 730.6, 722.8, 738.8,
$ `Pressure Vacuum`   <dbl> -4.0, -4.0, -3.8, -4.4, -4.4, -4.4, -4.4,
$ PH                  <dbl> 8.36, 8.26, 8.94, 8.24, 8.26, 8.32, 8.40,
$ `Oxygen Filler`     <dbl> 0.022, 0.026, 0.024, 0.030, 0.030, 0.024,
$ `Bowl Setpoint`     <int> 120, 120, 120, 120, 120, 120, 120, 120, 12
$ `Pressure Setpoint` <dbl> 46.4, 46.8, 46.6, 46.0, 46.0, 46.0, 46.0,
$ `Air Pressurer`     <dbl> 142.6, 143.0, 142.0, 146.2, 146.2, 146.6,
$ `Carb Rel`          <dbl> 5.32, 5.30, 5.84, 5.42, 5.44, 5.44, 5.38,
```

Fig 18: R Code for dimension reduction

Interpretation

Fig 18 shows our reduced data-set has a total of 24 predictors plus our target variable (pH).

Imputing Missing Data

Fig 9 showed that there are various columns that contain missing values, however the number of missing values, about 1.2% of the total data, was relatively low for non-categorical predictors. Thus we elected to apply random value imputing, which replaces missing values with random values within the data range as you will see after imputing the distribution doesn't change dramatically validating the method and simplifying the approach.

Brand Code

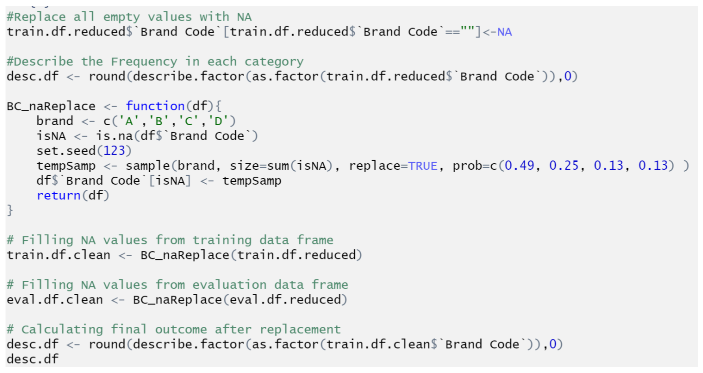
Preliminary EDA showed that Brand Code was a categorical variable with 4 categories: A, B, C and D (Fig?). There were a significant amount of missing values from this predictor and imbalanced categories(Fig?) therefore, we elected to randomly assign categories to missing variables so that the proportion of data in each category did not change. First we established proportions for each category as seen in the table below.

	A	**B**	**C**	**D**
Count	293	1239	304	615
Percent	11	48	12	24

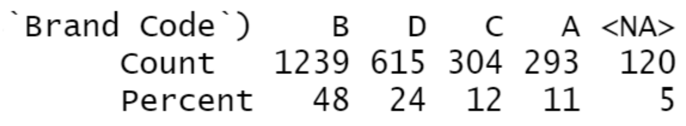
Fig 19: Itemized Brand Code Categories

The following R code randomly assigns categories to missing values, ensuring that values are evenly distributed randomly throughout all categories.

Code	**Output (before imputing)**

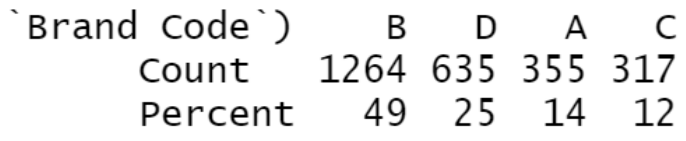


```
#Replace all empty values with NA
train.df.reduced$`Brand Code`[train.df.reduced$`Brand Code`==""]<-NA

#Describe the Frequency in each category
desc.df <- round(describe.factor(as.factor(train.df.reduced$`Brand Code`)),0)

BC_naReplace <- function(df){
    brand <- c('A','B','C','D')
    isNA <- is.na(df$`Brand Code`)
    set.seed(123)
    tempSamp <- sample(brand, size=sum(isNA), replace=TRUE, prob=c(0.49, 0.25, 0.13, 0.13) )
    df$`Brand Code`[isNA] <- tempSamp
    return(df)
}

# Filling NA values from training data frame
train.df.clean <- BC_naReplace(train.df.reduced)

# Filling NA values from evaluation data frame
eval.df.clean <- BC_naReplace(eval.df.reduced)

# Calculating final outcome after replacement
desc.df <- round(describe.factor(as.factor(train.df.clean$`Brand Code`)),0)
desc.df
```



```
`Brand Code`)     B    D    C    A  <NA>
       Count   1239  615  304  293   120
       Percent   48   24   12   11     5
```

Output (after imputing)



```
`Brand Code`)     B    D    A    C
      Count    1264  635  355  317
      Percent    49   25   14   12
```

Fig 20 :R Code for imputing train and evaluation set (Brand Code Predictor)

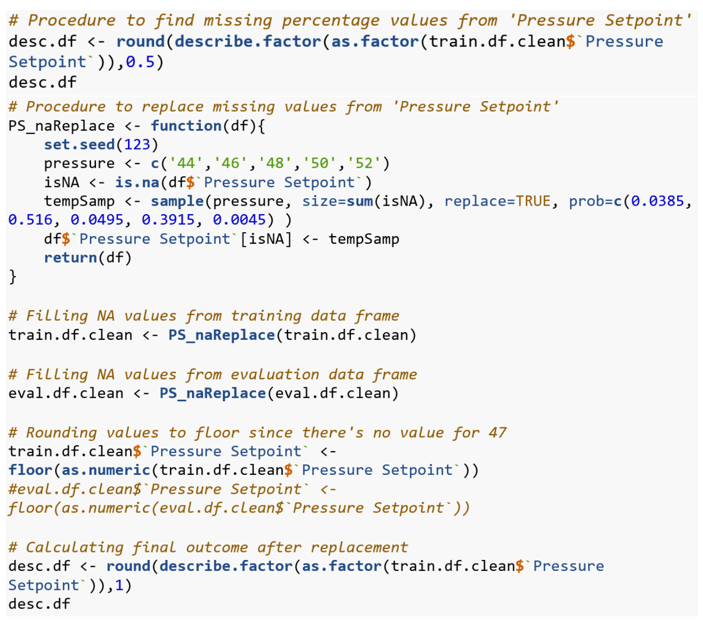
Interpretation

As seen in the output in Fig 20 after NA values are imputed the percent of each category increases uniformly, thereby keeping the shape of the data consistent.

Pressure Setpoint

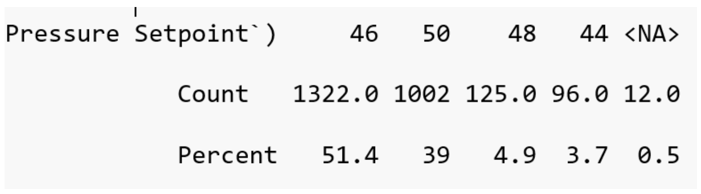
Fig 6 shows that **Pressure Setpoint** appears to exhibit discrete behavior similar to a categorical variable.The number of missing values is relatively small (0.5%) as seen in Fig 21 below. Therefore we elected to use the same random imputing method we employed for the **Brand Code** predictor.

Code



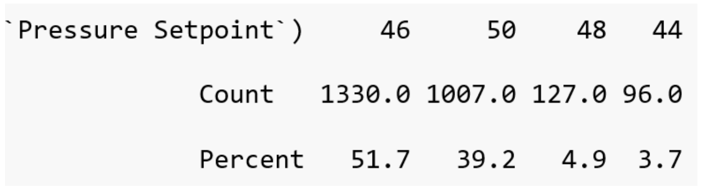
```
# Procedure to find missing percentage values from 'Pressure Setpoint'
desc.df <- round(describe.factor(as.factor(train.df.clean$`Pressure
Setpoint`)),0.5)
desc.df
# Procedure to replace missing values from 'Pressure Setpoint'
PS_naReplace <- function(df){
    set.seed(123)
    pressure <- c('44','46','48','50','52')
    isNA <- is.na(df$`Pressure Setpoint`)
    tempSamp <- sample(pressure, size=sum(isNA), replace=TRUE, prob=c(0.0385,
0.516, 0.0495, 0.3915, 0.0045) )
    df$`Pressure Setpoint`[isNA] <- tempSamp
    return(df)
}

# Filling NA values from training data frame
train.df.clean <- PS_naReplace(train.df.clean)

# Filling NA values from evaluation data frame
eval.df.clean <- PS_naReplace(eval.df.clean)

# Rounding values to floor since there's no value for 47
train.df.clean$`Pressure Setpoint` <-
floor(as.numeric(train.df.clean$`Pressure Setpoint`))
#eval.df.clean$`Pressure Setpoint` <-
floor(as.numeric(eval.df.clean$`Pressure Setpoint`))

# Calculating final outcome after replacement
desc.df <- round(describe.factor(as.factor(train.df.clean$`Pressure
Setpoint`)),1)
desc.df
```

Output (Before)



```
Pressure Setpoint`)      46     50     48    44  <NA>

            Count    1322.0 1002 125.0 96.0  12.0

            Percent    51.4   39   4.9  3.7   0.5
```

Output (After)



```
`Pressure Setpoint`)      46      50     48    44

              Count    1330.0 1007.0 127.0 96.0

              Percent    51.7   39.2    4.9  3.7
```

Fig 21: R Code for identifying categorical behavior of and NA values for Pressure Setpoint

Interpretation

As you can see from Fig 21, before the procedure to replace missing values was implemented they fell in one of five categories: 46, 50, 58, 44 and NA. After the procedure we see that all NA values are randomly assigned a value evenly among 4 categories.

All Other Predictors

Based on the behavior of all other predictors and the fact that they contained only a very small percentage of missing values, we chose to impute with random values within the range of each predictor. The function below allowed us to automate this process and apply to all predictors.

Code

```
# Procedure to replace NA with automated random value
P_naReplace <- function(df){
    cPred <- dim(df)[2]
    for (i in 2:cPred){
        isNA <- is.na(df[,i])
        minP <- min(df[,i], na.rm = TRUE)
        maxP <- max(df[,i], na.rm = TRUE)
        set.seed(123)
        tempSamp <- round(runif(sum(isNA), min=minP, max=maxP),2)
        df[isNA,i] <- tempSamp
    }
    return(df)
}
```

Fig 22: R Code for random imputation procedure for all remaining variables

Interpretation

This procedure selects each column, identifies the minimum and maximum values, and replaces missing values with a random number within that range. Below we confirm that all NA values are resolved by generating an aggrplot [Fig23].

Code

```
aggr(train.df.clean3, col=c('navyblue','red'), numbers=TRUE,
sortVars=TRUE, labels=names(selCols), cex.axis=.7, gap=3,
ylab=c("Histogram of missing data","Pattern"))
```

Output

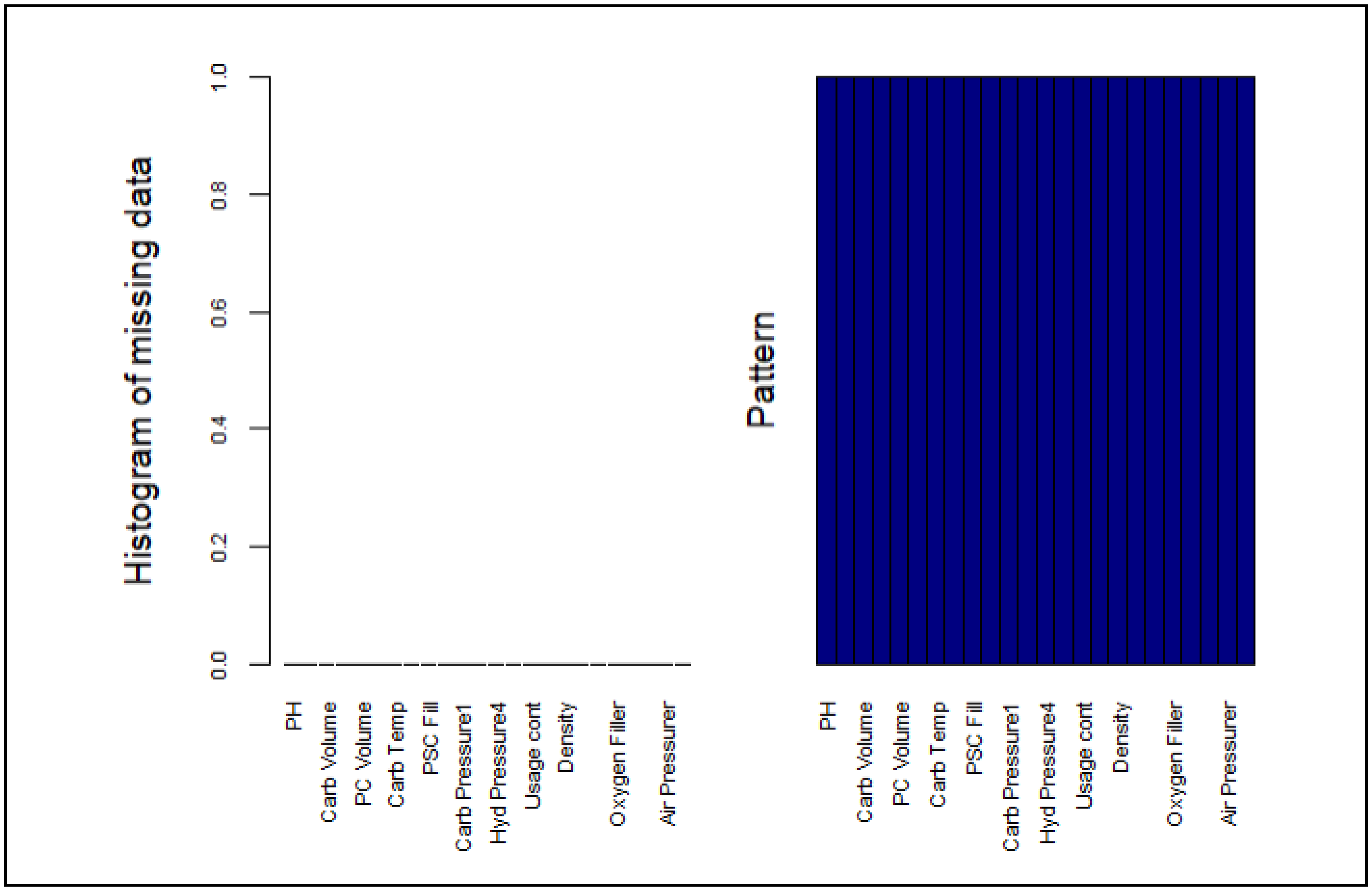


Fig 23 : R Code for generating histogram and plot for missing values

Center and Scale

Due to the various shapes of each predictor as described in Fig 6, it is necessary to center and scale each predictor before training a model. To do this we used the **prePrcoess()** function from the **caret** library in R as seen in Fig? Below.

Code

```r
# Reset Variables
train.df.clean<-train.df.clean3

# Find mean and standard deviation (mu and sigma) to revert the preprocess
ph.mu <- mean(train.df.clean$PH)
ph.sigma <- sd(train.df.clean$PH)

# Pre-process function
trans <- preProcess(train.df.clean, method = c("center", "scale"))
trans
```

Output

```
## Created from 2571 samples and 24 variables
##
## Pre-processing:
##   - centered (23)
##   - ignored (1)
##   - scaled (23)
```

Visualization

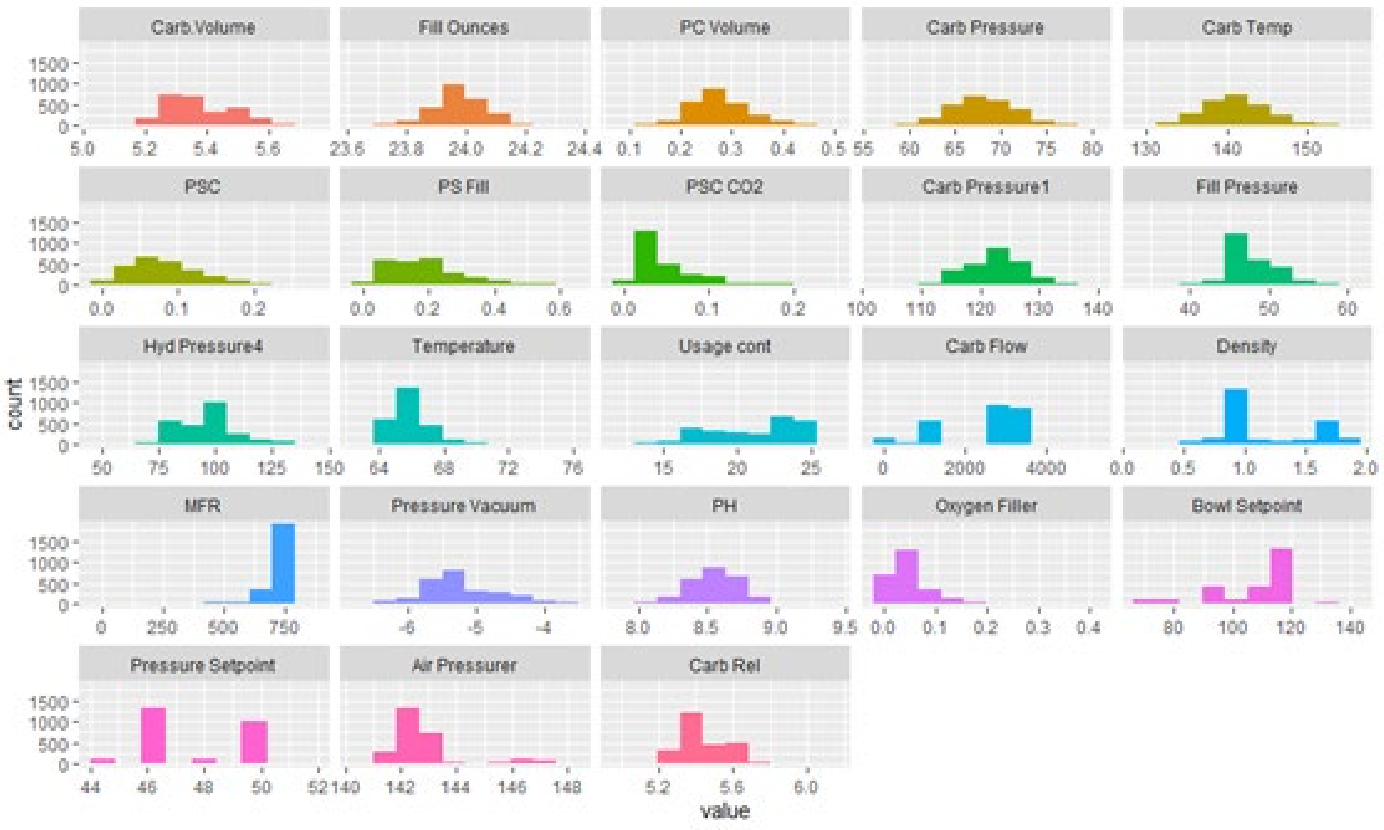


Fig 24: R Code for centering, scaling and visualization

Interpretation

We can clearly see that 23 out of the 24 variables in the training data required some scaling and centering. A final visualization of each variable confirmed that the distributions are normalized, scaled and ready for modeling [Fig 24]. We will continue to analyze the behavior of our variables to determine the best models to train and test.

Post Processing Analysis of Predictors

Methods including K-Means Clustering and Principal Component Analysis were used to determine the types of models we will choose to train and test for this data set.

K-means Clustering Analysis

K-means clustering attempts to find an optimal number of **clusters** (or groups of similar characteristics) within the data set. This process is performed by utilizing the **elbow method** and via calculations of the **within-clusters sum of squares** using the **kmeans()** function from the **stats** R library.

Code

```r
# K-means Clustering Analysis.
# Using train.df.transformed
train_df_trans <- subset(train.df.transformed, select = -`Brand Code`)
set.seed(123)
wcss = vector()
for (i in 1:10) {wcss[i] = sum(kmeans(train_df_trans, i)$withinss)}
plot(1:10, wcss, type = 'b',  main = paste('Elbow Method'), xlab =
'clusters', ylab = 'WCSS')
```

Output

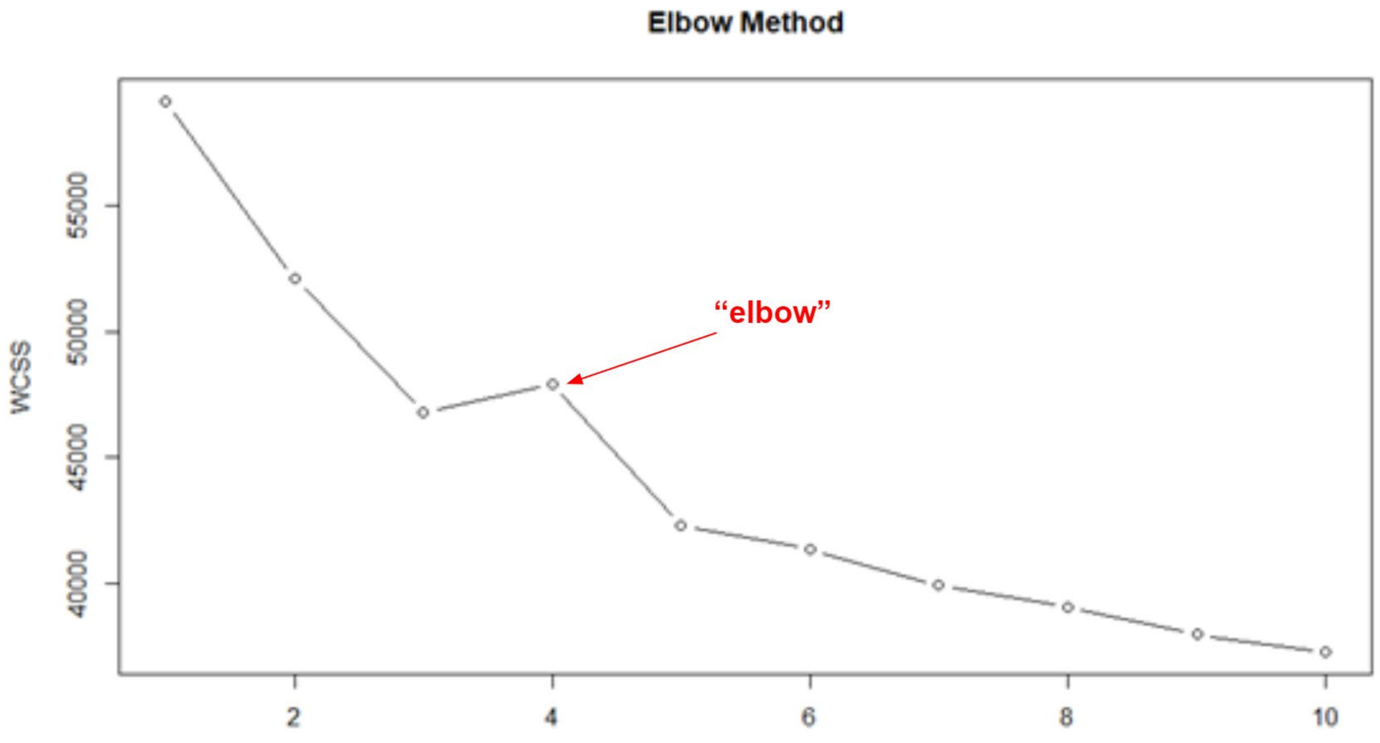


Fig 25: Kmeans clustering elbow method

Interpretation

The largest bend in the visualization above appears to be at cluster 3. Let us evaluate the 3 clusters in our data set and further investigate its granular details.

Code	Output

```r
# K-means Clustering Analysis - Granular details.
set.seed(123)
kmeans = kmeans(x = train_df_trans, centers = 3)
y_kmeans = kmeans$cluster
t(kmeans$centers)
```

```
##                               1            2           3
## PH                 0.3583553909  0.255127700 -0.63888983
## Carb Volume       -0.5850009055  1.180583357 -0.39723508
## Fill Ounces       -0.0484037884 -0.120161746  0.16357929
## PC Volume          0.3638163885 -0.246580563 -0.19269185
## Carb Pressure     -0.2938431017  0.651040121 -0.25186724
## Carb Temp          0.0115457423  0.042440846 -0.05144381
## PSC                0.0378338482 -0.123748376  0.06843128
## PSC Fill           0.0225259170 -0.004389614 -0.02173976
## PSC CO2            0.0004283861 -0.093875059  0.08416548
## Carb Pressure1    -0.4729073314  0.071787717  0.47476974
## Fill Pressure     -0.4613715844 -0.281033078  0.77977445
## Hyd Pressure4      0.2580265812 -0.808996772  0.43516913
## Temperature        0.2029809016 -0.315279771  0.05274508
## Usage cont        -0.5475820768  0.024994023  0.60215820
## Carb Flow          0.3212910369 -0.051853450 -0.31977779
## Density           -0.5884822434  1.323020070 -0.52170936
## MFR                0.0209751598  0.054997328 -0.07352428
## Pressure Vacuum    0.5192661147 -0.097919269 -0.50409237
## Oxygen Filler      0.4300095308 -0.096802623 -0.40327289
## Bowl Setpoint      0.6878377342  0.119309382 -0.89229506
```

```
## Pressure Setpoint -0.5140620437 -0.416971819  0.96247140
## Air Pressurer      0.0013945614 -0.125724172  0.11178398
## Carb Rel          -0.4560368575  1.249054357 -0.60610638
```

Fig 26 : KMeans analysis for each cluster per variable

Interpretation

As seen in Fig 26 the values in each column correspond to the mean of a particular predictor (ie **Carb Volume**, **Fill Ounces,** etc) whose values fall into a specific cluster (1, 2, and 3). We cannot discern the differences between clusters by simply looking at the values. After analyzing these further no additional insight was gained. This may indicate that certain variables are linearly correlated while others are not, which is proving difficult to resolve. Given boundaries between variables were not clearly identified, we tried a different approach: Principal Component Analysis (PCA).

Principal Component Analysis

Principal Component Analysis (PCA) is a method used to reduce the dimensions of a data set into linearly independent components, or features, that are linear combinations of original predictors. PCA was applied to all predictors except for Brand Code. This variable will be inspected after PCA is done.

Code

```r
# Principal Component
# Reference: https://www.datacamp.com/community/tutorials/pca-analysis-
r#interpret
df.train.pca <- prcomp(train.df.transformed[,-2], center = TRUE, scale. =
TRUE)
summary(df.train.pca)
```

Output (formatted)

Components	Proportion of Variance	Components	Proportion of Variance

PC1	0.1523	PC11	0.033663
PC2	0.1316	PC12	0.03292
PC3	0.08303	PC13	0.03057
PC4	0.0713	PC14	0.03007
PC5	0.05918	PC15	0.0273
PC6	0.05589	PC16	0.02227
PC7	0.05323	PC17	0.0214
PC8	0.04434	PC18	0.01941
PC9	0.0426	PC19	0.01522
PC10	0.04018	PC20	0.01244

Fig 27: PCA Output - Measure of Proportion and Variability (PC20 - PC23 Not shown)

Interpretation

Fig 27 shows that there are 23 principal components/ features derived from the PCA algorithm. The first two components account for approximately 28.12% of the variance in the data. To gain a better understanding of the behavior of the first two principal components, we plotted the first principal component on the x-axis and the second principal component on the y-axis using the **ggbiplot()** function from the **ggbiplot** library. Data points that fall into Brand Code categories and predictor axes are also labeled as well.

Code

```
ggbiplot(df.train.pca, alpha = 0.2, obs.scale = 1, var.scale = 1,
  groups = train.df.transformed$`Brand Code`, ellipse = TRUE, circle = FALSE)
+
  scale_color_discrete(name = '') +
  theme(legend.direction = 'horizontal', legend.position = 'top')
```

Output

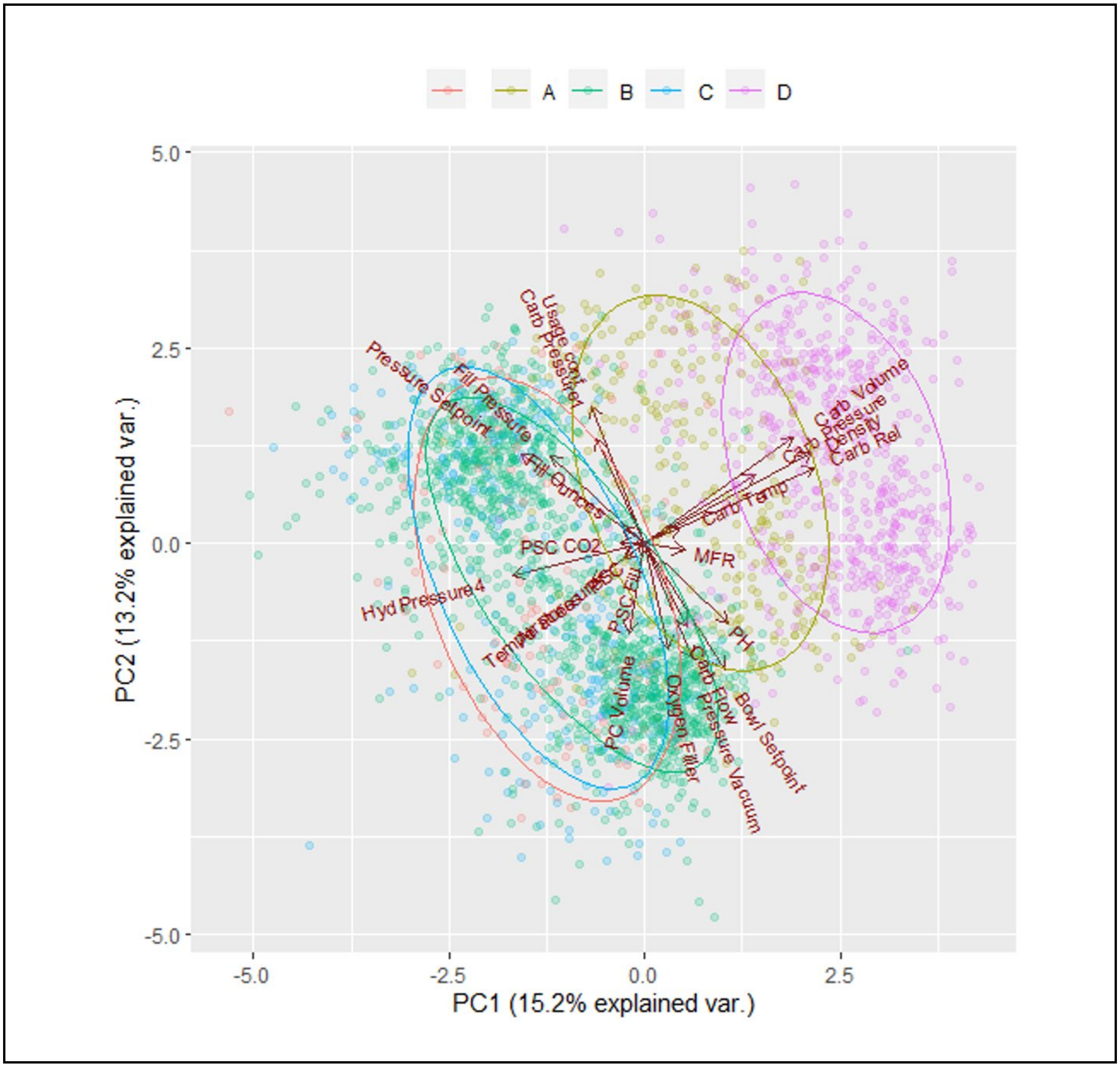


Fig 28 : PCA Visualization (PC1 and PC2)

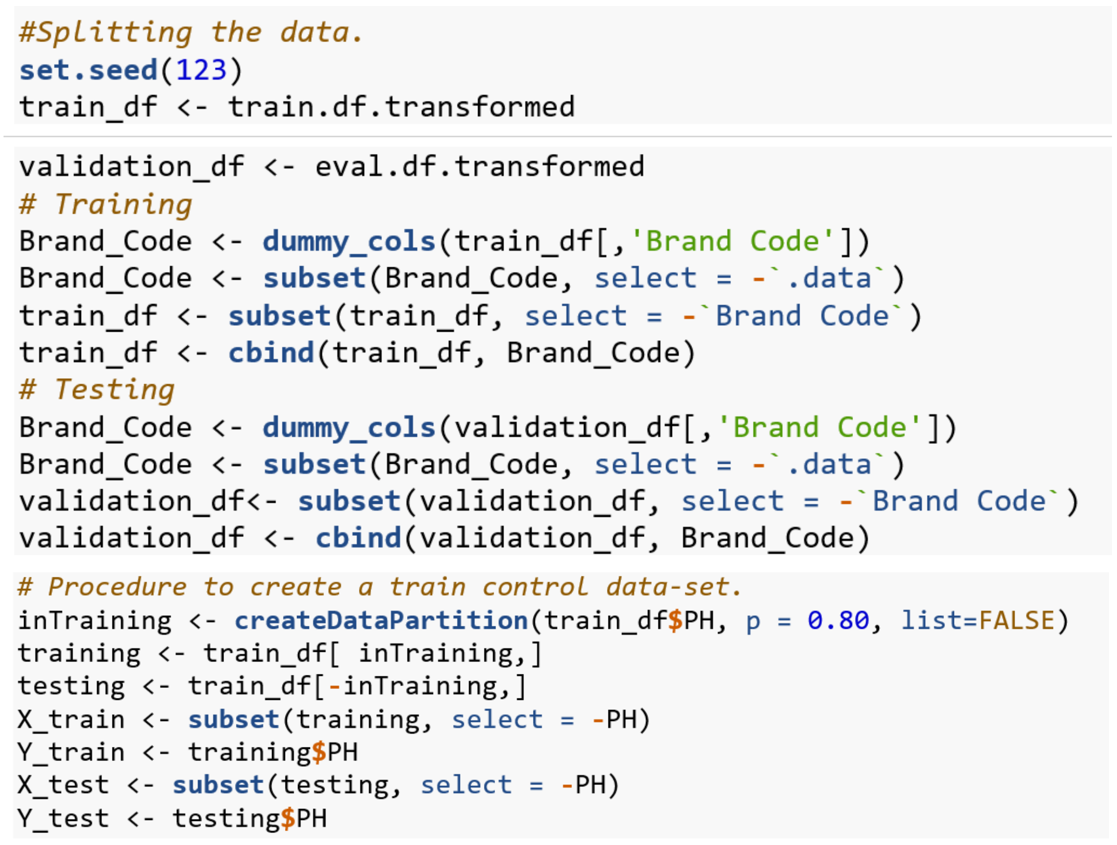
Interpretation

The arrow labels represent "contributive vectors" in that they show to what extent each original predictor contributes to each principal component. We can see that **Carb Volume, Carb Pressure, Density, Carb Rel,** and **Carb Temp** all have significant contributions to the first component (PC1), whereas **Carb Pressure, Pressure Vacuum, Carb Flow,** and **Oxygen Filler** contribute to the most to the second component (PC2). We also see some clustering of data points in different **Brand Code** categories. It appears that beverages in **Brand Code D** are associated with higher Carb Volume, Density, Carb Pressure and Carb Temp, whereas **Brand Code B** and **Brand Code C** are associated with high **Fill Ounce, Fill Pressure** and **Hyd Pressure4**. In fact, **Brand Code B** and **Brand Code C** appear to have many similarities which agrees with our K-means clustering analysis which identified 3 primary clusters, and are indistinguishable from each other.

Model Fitting

Group 4 chose to fit the datasets to the following models: **Linear Regression**, **Random Forest,** and **Gradient Boosting Machine** (through the **Xgboost** package**).** Following best practices in modeling, the datasets were split into test/train segments (via the **caret** package) and categorical variables were **one hot key encoded** (via **fastDummies** library) as seen in Fig 29 below.

Code



```
#Splitting the data.
set.seed(123)
train_df <- train.df.transformed

validation_df <- eval.df.transformed
# Training
Brand_Code <- dummy_cols(train_df[,'Brand Code'])
Brand_Code <- subset(Brand_Code, select = -`.data`)
train_df <- subset(train_df, select = -`Brand Code`)
train_df <- cbind(train_df, Brand_Code)
# Testing
Brand_Code <- dummy_cols(validation_df[,'Brand Code'])
Brand_Code <- subset(Brand_Code, select = -`.data`)
validation_df<- subset(validation_df, select = -`Brand Code`)
validation_df <- cbind(validation_df, Brand_Code)

# Procedure to create a train control data-set.
inTraining <- createDataPartition(train_df$PH, p = 0.80, list=FALSE)
training <- train_df[ inTraining,]
testing <- train_df[-inTraining,]
X_train <- subset(training, select = -PH)
Y_train <- training$PH
X_test <- subset(testing, select = -PH)
Y_test <- testing$PH
```

Fig 29 : R Code for 80/20 train test split and initializing

Interpretation

Since **Brand Code** was a categorial (non-numerical) variable, **fastDummies** converts the single column of “A, B, C, D” into 3 columns labeled A, B and C respectively. These columns are populated with either a 0 or a 1; 0 indicates that the row does not belong to that variable, and 1 indicates that it does. This is also known as **one hot key encoding**; for example, an observation with **Brand Code** = ‘A” would have a 1 in the new column marked “A” and 0s for “B” and C”. This technique relies on a binary encoding for categorical variables, therefore eliminating the possibility of bias being introduced into our model due to perceived ranking order within variables.

The **creatDataPartition()** function from the **caret** library splits it into two subsets: one subset contains 80% of the observations and is used for **training** the models, and the remaining 20% is used to evaluate model performance. Both datasets are split into the X, **predictor variables** and Y, the target variable pH. Seeds were set to ensure repeatability of results.

Multiple Linear Regression

Ordinary Least Squares Regression (OLS) is a classical linear model that uses one variable to predict another. Multiple Linear Regression (MLR) is an equivalent method however uses multiple variables to predict a target variable; MLR is used here as a baseline due to simplicity.

Code

```r
lm_mod <- lm(PH ~ .,
             data = training,
             seed = 29)

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...)
:
##  extra argument 'seed' will be disregarded

summary(lm_mod)
```

Output

```
##
## Call:
## lm(formula = PH ~ ., data = training, seed = 29)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.95037 -0.50863  0.09951  0.55702  2.77422

##
## Coefficients: (1 not defined because of singularities)
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         0.1578954  0.0706466   2.235 0.025525 *
## `Carb Volume`      -0.0691392  0.0387575  -1.784 0.074590 .
## `Fill Ounces`      -0.0477303  0.0185554  -2.572 0.010173 *
## `PC Volume`        -0.0690604  0.0208754  -3.308 0.000955 ***
## `Carb Pressure`     0.0093801  0.0461560   0.203 0.838979
## `Carb Temp`        -0.0004649  0.0424302  -0.011 0.991258
## PSC                -0.0297062  0.0189988  -1.564 0.118071
## `PSC Fill`         -0.0162649  0.0188396  -0.863 0.388055
## `PSC CO2`          -0.0357103  0.0180753  -1.976 0.048331 *
## `Carb Pressure1`    0.1235736  0.0214549   5.760 9.71e-09 ***
## `Fill Pressure`     0.0474994  0.0249315   1.905 0.056896 .
## `Hyd Pressure4`    -0.0166504  0.0256101  -0.650 0.515669
```

```
## Temperature         -0.1116626  0.0203794  -5.479 4.80e-08 ***
## `Usage cont`        -0.1643754  0.0221270  -7.429 1.61e-13 ***
## `Carb Flow`          0.1255049  0.0235212   5.336 1.06e-07 ***
## Density             -0.0571036  0.0473462  -1.206 0.227925
## MFR                 -0.0638115  0.0224211  -2.846 0.004471 **
## `Pressure Vacuum`    0.0031887  0.0215295   0.148 0.882273
## `Oxygen Filler`      0.0696359  0.0201399   3.458 0.000556 ***
## `Bowl Setpoint`      0.2942108  0.0239752  12.271  < 2e-16 ***
## `Pressure Setpoint` -0.1332461  0.0260786  -5.109 3.53e-07 ***
## `Air Pressurer`      0.0046631  0.0189398   0.246 0.805549
## `Carb Rel`           0.1055170  0.0368774   2.861 0.004262 **
## .data_A             -0.4149322  0.0770714  -5.384 8.14e-08 ***
## .data_B             -0.0018921  0.1010655  -0.019 0.985065
## .data_C             -0.8348426  0.1104548  -7.558 6.14e-14 ***
## .data_D                     NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8099 on 2032 degrees of freedom
## Multiple R-squared:  0.341,  Adjusted R-squared:  0.3329
## F-statistic: 42.06 on 25 and 2032 DF,  p-value: < 2.2e-16
```

Fig 30: R Code for MLR

Interpretation

In Fig 30, the lm() function fits all predictor variables in the training set to the associated pH, and identifies regression coefficients and intercept for the linear model. Residuals statistics are displayed and indicate the residuals are normally distributed around 0. This is an important check as normally distributed residuals with mean of 0 are required for valid linear regression modeling. Residual deviations from an expected value of zero mean the predictors have not accounted for important information necessary in predicting the target.

Stepwise Regression was used to further reduce the number of predictors in this model - variables were subtracted in a backward **stepwise** fashion until the highest adjusted R^2 (due to multiple regressors) value was achieved, indicating the best (and lowest) combination of variables for predicting the target.

Code

```r
# Stepwise Backwards
step.model <- stepAIC(lm_mod, direction = "backward", trace=FALSE)
summary(step.model)
```

Output

```
##
## Call:
## lm(formula = PH ~ `Carb Volume` + `Fill Ounces` + `PC Volume` +
##     PSC + `PSC CO2` + `Carb Pressure1` + `Fill Pressure` + Temperature +
##     `Usage cont` + `Carb Flow` + MFR + `Oxygen Filler` + `Bowl Setpoint` +
##     `Pressure Setpoint` + `Carb Rel` + .data_A + .data_C, data = training,
##     seed = 29)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9064 -0.5170  0.1029  0.5512  2.9036
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          0.16378    0.02093   7.825 8.08e-15 ***
## `Carb Volume`       -0.08151    0.03008  -2.709 0.006797 **
## `Fill Ounces`       -0.04585    0.01829  -2.507 0.012257 *
## `PC Volume`         -0.06921    0.02009  -3.444 0.000584 ***
## PSC                 -0.03200    0.01845  -1.735 0.082938 .
## `PSC CO2`           -0.03931    0.01768  -2.223 0.026323 *
## `Carb Pressure1`     0.12346    0.02131   5.793 7.98e-09 ***
## `Fill Pressure`      0.04779    0.02476   1.930 0.053761 .
## Temperature         -0.10953    0.01994  -5.493 4.45e-08 ***
## `Usage cont`        -0.16544    0.02169  -7.629 3.61e-14 ***
## `Carb Flow`          0.12433    0.02267   5.485 4.66e-08 ***
## MFR                 -0.06051    0.02039  -2.968 0.003028 **
## `Oxygen Filler`      0.07085    0.01992   3.556 0.000385 ***
## `Bowl Setpoint`      0.29790    0.02291  13.003  < 2e-16 ***
## `Pressure Setpoint` -0.12951    0.02561  -5.057 4.65e-07 ***
## `Carb Rel`           0.08187    0.03052   2.683 0.007364 **
## .data_A             -0.45989    0.05433  -8.465  < 2e-16 ***
## .data_C             -0.84201    0.05816 -14.477  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.809 on 2040 degrees of freedom
## Multiple R-squared:  0.3399, Adjusted R-squared:  0.3344
## F-statistic: 61.78 on 17 and 2040 DF,  p-value: < 2.2e-16
```

Fig 31: R Code for Stepwise MLR

Interpretation

Fig 31 shows that once we pass the model into the **stepAIC()** function it subtracts variables until the best adjusted R-squared value is achieved. We see from the output that there are now 17 variables in this stepwise model, as opposed to 25 as seen in Fig 30. Residuals appear normally distributed around 0, making the regression assumption valid. 33.9% of the variability is captured by the optimized model mas indicated by an R-squared of 0.3399. Interesting enough this represents a slight improvement over the full model, which had an adjusted R squared of .3329 or 33.3%. As a consequence, the gain in explained variance may only be due to dropped predictors improving the adjusted R^2, and not due to an increase in model accuracy.

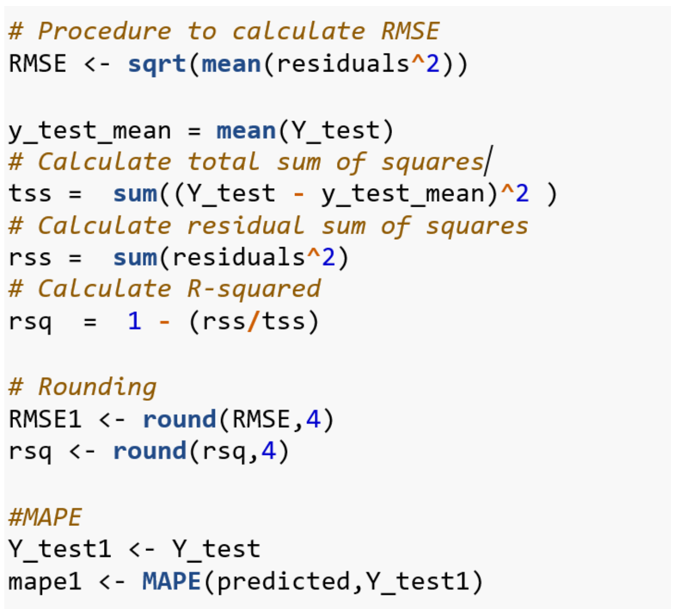
To measure the accuracy of our regression models, **pH** was predicted using the test set, and **MAPE** and **RMSE** was calculated as a benchmark to compare against other models.

Code

```r
# Procedure to calculate predicted values.
predicted <- predict(step.model, X_test)
residuals <- Y_test - predicted
```

Output

```r
paste0("OLS Regression R-Squard Value: ", rsq)

## [1] "OLS Regression R-Squard Value: 0.3023"

paste0("OLS Regression RMSE: ", RMSE1)

## [1] "OLS Regression RMSE: 0.8626"

paste0("OLS Regression MAPE: ", mape1)

## [1] "OLS Regression MAPE: 1.95749707147134"
```



```r
# Procedure to calculate RMSE
RMSE <- sqrt(mean(residuals^2))

y_test_mean = mean(Y_test)
# Calculate total sum of squares
tss =  sum((Y_test - y_test_mean)^2 )
# Calculate residual sum of squares
rss =  sum(residuals^2)
# Calculate R-squared
rsq  =  1 - (rss/tss)

# Rounding
RMSE1 <- round(RMSE,4)
rsq <- round(rsq,4)

#MAPE
Y_test1 <- Y_test
mape1 <- MAPE(predicted,Y_test1)
```

Fig 32: R Code for MAPE calculates for MLR model

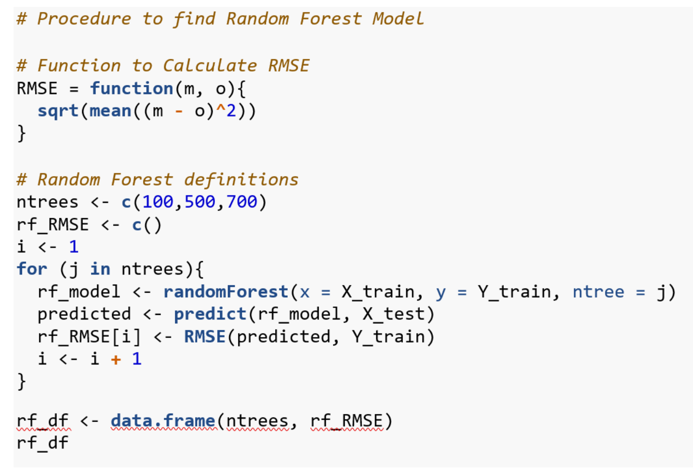
Interpretation

The predict function uses the stepwise linear regression model to calculate the **target variable** (pH) in the **testing** set. The errors, or residuals, are calculated as the difference between the predicted pH and the test set pH, and converted to a **RMSE** value. **MAPE** is similarly calculated between the predicted pH and test set using the **MAPE()** function from the **MLmetrics** library. The R squared of our model was 0.3023, indicating a somewhat positive linear relationship and that 30.23% of the variability for pH was explained in the model. The RMSE and MAPE were 0.82626 and 1.957, and serve as benchmarks to compare other models.

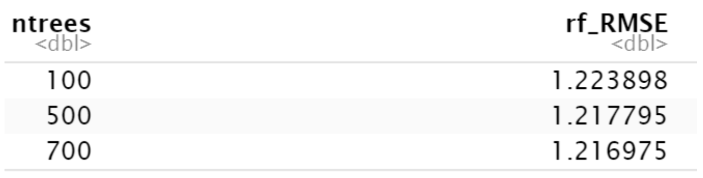
Random Forest

Random Forest model uses an ensemble of decision trees to create estimates for a target variable. These estimates are combined to achieve a final estimate with the lowest error. Three iterations were used for training with the number of trees (n) = 100, 500 and 700. **RMSE** and **MAPE** were calculated to compare accuracy with other models.

Code	**Output**



```
# Procedure to find Random Forest Model

# Function to Calculate RMSE
RMSE = function(m, o){
  sqrt(mean((m - o)^2))
}

# Random Forest definitions
ntrees <- c(100,500,700)
rf_RMSE <- c()
i <- 1
for (j in ntrees){
  rf_model <- randomForest(x = X_train, y = Y_train, ntree = j)
  predicted <- predict(rf_model, X_test)
  rf_RMSE[i] <- RMSE(predicted, Y_train)
  i <- i + 1
}

rf_df <- data.frame(ntrees, rf_RMSE)
rf_df
```



ntrees <dbl>	rf_RMSE <dbl>
100	1.223898
500	1.217795
700	1.216975

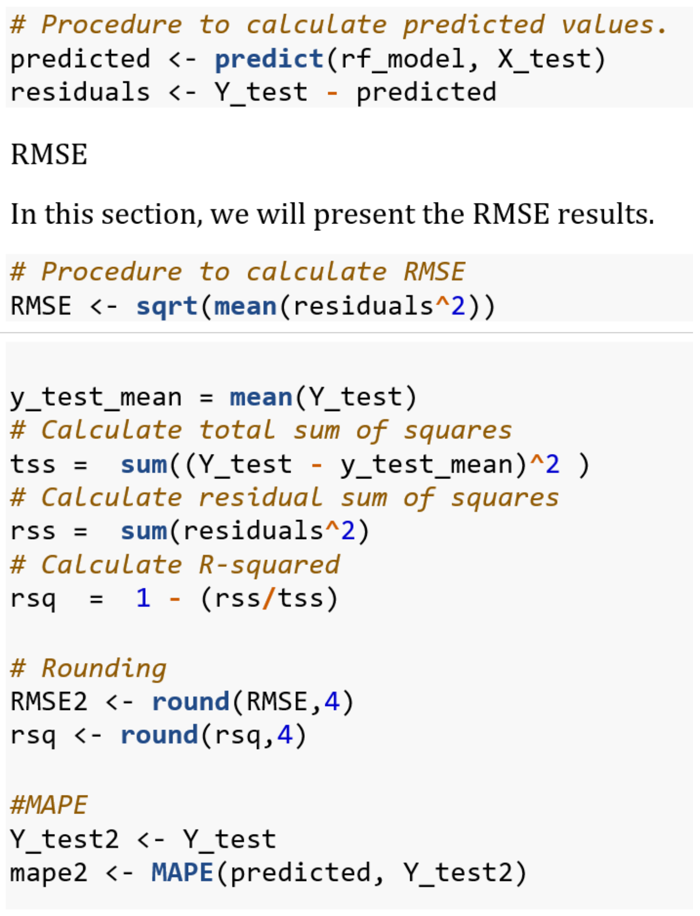
Fig 33: R Code for Random Forest Model

Interpretation

The n = 700 random forest model resulted in the lowest RMSE score (1.2169) and thus was the best performing--though only slightly better than n = 500 (1.2147) and n=100(1.2143). However, as the number of trees increases, the longer it takes for the model to be trained.

Following the previous methodology, the n = 700 random forest model was used to calculate the pH, and MAPE.

Code



```
# Procedure to calculate predicted values.
predicted <- predict(rf_model, X_test)
residuals <- Y_test - predicted
```

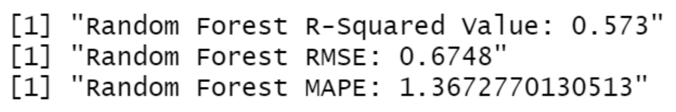
RMSE

In this section, we will present the RMSE results.

```
# Procedure to calculate RMSE
RMSE <- sqrt(mean(residuals^2))
```

```
y_test_mean = mean(Y_test)
# Calculate total sum of squares
tss =  sum((Y_test - y_test_mean)^2 )
# Calculate residual sum of squares
rss =  sum(residuals^2)
# Calculate R-squared
rsq  =  1 - (rss/tss)

# Rounding
RMSE2 <- round(RMSE,4)
rsq <- round(rsq,4)

#MAPE
Y_test2 <- Y_test
mape2 <- MAPE(predicted, Y_test2)
```

Output



```
[1] "Random Forest R-Squared Value: 0.573"
[1] "Random Forest RMSE: 0.6748"
[1] "Random Forest MAPE: 1.3672770130513"
```

Fig 34: R Code for calculating RMSE and MAPE for Random Forest (n = 700)

Interpretation

The n = 700 model is applied to the test data and residuals are calculated , which are then converted to a R squared, **RMSE** and **MAPE** score, which are 0.576, 0.676 and 1.326 respectively. Comparing these metrics to the multiple linear regression model concludes that the random forest model accounted for more variability (0.576 vs 0.3023) and had lower overall error (expressed in lower RMSE and MAPE). The Random Forest model outperformed the linear regression model but at the expense of longer computing time.

Gradient Boosting Tree Models

The final model we explored was a **Gradient Boosted Tree Model** - an advanced ensemble decision tree model, implemented through the XgBoost library**.** XgBoost is well known for its accuracy and takes significantly more training time than the other models. Unlike random forest ensemble, boosting is a sequential process in which tree models are built in series, scaled and used to inform the next tree model. Each model is combined in an ensemble to improve on the performance of the previous collection of models. A gradient boosting tree model was selected as the final model to train in this project because it provides methods to address the balance between **overfitting** and **underfitting** (which may have been an issue in the previous models trained).

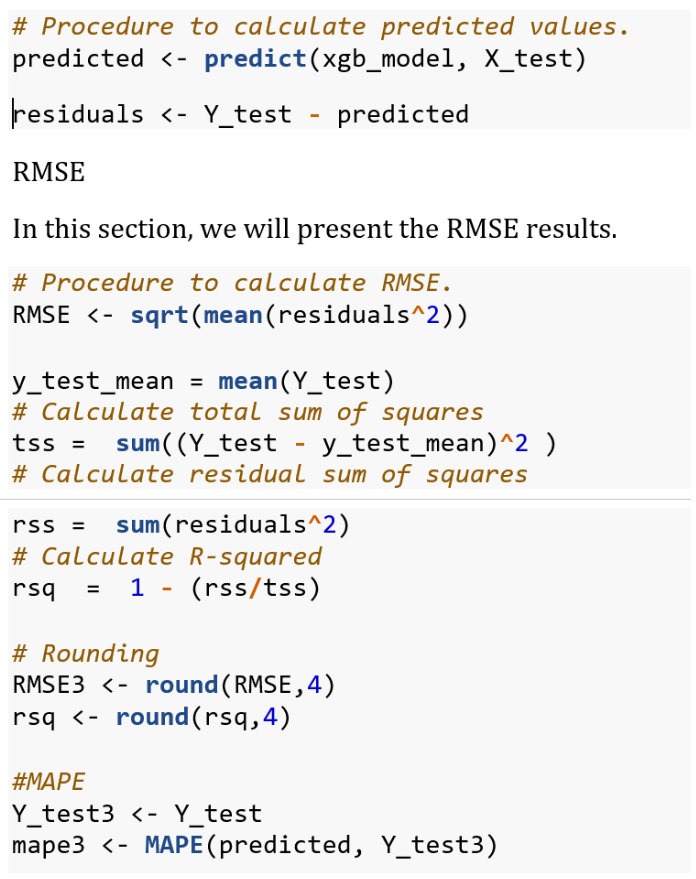
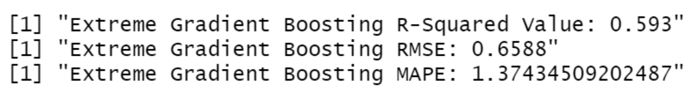
Code

```r
# XGboost works with using the xgb.DMatrix function
X_train = xgb.DMatrix(as.matrix(X_train))
X_test = xgb.DMatrix(as.matrix(X_test))

# Creating a cross validation control
xgb_trcontrol = trainControl(
  method = "cv",
  number = 5,
  allowParallel = TRUE,
  verboseIter = FALSE,
  returnData = FALSE
)

# Setting up a grid search for the best parameters
xgbGrid <- expand.grid(nrounds = c(100,200),  # this is n_estimators above
                       max_depth = c(10, 15, 20, 25),
                       colsample_bytree = seq(0.5, 0.9, length.out = 5),
                       ## The values below are default values in the sklearn-
api.
                       eta = 0.1,
                       gamma=0,
                       min_child_weight = 1,
                       subsample = 1
                      )
set.seed(123)
xgb_model = train(
  X_train, Y_train,
  trControl = xgb_trcontrol,
  tuneGrid = xgbGrid,
  method = "xgbTree"
)
```

Fig 35: R Code for calculating training Xgboost model

Interpretation

First training and test data sets were converted into a format that the **xgbTree** model can use. To fully leverage the advantages of gradient boosting trees, a 5-fold cross validation and a grid with a range of tuning parameters were specified; these will be used to select optimal tuning parameters that decrease error and find a balance between **underfitting** and **overfitting** in one step during training. Similar to previous models, **MAPE** and **RMSE** were subsequently calculated as seen in Fig 36 below.

Code	**Output**
	

```
# Procedure to calculate predicted values.
predicted <- predict(xgb_model, X_test)

residuals <- Y_test - predicted
```

RMSE

In this section, we will present the RMSE results.

```
# Procedure to calculate RMSE.
RMSE <- sqrt(mean(residuals^2))

y_test_mean = mean(Y_test)
# Calculate total sum of squares
tss =  sum((Y_test - y_test_mean)^2 )
# Calculate residual sum of squares
```

```
rss =  sum(residuals^2)
# Calculate R-squared
rsq  =  1 - (rss/tss)

# Rounding
RMSE3 <- round(RMSE,4)
rsq <- round(rsq,4)

#MAPE
Y_test3 <- Y_test
mape3 <- MAPE(predicted, Y_test3)
```

```
[1] "Extreme Gradient Boosting R-Squared Value: 0.593"
[1] "Extreme Gradient Boosting RMSE: 0.6588"
[1] "Extreme Gradient Boosting MAPE: 1.37434509202487"
```

Fig 36: R Code for calculating RMSE and MAPE for Gradient Boosting Model

Interpretation

RMSE and **MAPE** metrics were calculated and yielded , which are 0.6588 and 1.3743 respectively. Comparing **MAPE** metrics to the multiple linear regression and random forest models concludes that the gradient boosting model was very close to random forest, but posed too unstable for a reliable prediction. Each time we ran the model it yielded results that fluctuated between MAPE ~ 1.2 and MAPE ~ 1.4, therefore we did not choose it as the best model. It is interesting to note that the **RMSE** is less in the gradient boosting model than in the random forest model. This may be due to random forest being prone to overfitting whereas Xgboost is typically resistant to overfitting. This relationship also fluctuate due to the instability of the gradient boosting model.

Variable Importance

One advantage to gradient boosting models is that they can provide estimates of feature importance.This is of particular interest for us because it may inform recommendations that we give to the manufacturing company based on which predictors inform pH the most. To do this we applied the **varImp()** function from the **caret** library as seen in Fig 37.

Code

```r
# Procedure to extract the Variable Importance.
xgb_varimp <- varImp(xgb_model)
```

Output

```
## xgbTree variable importance
##
##   only 20 most important variables shown (out of 26)
##
##    Overall
## 12 100.000
## 11  40.275
## 24  35.466
## 17  32.299
## 14  30.157
## 18  26.060
## 16  25.947
## 13  20.140
```

```
## 8   20.089
## 21  19.624
## 19  17.173
## 15  14.337
## 2   12.419
## 0   12.061
## 20  10.926
## 9   10.235
## 10   8.243
## 5    6.965
## 3    6.777
## 22   6.631
```

Fig 37: R Code for calculating RMSE and MAPE for Gradient Boosting Model

Interpretation

Fig 37 shows the scores associated with each variable; we can see that "variable 12" is the most important variable in this model, followed by variables 11, 24, and 17. We can see however that although there are variables that contribute more to pH than others, they all contribute to predicting pH more than 5% and therefore, we assume that they are all significant predictors for pH. This validates our preprocessing steps and assumptions to keep these variables for training the model.

Results

Summary of Errors

Fig 38 displays error metrics, **MAPE** and **RMSE,** for the three models listed above: Multiple Linear Regression (labeled as **OLS)**, Random Forest and Gradient Boosting Tree Model.

Code

```r
mlacc <- data.frame(
model1 = c("OLS", "Random Forest", "XGBoost"),
mapec = c(mape1,mape2, mape3),
rmsec = c(RMSE1,RMSE2,RMSE3)
)
colnames(mlacc)<-c("MODEL","MAPE","RMSE")
mlacc
```

Output

MODEL <fctr>	MAPE <dbl>	RMSE <dbl>
OLS	1.957497	0.8626
Random Forest	1.367277	0.6748
XGBoost	1.374345	0.6588

Fig 38: R Code for displaying RMSE and MAPE for all models

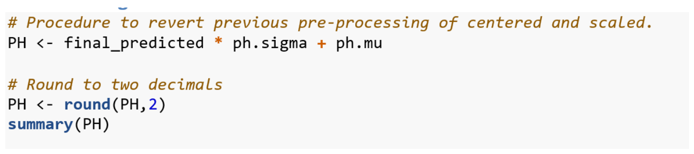
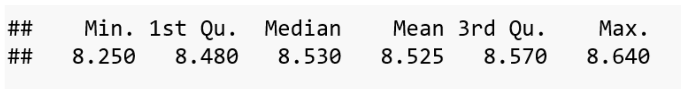
Interpretation

The Random Forest model achieved the lowest **MAPE** score (1.3673) and a relatively low **RMSE** score (0.6748). The gradient bootsting model performed well with respect to **RMSE** (0.65), however the **MAPE** score was slightly higher than random forest (1.3743) We speculate that the differences in **RMSE** are due to random forest models being more prone to overfitting. A lower **RMSE** indicates that less variability in pH is explained, and may also indicate that the model is less specific to the characteristics of training data, and therefore may be slighting overfitting. Given that the **MAPE** for Xgboost is more than Random Forest, we infer that Random Forest provides a more robust model that is better at generalizing features for all data. In addition it yielded results that were more consistent than Xgboost (as in, the error metric did not fluctuate as much as it did for Xgboost when running models multiple times). This is a classic illustration of the bias-variance trade off problem with machine learning models, to which gradient boosting models specifically addresses.

Considering how close the **RMSE** scores were between Random Forest and XgBoost, the deciding factor was the **MAPE** values which made for a clear decision: Random Forest is the best model at predicting pH. Multiple linear regression (labeled as OLS in Fig 38) had significantly higher error metrics and served as a baseline for assessing performance for random forest and the gradient boosting model.

Visualizing Results

Before final prediction can be made, pre-processed data must be recentered and rescaled back to the original form so it can make predictions on the appropriate scale. Values of pH must be between zero and 14 and maintain a similar distribution to training data target variables.

Code	Output
	

```
# Procedure to revert previous pre-processing of centered and scaled.
PH <- final_predicted * ph.sigma + ph.mu

# Round to two decimals
PH <- round(PH,2)
summary(PH)
```

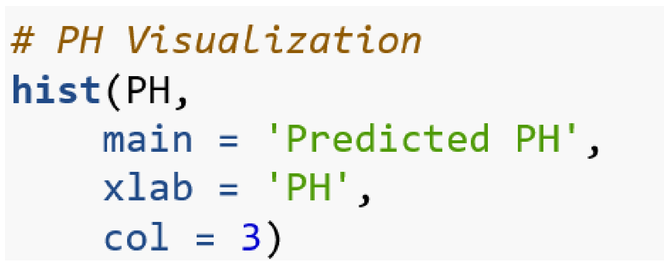
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   8.250   8.480   8.530   8.525   8.570   8.640
```

Fig 39: R Code for reversing pre processing

Interpretation

Fig 39 shows that after reversing preprocessing, the five number summary for predicted pH falls within zero and 14 as expected. It seems reasonable considering the context of the scenario. Fig 40 confirms the distribution of predicted pH values are consistent with training data.

Code	**Output**



```
# PH Visualization
hist(PH,
    main = 'Predicted PH',
    xlab = 'PH',
    col = 3)
```

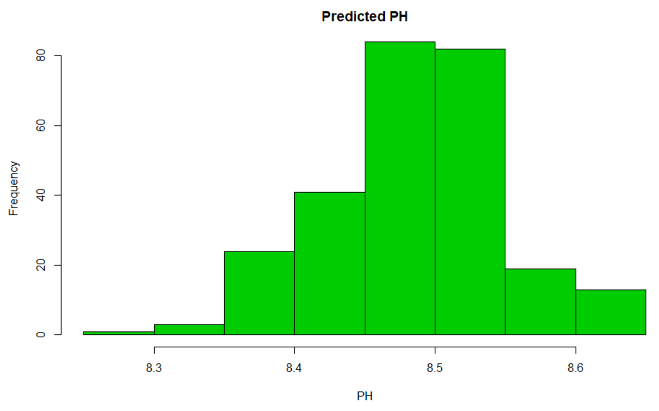
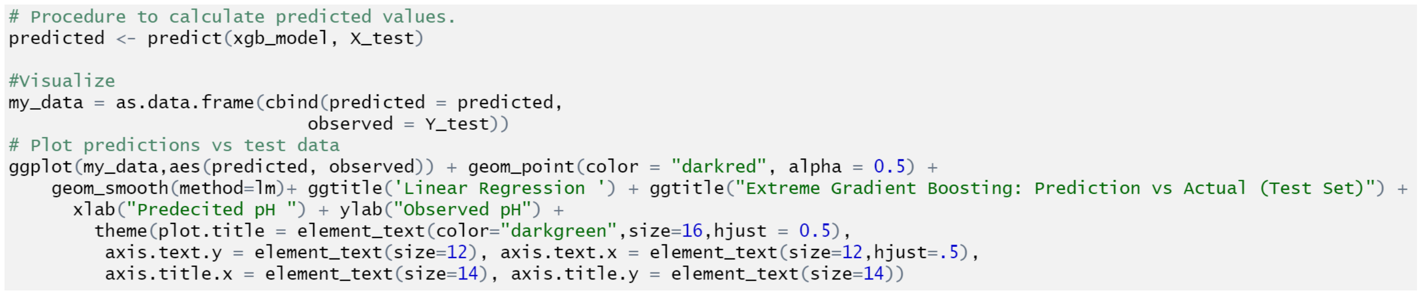


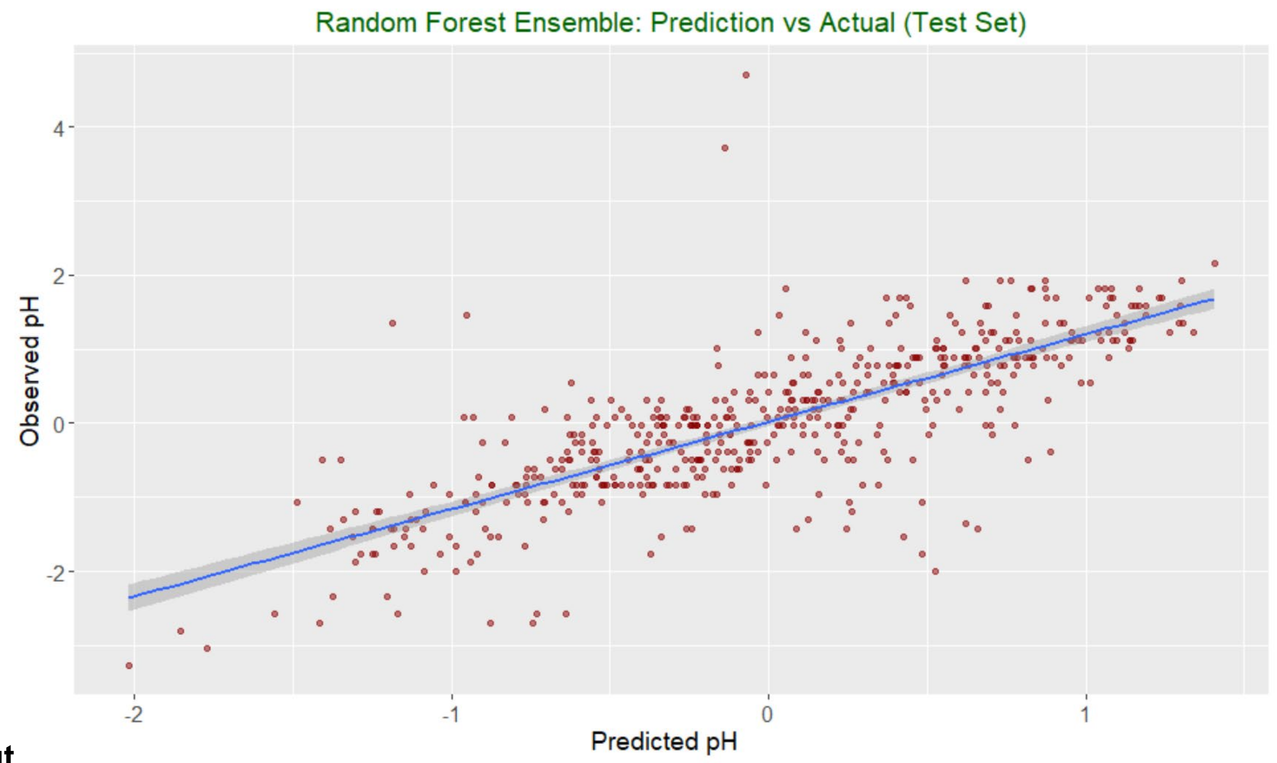
Fig 40: R Code for generating histogram of predicted pH

To get a better sense of the relationship between predicted and actual values determined by this model, they were plotted below.

Code



```
# Procedure to calculate predicted values.
predicted <- predict(xgb_model, X_test)

#Visualize
my_data = as.data.frame(cbind(predicted = predicted,
                            observed = Y_test))
# Plot predictions vs test data
ggplot(my_data,aes(predicted, observed)) + geom_point(color = "darkred", alpha = 0.5) +
    geom_smooth(method=lm)+ ggtitle('Linear Regression ') + ggtitle("Extreme Gradient Boosting: Prediction vs Actual (Test Set)") +
      xlab("Predecited pH ") + ylab("Observed pH") +
        theme(plot.title = element_text(color="darkgreen",size=16,hjust = 0.5),
         axis.text.y = element_text(size=12), axis.text.x = element_text(size=12,hjust=.5),
         axis.title.x = element_text(size=14), axis.title.y = element_text(size=14))
```



Output

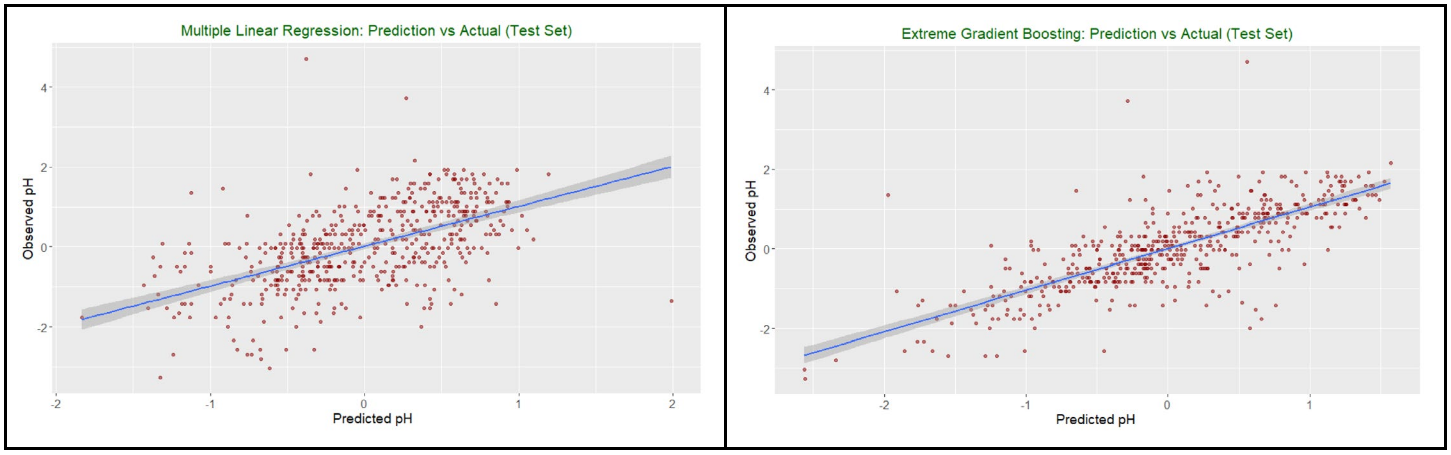


Fig 41: Predicted vs. Observed pH for all 3 models

Interpretation

Fig 41 illustrates the predicted pH via the gradient boosting model vs the actual pH values in the test set (visualization for MLR and Xgboost are also included for reference). We can see that our model follows the same trends as the actual data follows, and visually confirms that the gradient boosting model is an appropriate predictor of pH.

Conclusions

The research involved the development of three models for predicting pH based on production data. The performance metrics demonstrated lower error while using the Random Forest algorithm over Xgboost, and standard Multiple Linear Regression. However, it is worth mentioning the Xgboost model was a close second. Potentially, if used in industry both models could be run alongside one another to see how performance continues to vary. For this analysis only one set of predictions will be submitted therefore using the information collected in this report, it was determined that Random Forest will be the optimal model for making predictions.

Exploratory Insights

Importance of Original Variables

While the original data set contained 32 predictor variables, only 26 variables were retained in training models due to multicollinearity, redundancy of information, and low-explanation of variability. In an ideal world with zero acquisition, storage and/or opportunity costs of measuring production data, such extraneous and multiple model training is practical. However, from a business and engineering perspective, dimension reduction and model identification is a critical step in the predictive process as choices made can reduce cost associated with acquiring data, and training models (computational costs). In general, simple models with large training sets outperform complex models with small training sets due to their ability to generalize features for all data. Low dimensional models also limit the extent new data needs to be preprocessed to provide new insight.

Random Forest vs XgBoost

The tradeoff between the time and energy needed to train these models and the accuracy gained is scale dependent. The percent difference in the accuracy of these models will only become noticeable once the observations predicted reach the hundreds of thousands. Whether the tradeoff is worth it depends if the models will be used at that scale. Furthermore, as more training data becomes available, the changes in the performance of these two models may justify a single model at all scales of production.

Optimizing Manufacturing Processes

The point of the data science process is to provide insights, options, advice, and support for a diverse set of environments; a decision making tool backed up by data. In this case the objective was to predict pH levels in the manufacturing process for beverages. Sometimes this lack of context is an advantage in due part to its simplicity, however, in general domain knowledge tends to be a determinant factor when providing value towards the business. With the latter approach in mind the following insights were determined:

- The manufacturing processes among the different brands could be optimized since some of the elements don't seem to be relevant to the brand creation.
- Quality control in the manufacturing line is another improvement area; some of these variables presented some "senseless" readings that could have come from sloppy management of QA.
- Variable importance between brands B and C coincide well with one another (see Fig 42 below), so much so that consolidation of manufacturing lines/processes would be strongly suggested in order to further accumulate savings and drive efficiency.

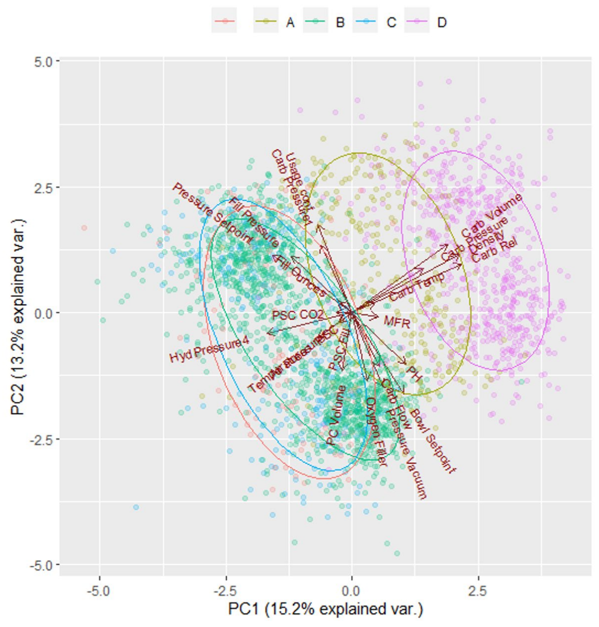


Fig 42: Visualization of PC1 and PC2 with original predictor contribution

References

1. "Science News for Students" https://www.sciencenewsforstudents.org/article/scientists-say-ph
2. Reddy, Avanija et al. "The pH of beverages in the United States." Journal of the American Dental Association (1939) vol. 147,4 (2016): 255-63. doi:10.1016/j.adaj.2015.10.019
3. Kuhn, Maxwell and Johnson Kjell. Applied Predictive Modeling. Springer 2013
4. Ortega, et all. "Summer DATA 624: Predictive Analysis Project 1".
5. Mean Absolute Percent Error. Wikipedia https://en.wikipedia.org/wiki/Mean_absolute_percentage_error.
6. Multicollinearity. Wikipedia. https://en.wikipedia.org/wiki/Multicollinearity
7. "Dimensionality Reduction." 2013, 33-52.
8. Marutho, D., Hendra Handaka, Wijaya, and Muljono. "The Determination of Cluster Number at K-Mean Using Elbow Method and Purity Evaluation on Headline News." 2018 International Seminar on Application for Technology of Information and Communication, 2018, 533-38.
9. Wang, Fu-Kwun, and Tadele Mamo. "Gradient Boosted Regression Model for the Degradation Analysis of Prismatic Cells." Computers & Industrial Engineering 144 (2020): Computers & Industrial Engineering, June 2020, Vol.144.
10. Li, Jia, Yujuan Si, Tao Xu, and Saibiao Jiang. "Deep Convolutional Neural Network Based ECG Classification System Using Information Fusion and One-Hot Encoding Techniques." Mathematical Problems in Engineering 2018 (2018): 10.
11. Osamor, Victor Chukwudi, Ezekiel Femi Adebiyi, Jelilli Olarenwaju Oyelade, Seydou Doumbia, and Jérémie Bourdon. "Reducing the Time Requirement of K-Means Algorithm (Reducing the Time Requirement of K-Means Algorithm)." 7, no. 12 (2012): E49946.
12. Arefin, Ahmed Shamsul, Carlos Riveros, Regina Berretta, Pablo Moscato, and Alexandre G. De Brevern. "GPU-FS- K NN: A Software Tool for Fast and Scalable K NN Computation Using GPUs (GPU-FS- K NN: A Fast and Scalable K NN Using GPUs)." 7, no. 8 (2012): E44000.
13. Coleman, C., and D. Swanson. "On MAPE-R as a Measure of Cross-sectional Estimation and Forecast Accuracy." Journal of Economic and Social Measurement 32, no. 4 (2007): 219-34.
14. https://books.google.com/books?id=mHmxNGKRlQsC&lpg=PA245&pg=PA245#v=onepage&q&f=false
15. Wu, Rundi, Chen, Xuelin, Zhuang, Yixin, and Chen, Baoquan. "Multimodal Shape Completion via Conditional Generative Adversarial Networks." 2020.
16. Black, John, Nigar Hashimzade, and Gareth Myles. "Ordinary Least Squares." A Dictionary of Economics, 2012, A Dictionary of Economics.
17. https://onesearch.cuny.edu/permalink/f/17bo7l3/TN_gvrl_refCX3708001173
18. Hess, Aaron S., and Hess, John R. "Principal Component Analysis." *Transfusion* 58, no. 7 (2018): 1580-582.
19. Wei, Suhua, Yi, Chuixiang, Fang, Wei, and Hendrey, George. "A Global Study of GPP Focusing on Light-Use Efficiency in a Random Forest Regression Model." 2017, Queens College, Publications and Research.
20. María Edith Seier Zúñiga. "KURTOSIS." Pesquimat 6, no. 2 (2014): Pesquimat, 01 September 2014, Vol.6(2).
21. Lai, and Ing, Tze, Ching-Kang Leung. "Stepwise Regression." 2010, 1449-451.

22. Sukhpreet Singh Dhaliwal, Abdullah-Al Nahid, and Robert Abbas. "Effective Intrusion Detection System Using XGBoost." Information (Basel) 9, no. 7 (2018): 149.
23. "PH." 2014, 3318-319.