

Data 624 Project 2

ABC Beverage Analysis



Team Members: Maryluz Cruz, Bill Stepniak,

Sherranette Tinapunan

Data 624 Project 2

Maryluz Cruz, Bill Stepniak, Sherranette Tinapunan

5/11/2021

## 1. Problem Statement

This is role playing. I am your new boss. I am in charge of production at ABC Beverage and you are a team of data scientists reporting to me. My leadership has told me that new regulations are requiring us to understand our manufacturing process, the predictive factors and be able to report to them our predictive model of PH. Please use the historical data set I am providing. Build and report the factors in BOTH a technical and non-technical report. I like to use Word and Excel. Please provide your non-technical report in a business friendly readable document and your predictions in an Excel readable format. The technical report should show clearly the models you tested and how you selected your final approach. Please submit both RPubs links and .rmd files or other readable formats for technical and non-technical reports. Also submit the excel file showing the prediction of your models for pH.

## 2. Data Exploration

### Load Data

Two files are provided for this project. The file StudentData.xlsx contains the data used to build the predictive models for PH. The file StudentEvaluation.xlsx contains the data used to evaluate the selected model. The PH column in this file has no data. These files were converted to .csv and uploaded to the GitHub for easy loadability.

### Summary of Student Data

The Student Data data set has 2,571 observations and 33 variables.

This data set contains categorical, continuous, and discrete variables. The data set has some missing values for most of the variables. The response variable PH has four missing values. All the predictor variables have missing values except forPressure.Vacuum and Air.Pressure. The categorical variable Brand.Code has blank values.

Data summary

|  |  |
| --- | --- |
| Name | student\_data |
| Number of rows | 2571 |
| Number of columns | 33 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| character | 1 |
| numeric | 32 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: character**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | min | max | empty | n\_unique | whitespace |
| ï..Brand.Code | 0 | 1 | 0 | 1 | 120 | 5 | 0 |

**Variable type: numeric**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| Carb.Volume | 10 | 1.00 | 5.37 | 0.11 | 5.04 | 5.29 | 5.35 | 5.45 | 5.70 | ▁▆▇▅▁ |
| Fill.Ounces | 38 | 0.99 | 23.97 | 0.09 | 23.63 | 23.92 | 23.97 | 24.03 | 24.32 | ▁▂▇▂▁ |
| PC.Volume | 39 | 0.98 | 0.28 | 0.06 | 0.08 | 0.24 | 0.27 | 0.31 | 0.48 | ▁▃▇▂▁ |
| Carb.Pressure | 27 | 0.99 | 68.19 | 3.54 | 57.00 | 65.60 | 68.20 | 70.60 | 79.40 | ▁▅▇▃▁ |
| Carb.Temp | 26 | 0.99 | 141.09 | 4.04 | 128.60 | 138.40 | 140.80 | 143.80 | 154.00 | ▁▅▇▃▁ |
| PSC | 33 | 0.99 | 0.08 | 0.05 | 0.00 | 0.05 | 0.08 | 0.11 | 0.27 | ▆▇▃▁▁ |
| PSC.Fill | 23 | 0.99 | 0.20 | 0.12 | 0.00 | 0.10 | 0.18 | 0.26 | 0.62 | ▆▇▃▁▁ |
| PSC.CO2 | 39 | 0.98 | 0.06 | 0.04 | 0.00 | 0.02 | 0.04 | 0.08 | 0.24 | ▇▅▂▁▁ |
| Mnf.Flow | 2 | 1.00 | 24.57 | 119.48 | -100.20 | -100.00 | 65.20 | 140.80 | 229.40 | ▇▁▁▇▂ |
| Carb.Pressure1 | 32 | 0.99 | 122.59 | 4.74 | 105.60 | 119.00 | 123.20 | 125.40 | 140.20 | ▁▃▇▂▁ |
| Fill.Pressure | 22 | 0.99 | 47.92 | 3.18 | 34.60 | 46.00 | 46.40 | 50.00 | 60.40 | ▁▁▇▂▁ |
| Hyd.Pressure1 | 11 | 1.00 | 12.44 | 12.43 | -0.80 | 0.00 | 11.40 | 20.20 | 58.00 | ▇▅▂▁▁ |
| Hyd.Pressure2 | 15 | 0.99 | 20.96 | 16.39 | 0.00 | 0.00 | 28.60 | 34.60 | 59.40 | ▇▂▇▅▁ |
| Hyd.Pressure3 | 15 | 0.99 | 20.46 | 15.98 | -1.20 | 0.00 | 27.60 | 33.40 | 50.00 | ▇▁▃▇▁ |
| Hyd.Pressure4 | 30 | 0.99 | 96.29 | 13.12 | 52.00 | 86.00 | 96.00 | 102.00 | 142.00 | ▁▃▇▂▁ |
| Filler.Level | 20 | 0.99 | 109.25 | 15.70 | 55.80 | 98.30 | 118.40 | 120.00 | 161.20 | ▁▃▅▇▁ |
| Filler.Speed | 57 | 0.98 | 3687.20 | 770.82 | 998.00 | 3888.00 | 3982.00 | 3998.00 | 4030.00 | ▁▁▁▁▇ |
| Temperature | 14 | 0.99 | 65.97 | 1.38 | 63.60 | 65.20 | 65.60 | 66.40 | 76.20 | ▇▃▁▁▁ |
| Usage.cont | 5 | 1.00 | 20.99 | 2.98 | 12.08 | 18.36 | 21.79 | 23.75 | 25.90 | ▁▃▅▃▇ |
| Carb.Flow | 2 | 1.00 | 2468.35 | 1073.70 | 26.00 | 1144.00 | 3028.00 | 3186.00 | 5104.00 | ▂▅▆▇▁ |
| Density | 1 | 1.00 | 1.17 | 0.38 | 0.24 | 0.90 | 0.98 | 1.62 | 1.92 | ▁▅▇▂▆ |
| MFR | 212 | 0.92 | 704.05 | 73.90 | 31.40 | 706.30 | 724.00 | 731.00 | 868.60 | ▁▁▁▂▇ |
| Balling | 1 | 1.00 | 2.20 | 0.93 | -0.17 | 1.50 | 1.65 | 3.29 | 4.01 | ▁▇▇▁▇ |
| Pressure.Vacuum | 0 | 1.00 | -5.22 | 0.57 | -6.60 | -5.60 | -5.40 | -5.00 | -3.60 | ▂▇▆▂▁ |
| PH | 4 | 1.00 | 8.55 | 0.17 | 7.88 | 8.44 | 8.54 | 8.68 | 9.36 | ▁▅▇▂▁ |
| Oxygen.Filler | 12 | 1.00 | 0.05 | 0.05 | 0.00 | 0.02 | 0.03 | 0.06 | 0.40 | ▇▁▁▁▁ |
| Bowl.Setpoint | 2 | 1.00 | 109.33 | 15.30 | 70.00 | 100.00 | 120.00 | 120.00 | 140.00 | ▁▂▃▇▁ |
| Pressure.Setpoint | 12 | 1.00 | 47.62 | 2.04 | 44.00 | 46.00 | 46.00 | 50.00 | 52.00 | ▁▇▁▆▁ |
| Air.Pressurer | 0 | 1.00 | 142.83 | 1.21 | 140.80 | 142.20 | 142.60 | 143.00 | 148.20 | ▅▇▁▁▁ |
| Alch.Rel | 9 | 1.00 | 6.90 | 0.51 | 5.28 | 6.54 | 6.56 | 7.24 | 8.62 | ▁▇▂▃▁ |
| Carb.Rel | 10 | 1.00 | 5.44 | 0.13 | 4.96 | 5.34 | 5.40 | 5.54 | 6.06 | ▁▇▇▂▁ |
| Balling.Lvl | 1 | 1.00 | 2.05 | 0.87 | 0.00 | 1.38 | 1.48 | 3.14 | 3.66 | ▁▇▂▁▆ |

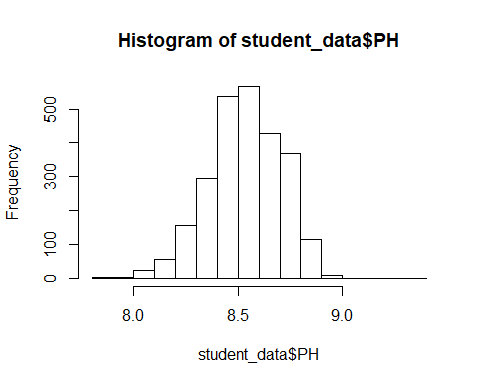
Predictor Brand.Code has four different categories, and there are 120 observations with categories that are not known.

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

Doing a little cleanup of the variable name i..Brand.Code and simply renaming this to Brand.Code.

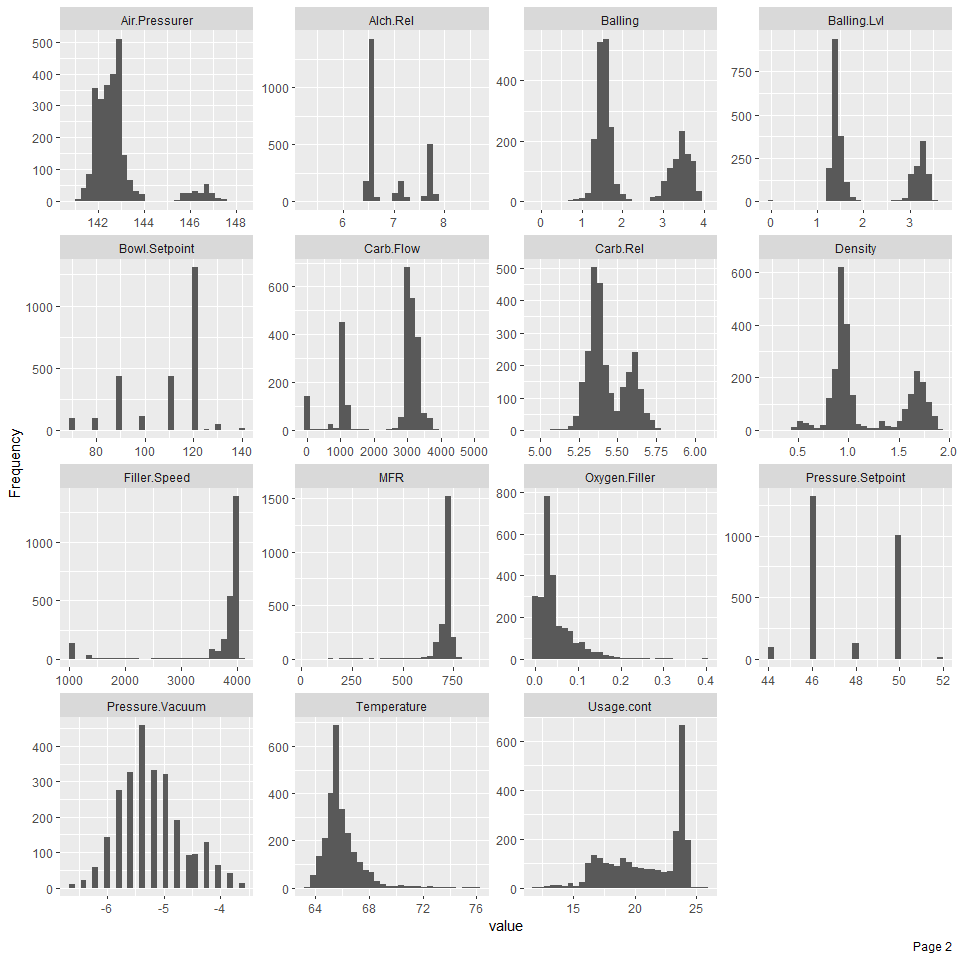
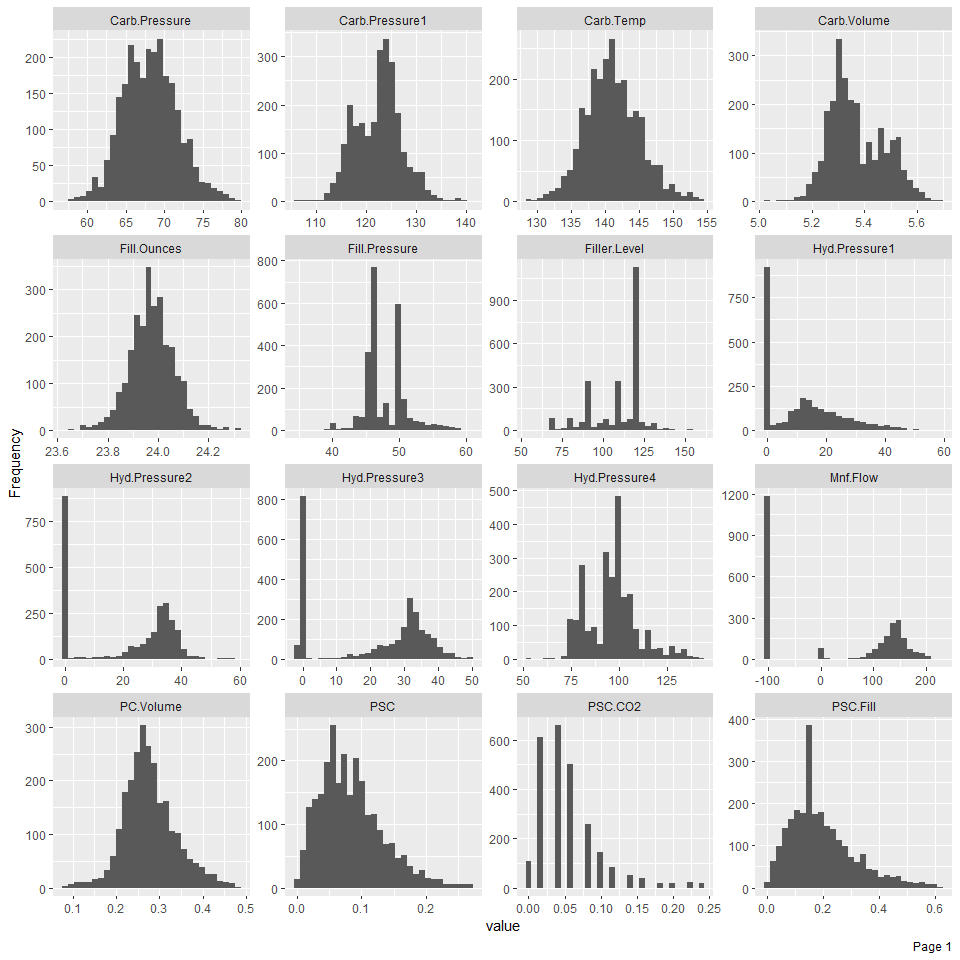
### Disribution of Response Variable PH

The distribution of the response variable PH is approximately normal with some degree of skewness to the left. The mean PH value is 8.5. Because we have fewer observations towards the end of the distribution, this could mean that the models are not able to predict PH values that fall towards the tails of the distribution.



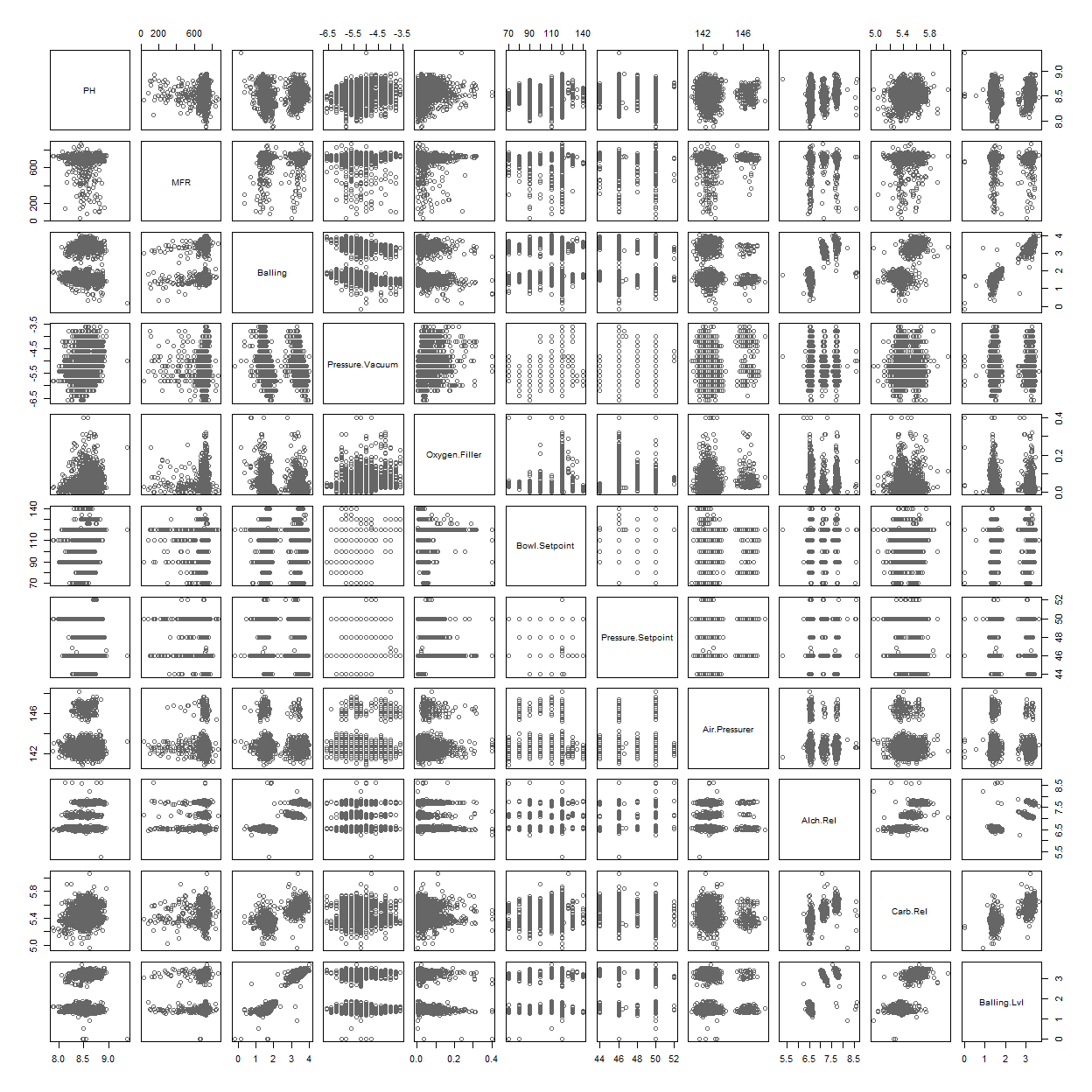
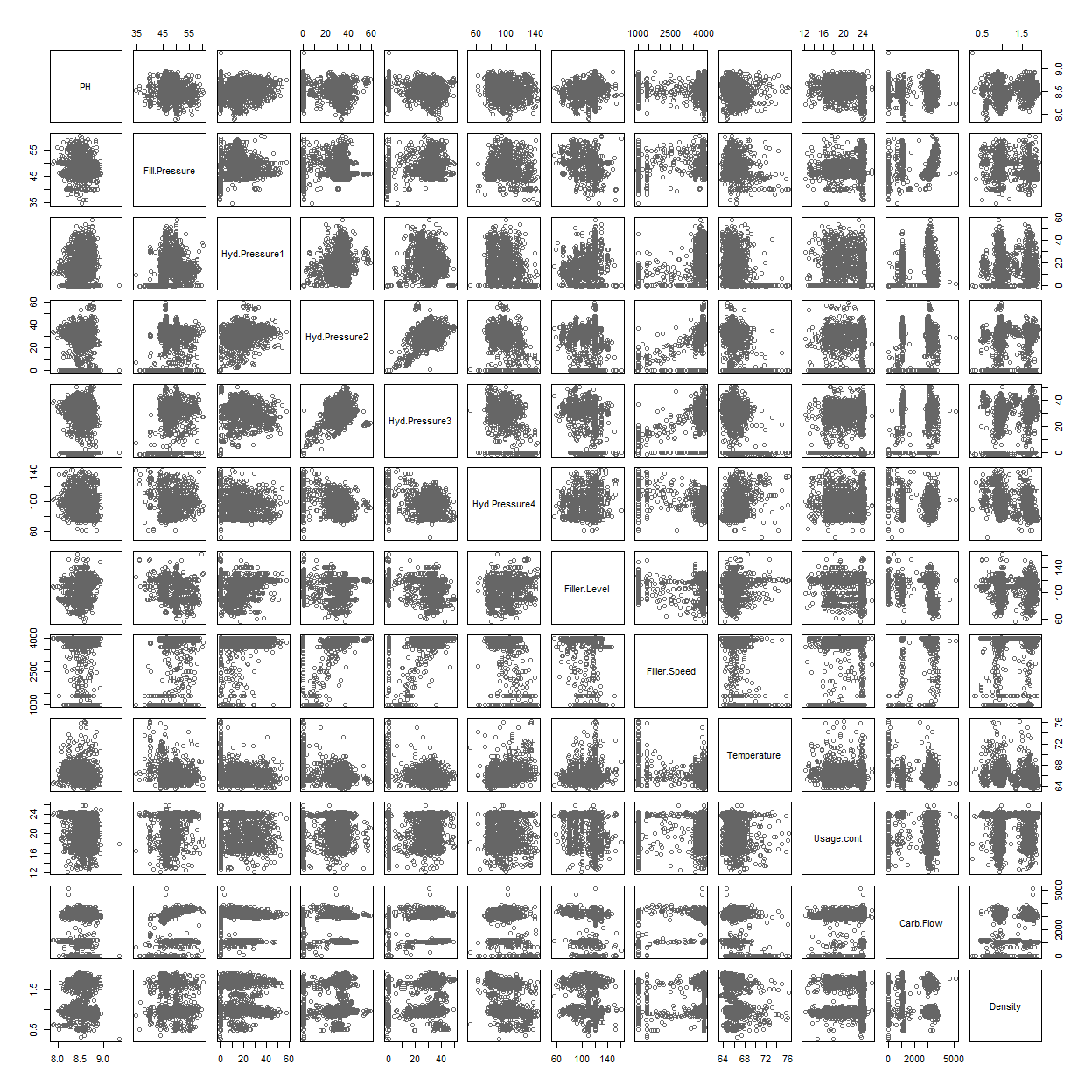
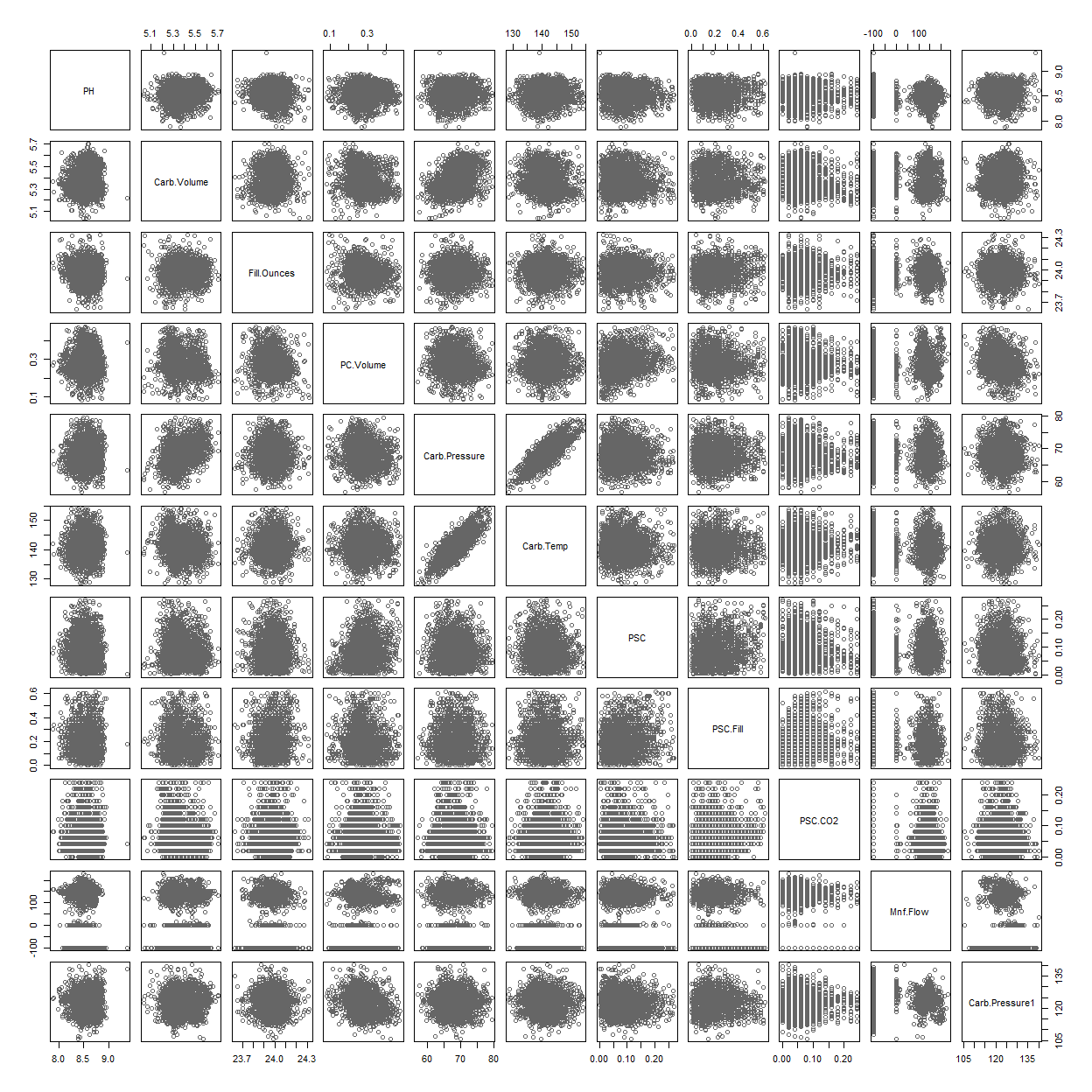
### Distribution of Predictors

Below shows the distribution of the numeric predictors. We can see that some variables have bimodal features (e.g., Air Pressure, Hyd.Pressure2, Hyd.Pressure3, Balling). Multimodal distributions could suggests different groups or different regions with another type of distribution shape. Some distributions appear to be more discrete with much more limited set of distinct values such as Pressure Setpoint, PSC.CO2, and Bowl.Setpoint.



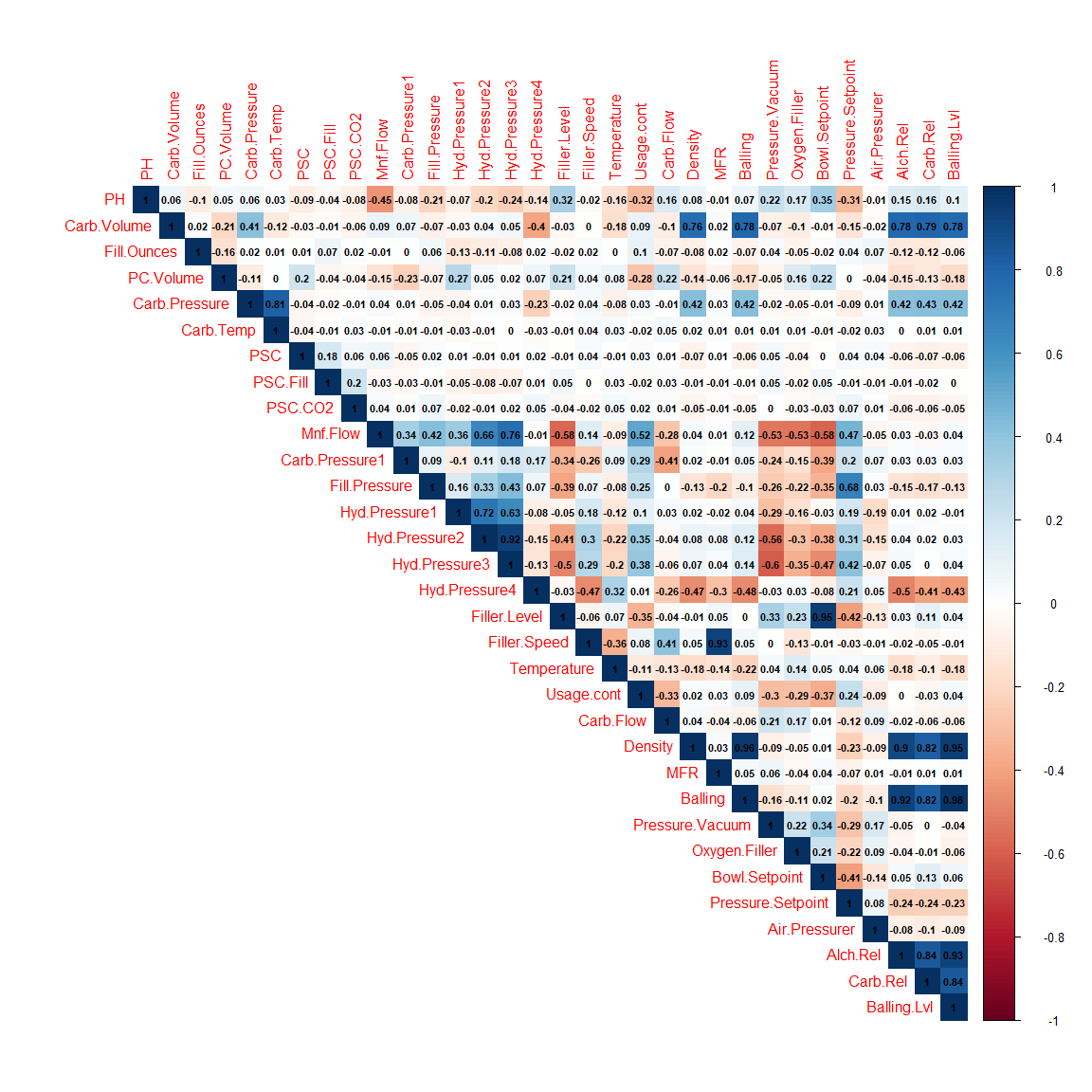
### Scatter Plots of Predictors with Response Variable

Below is a pairwise plot of the predictor variables versus the response variable PH. This should give us an idea of the predictors’ relationship to the response variable.



### Correlation

Below is heat map of correlations between variables. As you can see, some variables are more strongly correlated than others. Response variable PH is more strongly correlated to Mnf.Flow. Hyd.Pressure3 has a somewhat moderate negative correlation with Pressure.Vaccuum. There are some variables with significant positive correlations. Balling is strongly correlated with Balling.Lv1 (0.98), among others. Concerns about multicollinearity should be considered when selecting features for our models. We should avoid including pairs that are strongly correlated with each other.



### Identify Multicollinearity Problem

The vifcor function of the usdm package calculates variance inflation factor (VIF). This function excludes highly correlated variables from the set through stepwise procedure. This is function is used to deal with multicollinearity problem.

The function identified 6 variables from the 31 predictors that have collinearity problem. These are Balling, Bowl.Setpoint, Balling.Lvl,MFR, Hyd.Pressure3, and Alch.Rel.

## 6 variables from the 31 input variables have collinearity problem:   
##   
## Balling Bowl.Setpoint Balling.Lvl MFR Hyd.Pressure3 Alch.Rel   
##   
## After excluding the collinear variables, the linear correlation coefficients ranges between:   
## min correlation ( Pressure.Setpoint ~ PC.Volume ): 0.0001991472   
## max correlation ( Carb.Rel ~ Density ): 0.852689   
##   
## ---------- VIFs of the remained variables --------   
## Variables VIF  
## 1 Carb.Volume 17.159340  
## 2 Fill.Ounces 1.153764  
## 3 PC.Volume 1.685901  
## 4 Carb.Pressure 43.298925  
## 5 Carb.Temp 35.460787  
## 6 PSC 1.155980  
## 7 PSC.Fill 1.109449  
## 8 PSC.CO2 1.064636  
## 9 Mnf.Flow 4.262754  
## 10 Carb.Pressure1 1.434406  
## 11 Fill.Pressure 3.490621  
## 12 Hyd.Pressure1 2.935470  
## 13 Hyd.Pressure2 4.900597  
## 14 Hyd.Pressure4 1.752686  
## 15 Filler.Level 2.618103  
## 16 Filler.Speed 1.273500  
## 17 Temperature 1.151185  
## 18 Usage.cont 1.718776  
## 19 Carb.Flow 1.987496  
## 20 Density 4.499376  
## 21 Pressure.Vacuum 2.054866  
## 22 Oxygen.Filler 1.561878  
## 23 Pressure.Setpoint 3.300894  
## 24 Air.Pressurer 1.167671  
## 25 Carb.Rel 6.339500

Below, we remove the 6 variables with collinearity problem and run the vifcor function again after removing these 6 problematic variables. The output confirms that there are no more variables with collinearity problem. We will be dropping these 6 predictors from the data frame.

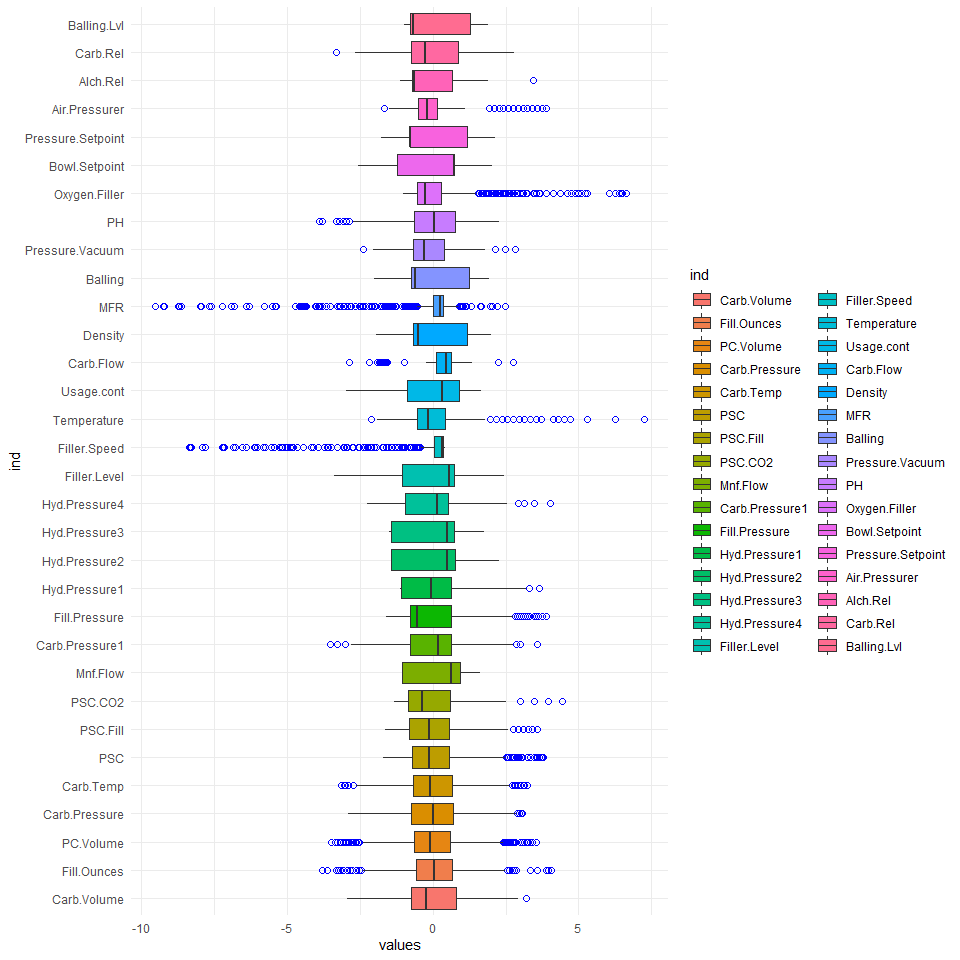
## No variable from the 25 input variables has collinearity problem.   
##   
## The linear correlation coefficients ranges between:   
## min correlation ( Density ~ Usage.cont ): -1.283718e-05   
## max correlation ( Carb.Rel ~ Density ): 0.8475287   
##   
## ---------- VIFs of the remained variables --------   
## Variables VIF  
## 1 Carb.Volume 16.772613  
## 2 Fill.Ounces 1.145540  
## 3 PC.Volume 1.644366  
## 4 Carb.Pressure 43.057978  
## 5 Carb.Temp 35.275573  
## 6 PSC 1.154486  
## 7 PSC.Fill 1.108167  
## 8 PSC.CO2 1.059999  
## 9 Mnf.Flow 4.164543  
## 10 Carb.Pressure1 1.499823  
## 11 Fill.Pressure 2.534785  
## 12 Hyd.Pressure1 2.934876  
## 13 Hyd.Pressure2 4.965940  
## 14 Hyd.Pressure4 1.952196  
## 15 Filler.Level 2.448071  
## 16 Filler.Speed 2.027988  
## 17 Temperature 1.208222  
## 18 Usage.cont 1.671926  
## 19 Carb.Flow 2.132049  
## 20 Density 4.355882  
## 21 Pressure.Vacuum 1.997420  
## 22 Oxygen.Filler 1.512114  
## 23 Pressure.Setpoint 2.650392  
## 24 Air.Pressurer 1.160505  
## 25 Carb.Rel 6.017929

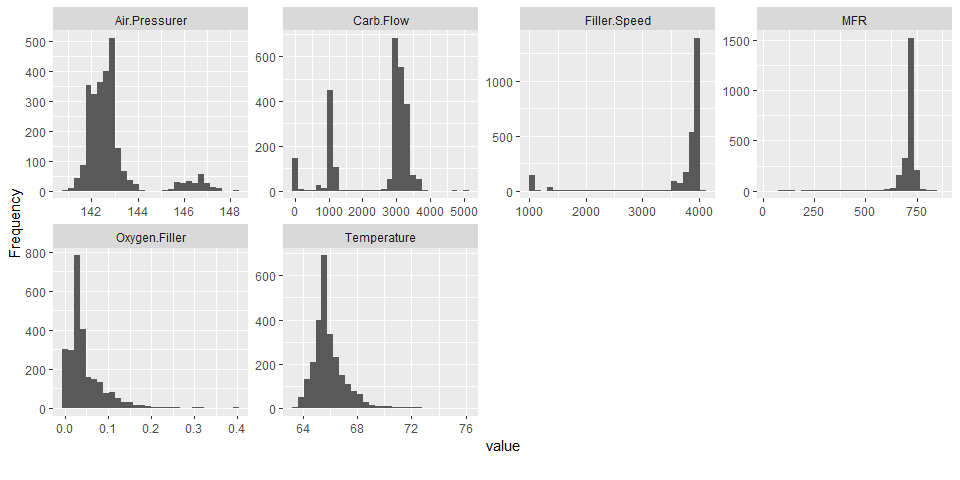
### Box Plots

Below is a scaled box plots of the variables. As you can see, there are some outliers in the data set. Usually, there are three general reasons for outliers. Reasons can include data entry or measurement error, sampling problems or unusual conditions, or simply a natural variation. As we do not have expert domain knowledge about the process that generated the data, we are unsure as to the nature of the reasons behind these outliers. As a result of this, it is hard do a reasonable assessment on whether to drop the outliers or not. If these outliers are a result of natural variation in the process, they capture valuable information that should probably be represented in the model.

For example, predictors MFR, Filler.Speed, and Oxygen.Filler appear to have some significant outliers.

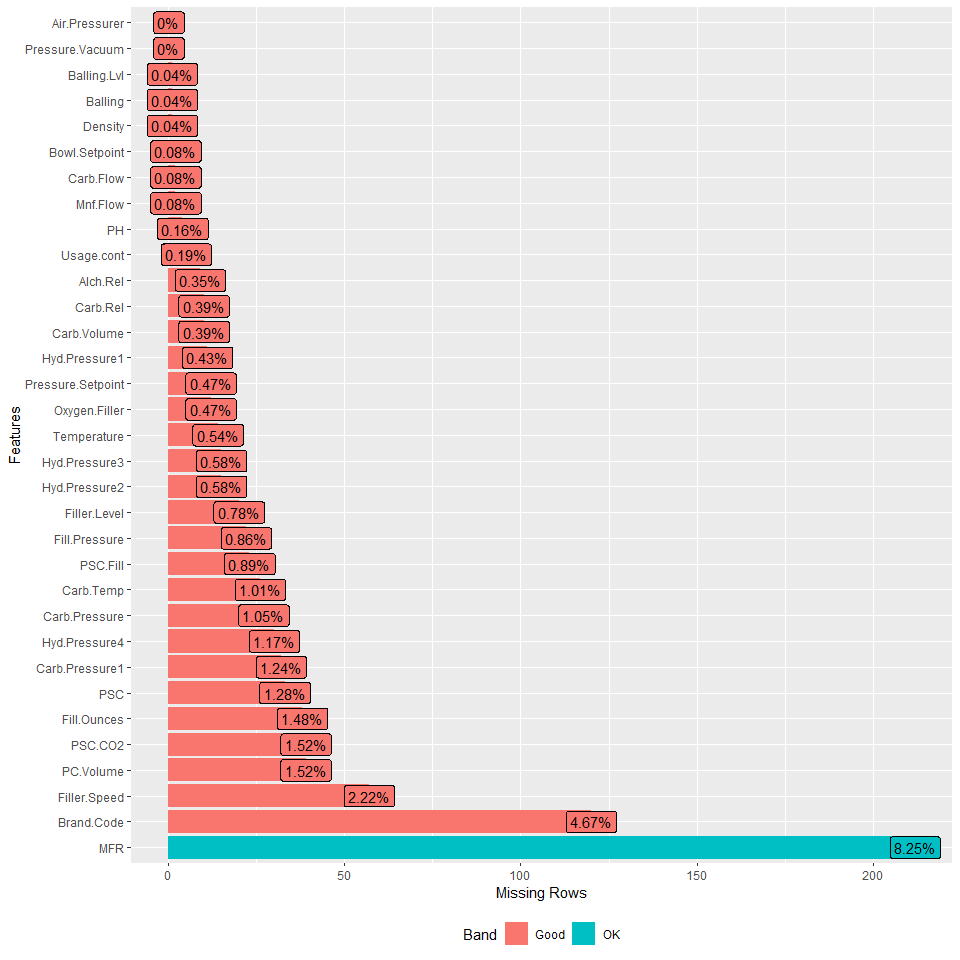
The histogram chart below shows the distribution of predictors that appear to have some outliers. Carb.Flow and Air.Pressure appear to have multimodal distributions.





### Missing Data

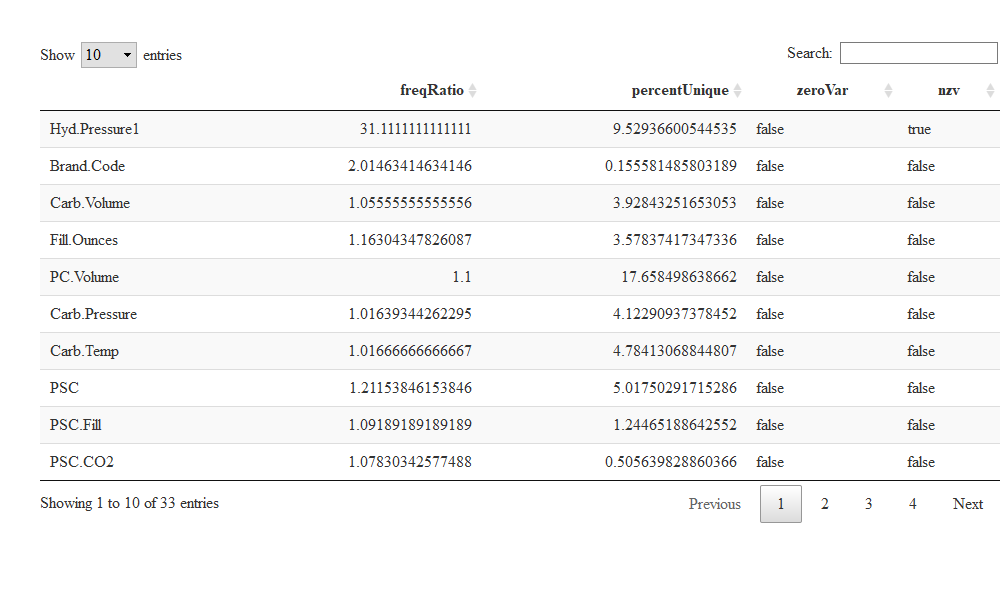
As you can see, approximately 8.25 percent of the rows are missing values for MFR with a count of 212 missing values. Bias is likely in analyses when missingness is more than 10%. So, based on this rule of thumb, it’s probably safe to impute missing values for MFR. Earlier data exploration revealed that the categorical variable Brand.Code has missing values. However, the plot\_missing function did not reflect this, which originally showed a 0% value for missingness as the missing data are not represented by the value NA. To reflect the accurate missing ratio of Brand.Code, the value NA was assigned. Approximately, 4.67 percent of rows are missing for Brand.Code. The response variable PH missing ration is 0.16 percent with a count of 4. The rest of the other variables have low missing ratios, which suggests that imputation can be applied safely.



### Near-Zero Variance

Below checks for predictors that show near zero-variance. These are predictors that do not vary much across observations and do not add much predictive information.

freqRatio is the ratio of frequencies for the most common value over the second most common value. percentUnique is the percentage of unique data points out of the total number of data points. zeroVar shows true if the predictor only has one distinct value; otherwise, false. nzv shows true if the predictor is a near zero variance predictor. The table below is sorted by nzv, and as you can see, predictor Hyd.Pressure1 has been determined to be a near zero-variance predictor. The rest of the predictors are not near zero-variance.



## 3. Data Preparation

Based on the data exploration findings, we learned that there are missing values in the data set, predictors that are strongly correlated with each other, predictors that show near zero-variance, and predictors with outliers.

The student\_data data frame is copied before we start modifying the data frame by dropping variables and imputing missing values. The student\_eval data is also processed in parallel with student\_data.

### Near Zero-Variance

During data exploration, it was determined that predictor Hyd.Pressure1 is a near zero-variance variable. This variable is dropped.

### Multicollinearity

During data exploration, the vifcor function identified 6 predictors that have collinearity problems. These variables are dropped.

Drop variables Balling, Bowl.Setpoint, Balling.Lvl,MFR, Hyd.Pressure3, Alch.Rel.

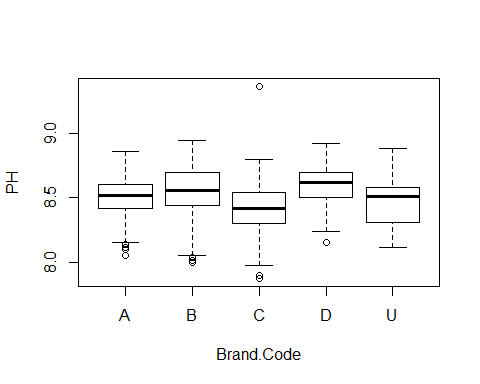
### Missing Values

* Drop observations with missing PH values
* Assign Brand.Code with blank values to U.
* Impute missing values.

The code below drops observations with missing PH values.

The blank Brand.Code values were explicitly assigned the value NA previously. The code below assigns the category U to blank values of Brand.Code.

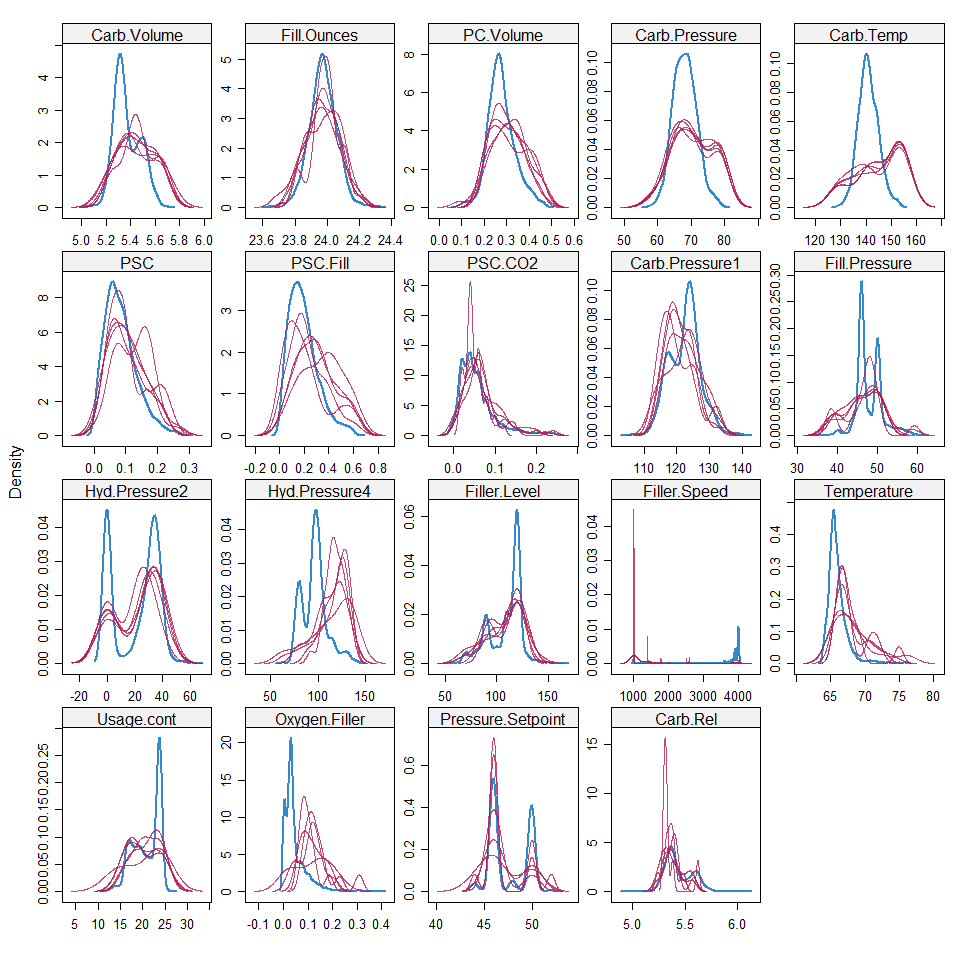
Below is box plot of response variable PH by Brand.Code grouping.



Impute missing values through mice package. Below is an excerpt from the mice package documentation.

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. The MICE algorithm can impute mixes of continuous, binary, unordered categorical and ordered categorical data. In addition, MICE can impute continuous two-level data, and maintain consistency between imputations by means of passive imputation. Many diagnostic plots are implemented to inspect the quality of the imputations. Apply the imputation function. The returns an S3 object of class mids (multiply imputed data set).

Below is density plot of the variables with imputed data.



The complete function of the mice package exports the imputed data. Update student\_data with the imputed results.

After the the data preparation steps applied above, the data frame is left with 2,567 observations and 26 variables. As you can see, we no longer have any missing values.

Data summary

|  |  |
| --- | --- |
| Name | student\_data |
| Number of rows | 2567 |
| Number of columns | 26 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| character | 1 |
| numeric | 25 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: character**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | min | max | empty | n\_unique | whitespace |
| Brand.Code | 0 | 1 | 1 | 1 | 0 | 5 | 0 |

**Variable type: numeric**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| Carb.Volume | 0 | 1 | 5.37 | 0.11 | 5.04 | 5.29 | 5.35 | 5.46 | 5.70 | ▁▆▇▅▁ |
| Fill.Ounces | 0 | 1 | 23.97 | 0.09 | 23.63 | 23.92 | 23.97 | 24.03 | 24.32 | ▁▂▇▂▁ |
| PC.Volume | 0 | 1 | 0.28 | 0.06 | 0.08 | 0.24 | 0.27 | 0.31 | 0.48 | ▁▃▇▂▁ |
| Carb.Pressure | 0 | 1 | 68.22 | 3.58 | 57.00 | 65.60 | 68.20 | 70.60 | 79.40 | ▁▅▇▃▁ |
| Carb.Temp | 0 | 1 | 141.13 | 4.14 | 128.60 | 138.40 | 140.80 | 143.80 | 154.00 | ▁▅▇▃▁ |
| PSC | 0 | 1 | 0.08 | 0.05 | 0.00 | 0.05 | 0.08 | 0.11 | 0.27 | ▆▇▃▁▁ |
| PSC.Fill | 0 | 1 | 0.20 | 0.12 | 0.00 | 0.10 | 0.18 | 0.26 | 0.62 | ▆▇▃▁▁ |
| PSC.CO2 | 0 | 1 | 0.06 | 0.04 | 0.00 | 0.02 | 0.04 | 0.08 | 0.24 | ▇▅▂▁▁ |
| Mnf.Flow | 0 | 1 | 24.63 | 119.50 | -100.20 | -100.00 | 70.20 | 140.80 | 229.40 | ▇▁▁▇▂ |
| Carb.Pressure1 | 0 | 1 | 122.57 | 4.73 | 105.60 | 119.00 | 123.20 | 125.40 | 140.20 | ▁▃▇▂▁ |
| Fill.Pressure | 0 | 1 | 47.91 | 3.19 | 34.60 | 46.00 | 46.40 | 50.00 | 60.40 | ▁▁▇▂▁ |
| Hyd.Pressure2 | 0 | 1 | 20.99 | 16.37 | 0.00 | 0.00 | 28.60 | 34.60 | 59.40 | ▇▂▇▃▁ |
| Hyd.Pressure4 | 0 | 1 | 96.50 | 13.36 | 62.00 | 86.00 | 96.00 | 102.00 | 142.00 | ▂▆▇▂▁ |
| Filler.Level | 0 | 1 | 109.26 | 15.70 | 55.80 | 98.40 | 118.40 | 120.00 | 161.20 | ▁▃▅▇▁ |
| Filler.Speed | 0 | 1 | 3640.93 | 837.01 | 998.00 | 3857.00 | 3980.00 | 3997.00 | 4030.00 | ▁▁▁▁▇ |
| Temperature | 0 | 1 | 65.97 | 1.39 | 63.60 | 65.20 | 65.60 | 66.40 | 76.20 | ▇▃▁▁▁ |
| Usage.cont | 0 | 1 | 20.99 | 2.98 | 12.08 | 18.37 | 21.78 | 23.75 | 25.90 | ▁▃▅▃▇ |
| Carb.Flow | 0 | 1 | 2472.70 | 1070.27 | 26.00 | 1169.00 | 3030.00 | 3188.00 | 5104.00 | ▂▅▆▇▁ |
| Density | 0 | 1 | 1.17 | 0.38 | 0.24 | 0.90 | 0.98 | 1.62 | 1.92 | ▁▅▇▂▆ |
| Pressure.Vacuum | 0 | 1 | -5.22 | 0.57 | -6.60 | -5.60 | -5.40 | -5.00 | -3.60 | ▂▇▆▂▁ |
| PH | 0 | 1 | 8.55 | 0.17 | 7.88 | 8.44 | 8.54 | 8.68 | 9.36 | ▁▅▇▂▁ |
| Oxygen.Filler | 0 | 1 | 0.05 | 0.05 | 0.00 | 0.02 | 0.03 | 0.06 | 0.40 | ▇▁▁▁▁ |
| Pressure.Setpoint | 0 | 1 | 47.61 | 2.04 | 44.00 | 46.00 | 46.00 | 50.00 | 52.00 | ▁▇▁▆▁ |
| Air.Pressurer | 0 | 1 | 142.83 | 1.21 | 140.80 | 142.20 | 142.60 | 143.00 | 148.20 | ▅▇▁▁▁ |
| Carb.Rel | 0 | 1 | 5.44 | 0.13 | 4.96 | 5.34 | 5.40 | 5.54 | 6.06 | ▁▇▇▂▁ |

### Outliers

We’re not dropping any observations with outliers.

### Create Dummy Variables for Categorical Variable Brand.Code

The code below creates dummy variables that encode categorical variables into 1 or 0. The dummy variables are added to the data frame, and the categorical variable is then removed from the data frame.

After the steps above were applied, the student\_data data frame has 30 variables.

Data summary

|  |  |
| --- | --- |
| Name | student\_data |
| Number of rows | 2567 |
| Number of columns | 30 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| numeric | 30 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: numeric**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| Carb.Volume | 0 | 1 | 5.37 | 0.11 | 5.04 | 5.29 | 5.35 | 5.46 | 5.70 | ▁▆▇▅▁ |
| Fill.Ounces | 0 | 1 | 23.97 | 0.09 | 23.63 | 23.92 | 23.97 | 24.03 | 24.32 | ▁▂▇▂▁ |
| PC.Volume | 0 | 1 | 0.28 | 0.06 | 0.08 | 0.24 | 0.27 | 0.31 | 0.48 | ▁▃▇▂▁ |
| Carb.Pressure | 0 | 1 | 68.22 | 3.58 | 57.00 | 65.60 | 68.20 | 70.60 | 79.40 | ▁▅▇▃▁ |
| Carb.Temp | 0 | 1 | 141.13 | 4.14 | 128.60 | 138.40 | 140.80 | 143.80 | 154.00 | ▁▅▇▃▁ |
| PSC | 0 | 1 | 0.08 | 0.05 | 0.00 | 0.05 | 0.08 | 0.11 | 0.27 | ▆▇▃▁▁ |
| PSC.Fill | 0 | 1 | 0.20 | 0.12 | 0.00 | 0.10 | 0.18 | 0.26 | 0.62 | ▆▇▃▁▁ |
| PSC.CO2 | 0 | 1 | 0.06 | 0.04 | 0.00 | 0.02 | 0.04 | 0.08 | 0.24 | ▇▅▂▁▁ |
| Mnf.Flow | 0 | 1 | 24.63 | 119.50 | -100.20 | -100.00 | 70.20 | 140.80 | 229.40 | ▇▁▁▇▂ |
| Carb.Pressure1 | 0 | 1 | 122.57 | 4.73 | 105.60 | 119.00 | 123.20 | 125.40 | 140.20 | ▁▃▇▂▁ |
| Fill.Pressure | 0 | 1 | 47.91 | 3.19 | 34.60 | 46.00 | 46.40 | 50.00 | 60.40 | ▁▁▇▂▁ |
| Hyd.Pressure2 | 0 | 1 | 20.99 | 16.37 | 0.00 | 0.00 | 28.60 | 34.60 | 59.40 | ▇▂▇▃▁ |
| Hyd.Pressure4 | 0 | 1 | 96.50 | 13.36 | 62.00 | 86.00 | 96.00 | 102.00 | 142.00 | ▂▆▇▂▁ |
| Filler.Level | 0 | 1 | 109.26 | 15.70 | 55.80 | 98.40 | 118.40 | 120.00 | 161.20 | ▁▃▅▇▁ |
| Filler.Speed | 0 | 1 | 3640.93 | 837.01 | 998.00 | 3857.00 | 3980.00 | 3997.00 | 4030.00 | ▁▁▁▁▇ |
| Temperature | 0 | 1 | 65.97 | 1.39 | 63.60 | 65.20 | 65.60 | 66.40 | 76.20 | ▇▃▁▁▁ |
| Usage.cont | 0 | 1 | 20.99 | 2.98 | 12.08 | 18.37 | 21.78 | 23.75 | 25.90 | ▁▃▅▃▇ |
| Carb.Flow | 0 | 1 | 2472.70 | 1070.27 | 26.00 | 1169.00 | 3030.00 | 3188.00 | 5104.00 | ▂▅▆▇▁ |
| Density | 0 | 1 | 1.17 | 0.38 | 0.24 | 0.90 | 0.98 | 1.62 | 1.92 | ▁▅▇▂▆ |
| Pressure.Vacuum | 0 | 1 | -5.22 | 0.57 | -6.60 | -5.60 | -5.40 | -5.00 | -3.60 | ▂▇▆▂▁ |
| PH | 0 | 1 | 8.55 | 0.17 | 7.88 | 8.44 | 8.54 | 8.68 | 9.36 | ▁▅▇▂▁ |
| Oxygen.Filler | 0 | 1 | 0.05 | 0.05 | 0.00 | 0.02 | 0.03 | 0.06 | 0.40 | ▇▁▁▁▁ |
| Pressure.Setpoint | 0 | 1 | 47.61 | 2.04 | 44.00 | 46.00 | 46.00 | 50.00 | 52.00 | ▁▇▁▆▁ |
| Air.Pressurer | 0 | 1 | 142.83 | 1.21 | 140.80 | 142.20 | 142.60 | 143.00 | 148.20 | ▅▇▁▁▁ |
| Carb.Rel | 0 | 1 | 5.44 | 0.13 | 4.96 | 5.34 | 5.40 | 5.54 | 6.06 | ▁▇▇▂▁ |
| Brand.A | 0 | 1 | 0.11 | 0.32 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▁ |
| Brand.B | 0 | 1 | 0.48 | 0.50 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 | ▇▁▁▁▇ |
| Brand.C | 0 | 1 | 0.12 | 0.32 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▁ |
| Brand.D | 0 | 1 | 0.24 | 0.43 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▂ |
| Brand.U | 0 | 1 | 0.05 | 0.21 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▁ |

## 4. Model Building

In this section, three categories of models were considered: (1) linear, (2) non-linear, and (3) tree-based. In total, nine models were considered. Linear models considered include PLS, Ridge, ENet, and Lasso. Non-linear models considered include KNN, MARS, and SVM. Tree-based models considered include random forest and GBM.

### Split Train and Test Data

The code below splits the data frame student\_datainto training and test sets with 80 percent of the observations going to the training set and 20 percent going to the test set. The training set is used to tune the models.

Build data frames to save training and test performance.

### Linear Models

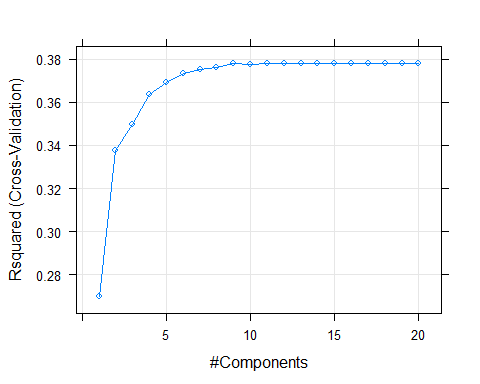
First set of models that will be done are the liner models. Linear models include **Partial Least Square** which is part of the pls library, There are three models that are from the elasticnect library and those models are **Ridge**, **Elastic Net**, and **Lasso**.

#### a. Partial Least Square (PLS)

Partial Least Squares is a type of technique that minimizes the predictors into smaller sets of components that are uncorrelated conducts partial least regression on them, and it does not do it on the original data. In PLS predictors can be highly correlated. Predictors are measured with error as it does not assume that the data is fixed. PLS uses Rsquared to select the optimal model. Here we build the PLS model.

## Partial Least Squares   
##   
## 2055 samples  
## 29 predictor  
##   
## Pre-processing: centered (29), scaled (29)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 1852, 1848, 1850, 1849, 1849, 1850, ...   
## Resampling results across tuning parameters:  
##   
## ncomp RMSE Rsquared MAE   
## 1 0.1475444 0.2697501 0.1163209  
## 2 0.1403038 0.3375482 0.1096748  
## 3 0.1390244 0.3495563 0.1093433  
## 4 0.1374810 0.3637951 0.1078142  
## 5 0.1368743 0.3694355 0.1067738  
## 6 0.1364368 0.3735789 0.1063230  
## 7 0.1362655 0.3752907 0.1064270  
## 8 0.1361407 0.3763309 0.1062417  
## 9 0.1359896 0.3777980 0.1060292  
## 10 0.1360214 0.3774610 0.1059708  
## 11 0.1359612 0.3778966 0.1059615  
## 12 0.1359398 0.3780788 0.1059709  
## 13 0.1359229 0.3782254 0.1059311  
## 14 0.1359397 0.3780868 0.1059473  
## 15 0.1359419 0.3780920 0.1059436  
## 16 0.1359748 0.3778372 0.1059720  
## 17 0.1359439 0.3781005 0.1059759  
## 18 0.1359370 0.3781625 0.1059648  
## 19 0.1359326 0.3781984 0.1059591  
## 20 0.1359354 0.3781738 0.1059602  
##   
## Rsquared was used to select the optimal model using the largest value.  
## The final value used for the model was ncomp = 13.

Below is the plot of the PLS model of the RSquared data.



Below is the performance of PLS model on the test set, which is similar to the training set.

## RMSE Rsquared MAE   
## 0.1379208 0.3750179 0.1071041

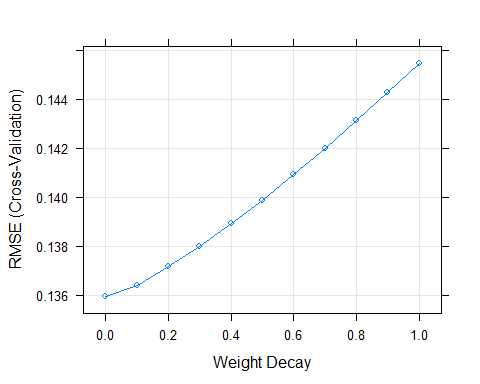
Save training and test performance.

### b. Ridge

Ridge regression is a technique that shrinks the regression coefficients, that makes variables, have an outcome with minor contributions, and makes the coefficients close to zero. The ridge model determines is RMSE by selecting the optimal model. Here we buld the ridge model.

## Ridge Regression   
##   
## 2055 samples  
## 29 predictor  
##   
## Pre-processing: centered (29), scaled (29)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 1852, 1848, 1850, 1849, 1849, 1850, ...   
## Resampling results across tuning parameters:  
##   
## lambda RMSE Rsquared MAE   
## 0.0 0.1359338 0.3781874 0.1059620  
## 0.1 0.1364204 0.3740103 0.1062830  
## 0.2 0.1371790 0.3685452 0.1068324  
## 0.3 0.1380115 0.3636493 0.1074350  
## 0.4 0.1389151 0.3593017 0.1080551  
## 0.5 0.1398857 0.3554164 0.1087703  
## 0.6 0.1409165 0.3519149 0.1095544  
## 0.7 0.1419997 0.3487341 0.1103864  
## 0.8 0.1431270 0.3458245 0.1112790  
## 0.9 0.1442907 0.3431470 0.1121981  
## 1.0 0.1454833 0.3406707 0.1131479  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was lambda = 0.

Below is the plot of the Ridge model of the RSquared data.



Below is the performance of ridge model on the test set, which is similar to the training set.

## RMSE Rsquared MAE   
## 0.1379925 0.3744831 0.1070752

Save training and test performance.

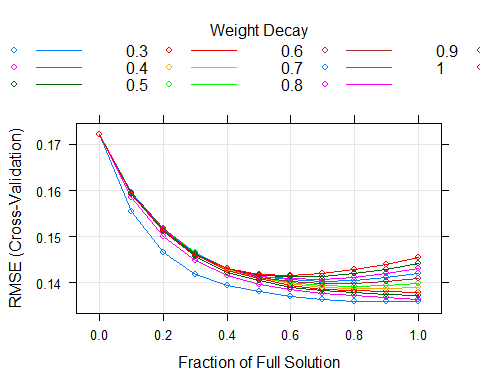
#### c. Elastic Net (ENet)

Elastic Net creates a regression model that is then penalized with the L1-norm and L2-norm. Enet effectively shrink coefficients and it also sets some coefficients to zero. Elastic Net Model uses RMSE uses to select the optimal model. Here, we build the Elastic Net model.

set.seed(1)  
enet\_model <- train(x=train\_X,  
 y= train\_Y,  
 method = "enet",  
 tuneGrid=expand.grid(.fraction = seq(0, 1, by=0.1),   
 .lambda = seq(0, 1, by=0.1)),  
 trControl = trainControl(method='cv') ,  
 preProc = c("center", "scale")  
 )  
enet\_model

## Elasticnet   
##   
## 2055 samples  
## 29 predictor  
##   
## Pre-processing: centered (29), scaled (29)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 1852, 1848, 1850, 1849, 1849, 1850, ...   
## Resampling results across tuning parameters:  
##   
## lambda fraction RMSE Rsquared MAE   
## 0.0 0.0 0.1720592 NaN 0.1384415  
## 0.0 0.1 0.1553818 0.2602260 0.1237906  
## 0.0 0.2 0.1466019 0.3063326 0.1160760  
## 0.0 0.3 0.1418796 0.3347373 0.1119024  
## 0.0 0.4 0.1394502 0.3503005 0.1096758  
## 0.0 0.5 0.1380169 0.3609219 0.1083283  
## 0.0 0.6 0.1370389 0.3687844 0.1072676  
## 0.0 0.7 0.1364295 0.3736785 0.1065075  
## 0.0 0.8 0.1360504 0.3767977 0.1061019  
## 0.0 0.9 0.1359254 0.3779960 0.1059529  
## 0.0 1.0 0.1359338 0.3781874 0.1059620  
## 0.1 0.0 0.1720592 NaN 0.1384415  
## 0.1 0.1 0.1583835 0.2419125 0.1263453  
## 0.1 0.2 0.1501410 0.2889712 0.1192044  
## 0.1 0.3 0.1447534 0.3165224 0.1144000  
## 0.1 0.4 0.1416784 0.3338812 0.1116758  
## 0.1 0.5 0.1397321 0.3469424 0.1099362  
## 0.1 0.6 0.1385626 0.3547983 0.1087232  
## 0.1 0.7 0.1377892 0.3611276 0.1079526  
## 0.1 0.8 0.1371588 0.3667610 0.1071974  
## 0.1 0.9 0.1367304 0.3707961 0.1066688  
## 0.1 1.0 0.1364204 0.3740103 0.1062830  
## 0.2 0.0 0.1720592 NaN 0.1384415  
## 0.2 0.1 0.1591009 0.2353980 0.1269506  
## 0.2 0.2 0.1510970 0.2845927 0.1200476  
## 0.2 0.3 0.1456469 0.3111884 0.1152072  
## 0.2 0.4 0.1423120 0.3279708 0.1122730  
## 0.2 0.5 0.1404181 0.3405321 0.1105412  
## 0.2 0.6 0.1390500 0.3501421 0.1091622  
## 0.2 0.7 0.1382867 0.3562557 0.1082711  
## 0.2 0.8 0.1378836 0.3605573 0.1077646  
## 0.2 0.9 0.1374722 0.3649844 0.1072444  
## 0.2 1.0 0.1371790 0.3685452 0.1068324  
## 0.3 0.0 0.1720592 NaN 0.1384415  
## 0.3 0.1 0.1594690 0.2315372 0.1272617  
## 0.3 0.2 0.1515975 0.2823682 0.1204822  
## 0.3 0.3 0.1461786 0.3073859 0.1156826  
## 0.3 0.4 0.1427266 0.3243036 0.1126391  
## 0.3 0.5 0.1408424 0.3356561 0.1109037  
## 0.3 0.6 0.1394810 0.3459020 0.1095137  
## 0.3 0.7 0.1386381 0.3530667 0.1085536  
## 0.3 0.8 0.1383166 0.3575235 0.1079813  
## 0.3 0.9 0.1381937 0.3603796 0.1077162  
## 0.3 1.0 0.1380115 0.3636493 0.1074350  
## 0.4 0.0 0.1720592 NaN 0.1384415  
## 0.4 0.1 0.1596282 0.2296124 0.1273963  
## 0.4 0.2 0.1518339 0.2813473 0.1206844  
## 0.4 0.3 0.1464405 0.3049629 0.1159124  
## 0.4 0.4 0.1429889 0.3213293 0.1128712  
## 0.4 0.5 0.1410509 0.3328259 0.1110791  
## 0.4 0.6 0.1398562 0.3421265 0.1097934  
## 0.4 0.7 0.1389961 0.3500782 0.1087926  
## 0.4 0.8 0.1387387 0.3547894 0.1082554  
## 0.4 0.9 0.1388278 0.3572329 0.1081374  
## 0.4 1.0 0.1389151 0.3593017 0.1080551  
## 0.5 0.0 0.1720592 NaN 0.1384415  
## 0.5 0.1 0.1596396 0.2291676 0.1274081  
## 0.5 0.2 0.1518751 0.2808993 0.1207097  
## 0.5 0.3 0.1464945 0.3035506 0.1159641  
## 0.5 0.4 0.1431023 0.3192552 0.1129726  
## 0.5 0.5 0.1411614 0.3311951 0.1111688  
## 0.5 0.6 0.1401334 0.3393502 0.1099751  
## 0.5 0.7 0.1393698 0.3473499 0.1090133  
## 0.5 0.8 0.1392186 0.3520623 0.1085895  
## 0.5 0.9 0.1395225 0.3541140 0.1086610  
## 0.5 1.0 0.1398857 0.3554164 0.1087703  
## 0.6 0.0 0.1720592 NaN 0.1384415  
## 0.6 0.1 0.1595818 0.2294755 0.1273680  
## 0.6 0.2 0.1517956 0.2807151 0.1206217  
## 0.6 0.3 0.1464390 0.3024651 0.1159194  
## 0.6 0.4 0.1431309 0.3177463 0.1129841  
## 0.6 0.5 0.1412323 0.3299454 0.1112058  
## 0.6 0.6 0.1403891 0.3370847 0.1101157  
## 0.6 0.7 0.1397688 0.3448843 0.1092316  
## 0.6 0.8 0.1398222 0.3489679 0.1090139  
## 0.6 0.9 0.1402964 0.3510050 0.1092505  
## 0.6 1.0 0.1409165 0.3519149 0.1095544  
## 0.7 0.0 0.1720592 NaN 0.1384415  
## 0.7 0.1 0.1595070 0.2299944 0.1273148  
## 0.7 0.2 0.1516582 0.2805581 0.1204783  
## 0.7 0.3 0.1463377 0.3015587 0.1158287  
## 0.7 0.4 0.1431205 0.3166373 0.1129535  
## 0.7 0.5 0.1413141 0.3287408 0.1112351  
## 0.7 0.6 0.1406710 0.3350319 0.1102612  
## 0.7 0.7 0.1402578 0.3422018 0.1095345  
## 0.7 0.8 0.1405061 0.3458989 0.1095020  
## 0.7 0.9 0.1411291 0.3479915 0.1098824  
## 0.7 1.0 0.1419997 0.3487341 0.1103864  
## 0.8 0.0 0.1720592 NaN 0.1384415  
## 0.8 0.1 0.1594168 0.2307456 0.1272502  
## 0.8 0.2 0.1514951 0.2803787 0.1203175  
## 0.8 0.3 0.1462148 0.3007756 0.1157205  
## 0.8 0.4 0.1430926 0.3157634 0.1129163  
## 0.8 0.5 0.1414196 0.3274490 0.1112628  
## 0.8 0.6 0.1409653 0.3331832 0.1104267  
## 0.8 0.7 0.1408215 0.3393204 0.1099235  
## 0.8 0.8 0.1412555 0.3428038 0.1100576  
## 0.8 0.9 0.1420265 0.3450480 0.1105468  
## 0.8 1.0 0.1431270 0.3458245 0.1112790  
## 0.9 0.0 0.1720592 NaN 0.1384415  
## 0.9 0.1 0.1593101 0.2315884 0.1271679  
## 0.9 0.2 0.1513188 0.2801547 0.1201471  
## 0.9 0.3 0.1460837 0.3001093 0.1156089  
## 0.9 0.4 0.1430672 0.3149814 0.1128717  
## 0.9 0.5 0.1415529 0.3261136 0.1113019  
## 0.9 0.6 0.1412980 0.3312954 0.1106343  
## 0.9 0.7 0.1414542 0.3363190 0.1103801  
## 0.9 0.8 0.1420483 0.3398880 0.1106476  
## 0.9 0.9 0.1429789 0.3422458 0.1112692  
## 0.9 1.0 0.1442907 0.3431470 0.1121981  
## 1.0 0.0 0.1720592 NaN 0.1384415  
## 1.0 0.1 0.1592008 0.2323180 0.1270817  
## 1.0 0.2 0.1511442 0.2798793 0.1199803  
## 1.0 0.3 0.1459585 0.2994972 0.1154996  
## 1.0 0.4 0.1430609 0.3141531 0.1128287  
## 1.0 0.5 0.1417152 0.3247313 0.1113631  
## 1.0 0.6 0.1416641 0.3293841 0.1108682  
## 1.0 0.7 0.1420963 0.3335644 0.1108523  
## 1.0 0.8 0.1428632 0.3372221 0.1112651  
## 1.0 0.9 0.1439644 0.3396531 0.1120260  
## 1.0 1.0 0.1454833 0.3406707 0.1131479  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were fraction = 0.9 and lambda = 0.

Below is the plot of the ENet model of the RSquared data.



Below is performance of ENet model on the test set, which is similar to training set.

## RMSE Rsquared MAE   
## 0.1378915 0.3748911 0.1070772

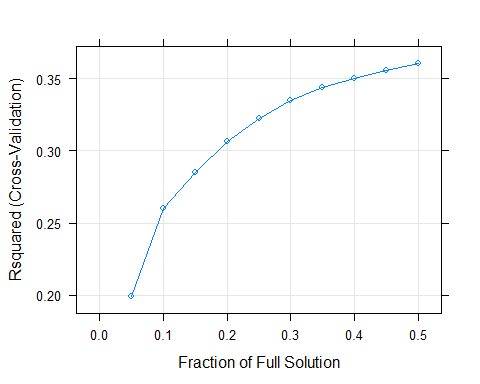
Save training and test performance.

#### d. Lasso

Lasso stands for Least Absolute Shrinkage and Selection Operator. As its name suggest Lasso shrinks the coeeficients of the regression toward zero which then penalizes the regression model to L1-norm, the sum of the absolute coefficients. Lasso uses the Rsquared in order to select the optimal model. Here, we build the lasso model.

## The lasso   
##   
## 2055 samples  
## 29 predictor  
##   
## Pre-processing: centered (29), scaled (29)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 1852, 1848, 1850, 1849, 1849, 1850, ...   
## Resampling results across tuning parameters:  
##   
## fraction RMSE Rsquared MAE   
## 0.00 0.1720592 NaN 0.1384415  
## 0.05 0.1623547 0.1988617 0.1297894  
## 0.10 0.1553818 0.2602260 0.1237906  
## 0.15 0.1503884 0.2851764 0.1194232  
## 0.20 0.1466019 0.3063326 0.1160760  
## 0.25 0.1437345 0.3223985 0.1135287  
## 0.30 0.1418796 0.3347373 0.1119024  
## 0.35 0.1404829 0.3437027 0.1106357  
## 0.40 0.1394502 0.3503005 0.1096758  
## 0.45 0.1386665 0.3560308 0.1089289  
## 0.50 0.1380169 0.3609219 0.1083283  
##   
## Rsquared was used to select the optimal model using the largest value.  
## The final value used for the model was fraction = 0.5.

Below is the plot of the Lasso model of the RSquared data.



Below is performance of lasso model on test set, which is similar to training set.

## RMSE Rsquared MAE   
## 0.1390828 0.3634917 0.1093774

Save training and test performance.

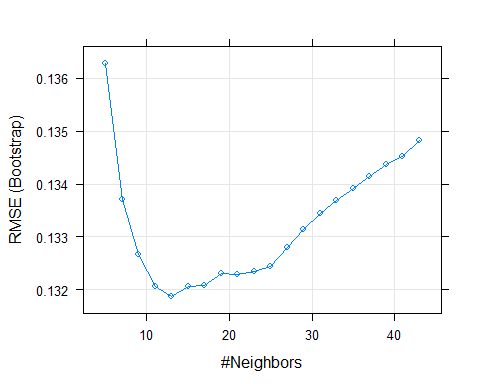
### Non-Linear Models

Next set of models that will be done are the non-linear models. The non\_linear models consist of **K-Nearest Neighbors (KNN)**, **Support Vector Machines (SVM)**,and **Multivariate Adaptive Regression Splines (MARS)**. SVM model and KNN model both use the caret library, while MARS uses the earth library.

**a. K-nearest Neighbors (KNN)**

Here, we build the K-nearest Neighbor model. RMSE metric was used to select the optimal model.

## k-Nearest Neighbors   
##   
## 2055 samples  
## 29 predictor  
##   
## Pre-processing: centered (29), scaled (29)   
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 2055, 2055, 2055, 2055, 2055, 2055, ...   
## Resampling results across tuning parameters:  
##   
## k RMSE Rsquared MAE   
## 5 0.1362882 0.4049885 0.10062242  
## 7 0.1337103 0.4149618 0.09969496  
## 9 0.1326581 0.4182120 0.09957466  
## 11 0.1320620 0.4198883 0.09943211  
## 13 0.1318634 0.4195381 0.09961616  
## 15 0.1320655 0.4168911 0.09997505  
## 17 0.1320747 0.4157675 0.10024156  
## 19 0.1323075 0.4133827 0.10051829  
## 21 0.1322902 0.4130841 0.10065080  
## 23 0.1323428 0.4121573 0.10085459  
## 25 0.1324471 0.4108564 0.10100372  
## 27 0.1328053 0.4074699 0.10136790  
## 29 0.1331310 0.4044819 0.10170746  
## 31 0.1334328 0.4016739 0.10201870  
## 33 0.1336948 0.3991875 0.10226727  
## 35 0.1339202 0.3969727 0.10242136  
## 37 0.1341456 0.3948643 0.10254753  
## 39 0.1343751 0.3926863 0.10269663  
## 41 0.1345295 0.3912840 0.10281018  
## 43 0.1348283 0.3887038 0.10305314  
##   
## RMSE was used to select the optimal model using the smallest value.  
## The final value used for the model was k = 13.



Below is the performance of the KNN model on the test set, which is similar to the training set.

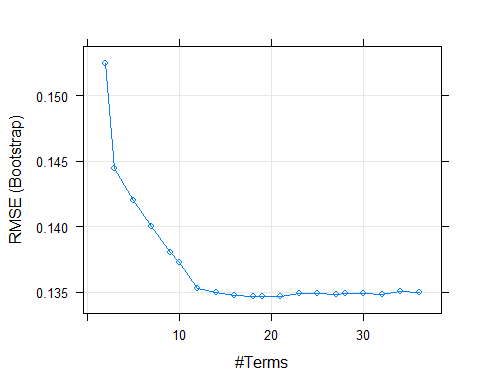
## RMSE Rsquared MAE   
## 0.12439734 0.49837261 0.09557478

Save training and test performance.

#### b. Multivariate Adaptive Regression Splines (MARS)

Here, we build the MARS model. RMSE metric was used to select the optimal model.

## Multivariate Adaptive Regression Spline   
##   
## 2055 samples  
## 29 predictor  
##   
## Pre-processing: centered (29), scaled (29)   
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 2055, 2055, 2055, 2055, 2055, 2055, ...   
## Resampling results across tuning parameters:  
##   
## nprune RMSE Rsquared MAE   
## 2 0.1525180 0.2094742 0.1197446  
## 3 0.1444375 0.2902816 0.1130738  
## 5 0.1420249 0.3146692 0.1104466  
## 7 0.1400350 0.3341658 0.1083860  
## 9 0.1380296 0.3530826 0.1063602  
## 10 0.1372706 0.3603058 0.1057218  
## 12 0.1352924 0.3785790 0.1038026  
## 14 0.1349637 0.3823557 0.1031480  
## 16 0.1347622 0.3847275 0.1029210  
## 18 0.1346738 0.3863708 0.1027334  
## 19 0.1346900 0.3865554 0.1027726  
## 21 0.1346389 0.3878520 0.1027844  
## 23 0.1348787 0.3865977 0.1028585  
## 25 0.1349155 0.3868247 0.1029269  
## 27 0.1348331 0.3879429 0.1027531  
## 28 0.1349035 0.3875977 0.1027481  
## 30 0.1349271 0.3878905 0.1026746  
## 32 0.1348401 0.3890486 0.1025479  
## 34 0.1350289 0.3876740 0.1026852  
## 36 0.1350152 0.3880050 0.1026450  
##   
## Tuning parameter 'degree' was held constant at a value of 1  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were nprune = 21 and degree = 1.



Below is the performance of the MARS model on the test set, which is similar to the training set.

## RMSE Rsquared MAE   
## 0.1340052 0.4103099 0.1027201

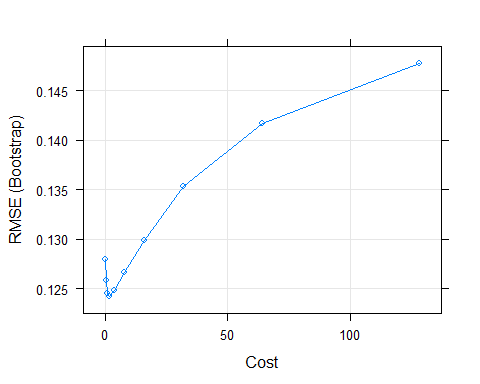
Save training and test performance.

## Warning in `[<-.factor`(`\*tmp\*`, iseq, value = "MARS"): invalid factor level, NA  
## generated  
  
## Warning in `[<-.factor`(`\*tmp\*`, iseq, value = "MARS"): invalid factor level, NA  
## generated

#### c. Support Vector Machines (SVM)

Here, we build the SVM model. RMSE was used to select the optimal model.

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 2055 samples  
## 29 predictor  
##   
## Pre-processing: centered (29), scaled (29)   
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 2055, 2055, 2055, 2055, 2055, 2055, ...   
## Resampling results across tuning parameters:  
##   
## C RMSE Rsquared MAE   
## 0.25 0.1279841 0.4484411 0.09512726  
## 0.50 0.1258799 0.4649780 0.09305339  
## 1.00 0.1245581 0.4760033 0.09180649  
## 2.00 0.1241910 0.4810630 0.09162848  
## 4.00 0.1248802 0.4802107 0.09250449  
## 8.00 0.1266428 0.4735991 0.09417736  
## 16.00 0.1298438 0.4592186 0.09681733  
## 32.00 0.1353447 0.4332973 0.10101334  
## 64.00 0.1416916 0.4051859 0.10565725  
## 128.00 0.1477701 0.3804602 0.11026098  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.02369026  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were sigma = 0.02369026 and C = 2.



Below is the performance of the SVM model on the test set, which is similar to the training set.

## RMSE Rsquared MAE   
## 0.1172226 0.5556294 0.0859758

Save training and test performance.

## Warning in `[<-.factor`(`\*tmp\*`, iseq, value = "SVM"): invalid factor level, NA  
## generated  
  
## Warning in `[<-.factor`(`\*tmp\*`, iseq, value = "SVM"): invalid factor level, NA  
## generated

### Tree-Based Models

Next set of models is part are the tree based models which consist of **Random Forest(RF)**, and the **Gradient Boosting Machine**. RF Model uses the randomforest library while GMB Model uses the caret library.

#### a. Random Forest(RF)

Here, we build the random forest model with ntree set to 100.

##   
## Call:  
## randomForest(x = train\_X, y = train\_Y, ntree = 100)   
## Type of random forest: regression  
## Number of trees: 100  
## No. of variables tried at each split: 9  
##   
## Mean of squared residuals: 0.01118717  
## % Var explained: 62.25

Below is the performance of the random forest model on the test set, which is similar to the training set.

## RMSE Rsquared MAE   
## 0.09796695 0.70761740 0.07363158

Save training and test performance.

#### b. Gradient Boosting Machine (GBM)

Here, we build the gradient boosting machine model. RMSE was used to select the optimal model.

## Stochastic Gradient Boosting   
##   
## 2055 samples  
## 29 predictor  
##   
## No pre-processing  
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 2055, 2055, 2055, 2055, 2055, 2055, ...   
## Resampling results across tuning parameters:  
##   
## interaction.depth n.trees RMSE Rsquared MAE   
## 1 100 0.1339239 0.3950323 0.10477222  
## 1 200 0.1322563 0.4050135 0.10263092  
## 1 300 0.1318939 0.4076331 0.10174900  
## 1 400 0.1320518 0.4071191 0.10147871  
## 1 500 0.1321515 0.4071142 0.10125885  
## 1 600 0.1323119 0.4068150 0.10111531  
## 1 700 0.1323415 0.4073659 0.10097784  
## 1 800 0.1326016 0.4062089 0.10108686  
## 1 900 0.1327688 0.4056856 0.10109589  
## 1 1000 0.1328946 0.4055744 0.10108683  
## 3 100 0.1250318 0.4692545 0.09592086  
## 3 200 0.1222785 0.4910517 0.09249375  
## 3 300 0.1212470 0.5000584 0.09118180  
## 3 400 0.1204244 0.5072077 0.09034816  
## 3 500 0.1199141 0.5117855 0.08975774  
## 3 600 0.1195945 0.5149241 0.08944118  
## 3 700 0.1192920 0.5177363 0.08916290  
## 3 800 0.1191023 0.5199261 0.08898788  
## 3 900 0.1191735 0.5199473 0.08894566  
## 3 1000 0.1190374 0.5212941 0.08883902  
## 5 100 0.1206083 0.5056220 0.09133563  
## 5 200 0.1180559 0.5254546 0.08857157  
## 5 300 0.1170877 0.5335114 0.08756047  
## 5 400 0.1166022 0.5377664 0.08695262  
## 5 500 0.1162364 0.5410361 0.08651740  
## 5 600 0.1161295 0.5422862 0.08635330  
## 5 700 0.1161070 0.5427562 0.08626253  
## 5 800 0.1159675 0.5441361 0.08611505  
## 5 900 0.1158866 0.5450034 0.08606533  
## 5 1000 0.1158797 0.5452960 0.08602816  
##   
## Tuning parameter 'shrinkage' was held constant at a value of 0.1  
##   
## Tuning parameter 'n.minobsinnode' was held constant at a value of 5  
## RMSE was used to select the optimal model using the smallest value.  
## The final values used for the model were n.trees = 1000, interaction.depth =  
## 5, shrinkage = 0.1 and n.minobsinnode = 5.

Below is the performance of the GMB model on the test set, which is similar to the training set.

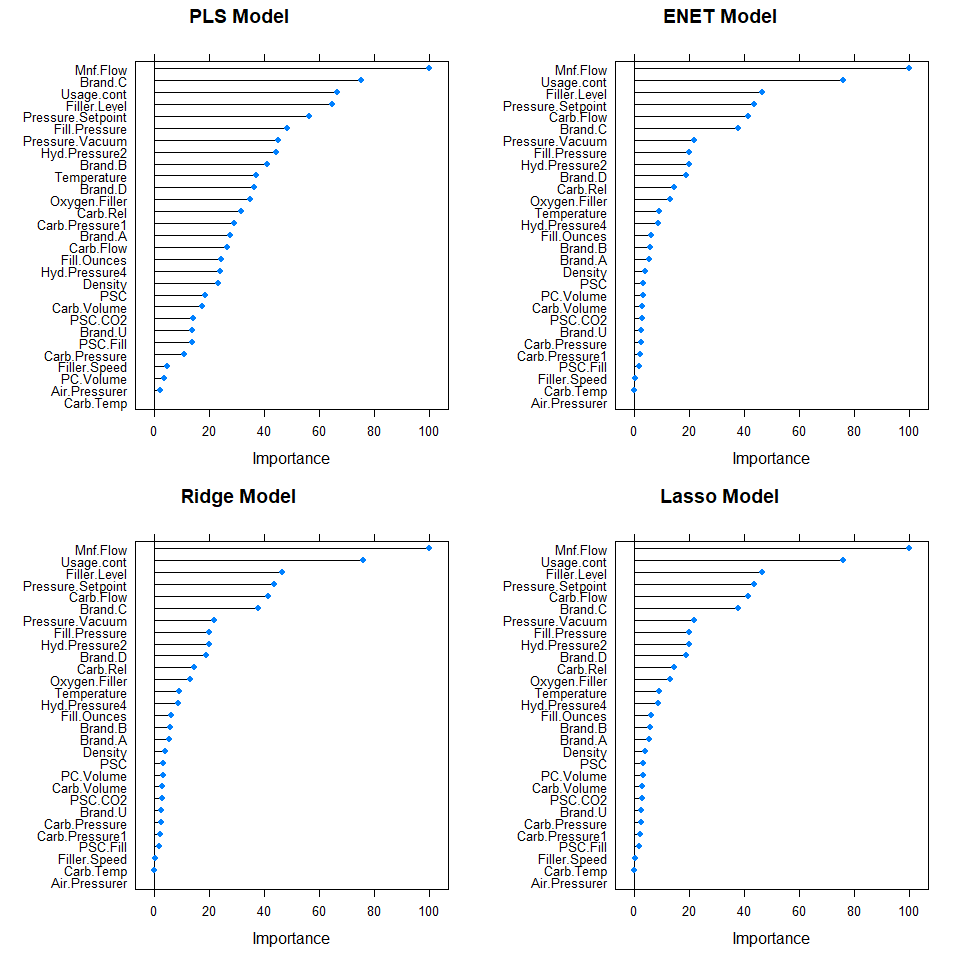
## RMSE Rsquared MAE   
## 0.1041354 0.6412081 0.0802140

## 5. Variable of Importance

Predictor Mnf.flow appears to be the most important predictor across the nine different models considered.

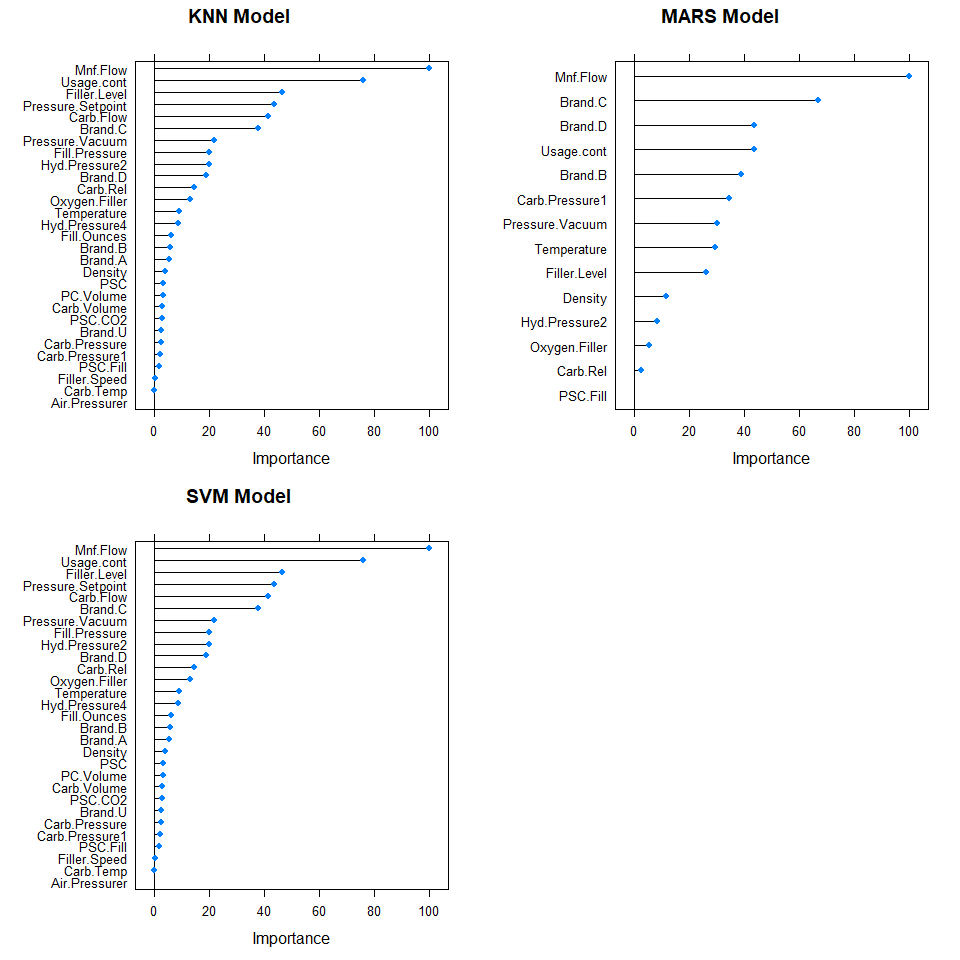
#### a. Linear Models

The chart below shows the levels of importance for each predictor. For all linear models considered, Mnf.flow shows as the most important predictor.



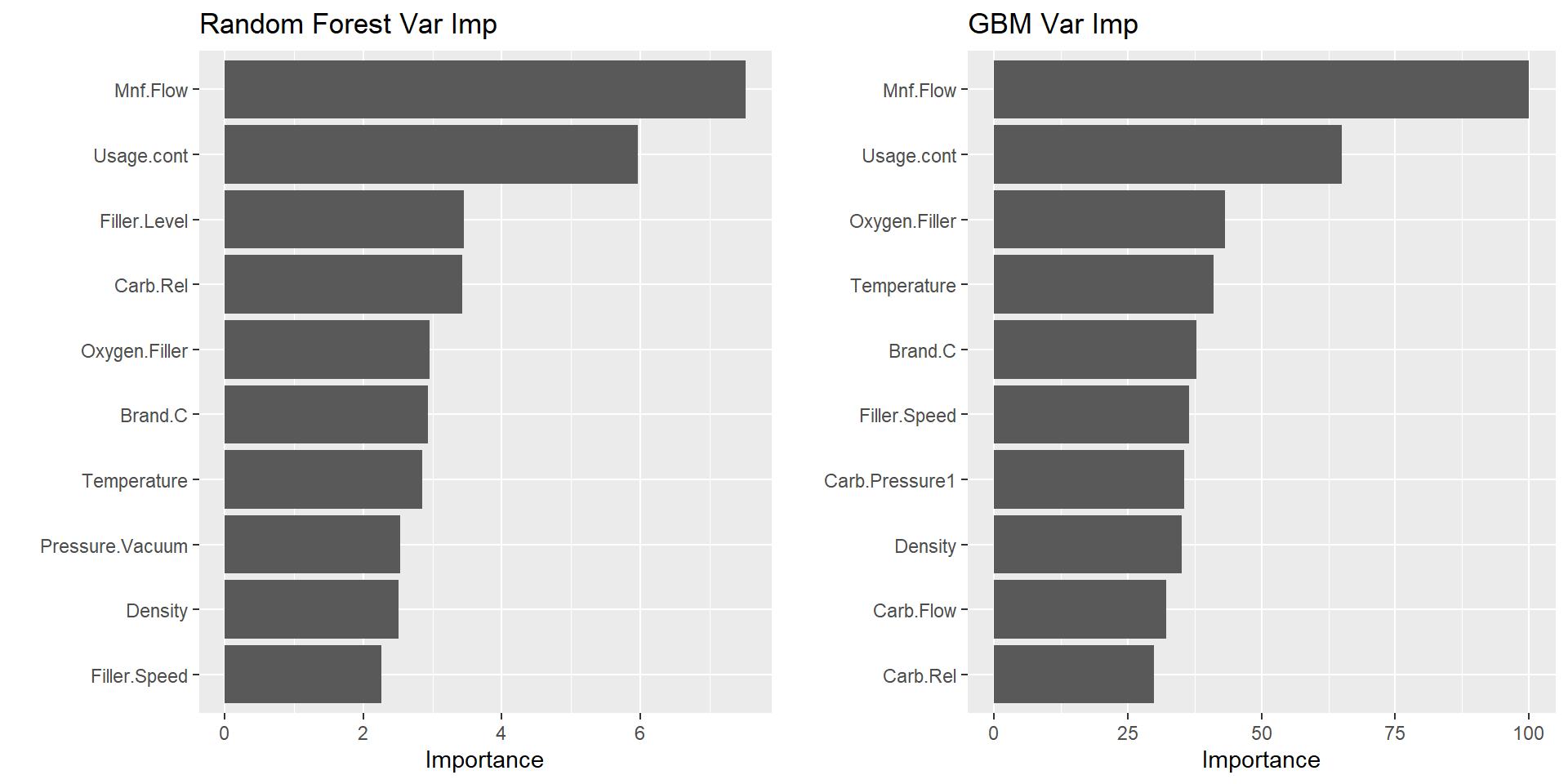
#### b. Non-Linear Models

The chart below shows the levels of importance for each predictor. For all the non-linear models, Mnf.Flow shows as the most important predictor.



#### c. Tree-Based Models

The chart below shows the levels of importance for each predictor. For all tree-based models considered, Mnf.flow is the most important predictor.

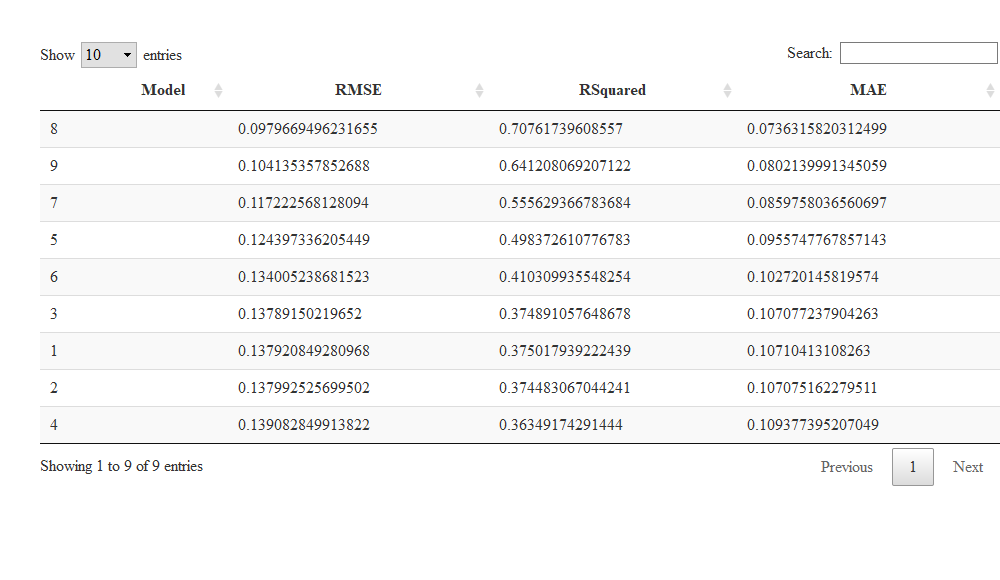


## 6. Model Selection

Nine different models were considered from three different classes of models, which includes linear, non-linear, and tree-based models. For model selection, we’ve decided to compare the performance of each model on the test set using RMSE. Below is a table that shows how well each model did based on RMSE, RSquared, and MAE.

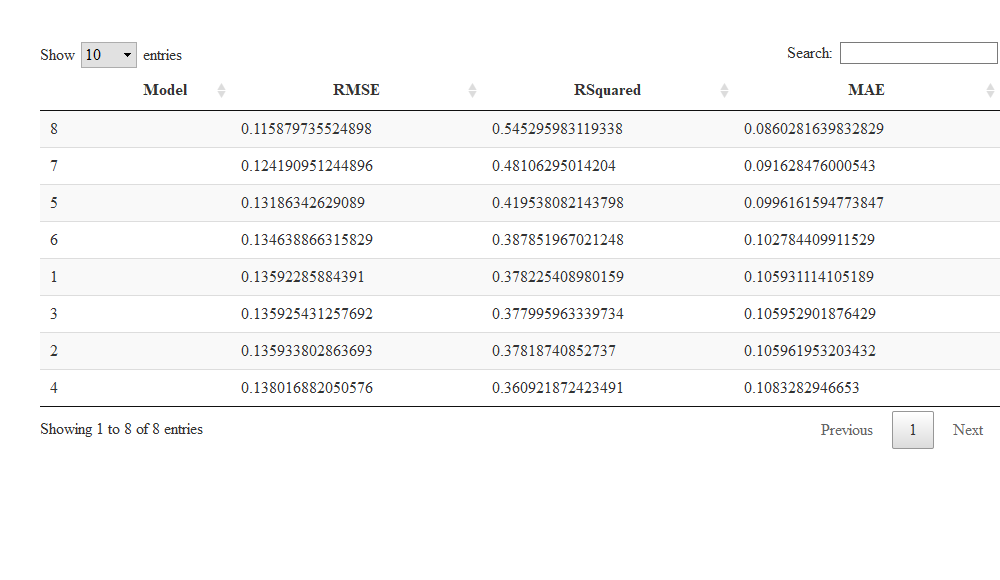
As you can see, Random Forest model is the best performing model based on lowest RMSE. It also happens to be the model with the highest r-squared as well.

### Performance of Models on Test Set



### Tuning Performance

Below is a table that shows the performance of each model on the training set. The random forest metrics are below the table (as we could not figure out how to add it in the table dynamically).



Random Forest is not in the table above. Below is this model’s performance on the training set.

##   
## Call:  
## randomForest(x = train\_X, y = train\_Y, ntree = 100)   
## Type of random forest: regression  
## Number of trees: 100  
## No. of variables tried at each split: 9  
##   
## Mean of squared residuals: 0.01118717  
## % Var explained: 62.25

## 7. Model Evaluation

The student\_eval data set has been processed in parallel with student\_data so that any variables dropped or any imputation done on student\_data is also performed on student\_eval.

At this stage, we should expect for student\_eval to have no missing values except for PH. The evaluation data set has 267 observations. As expected, there are 29 predictors.

Data summary

|  |  |
| --- | --- |
| Name | student\_eval |
| Number of rows | 267 |
| Number of columns | 30 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| logical | 1 |
| numeric | 29 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: logical**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | count |
| PH | 267 | 0 | NaN | : |

**Variable type: numeric**

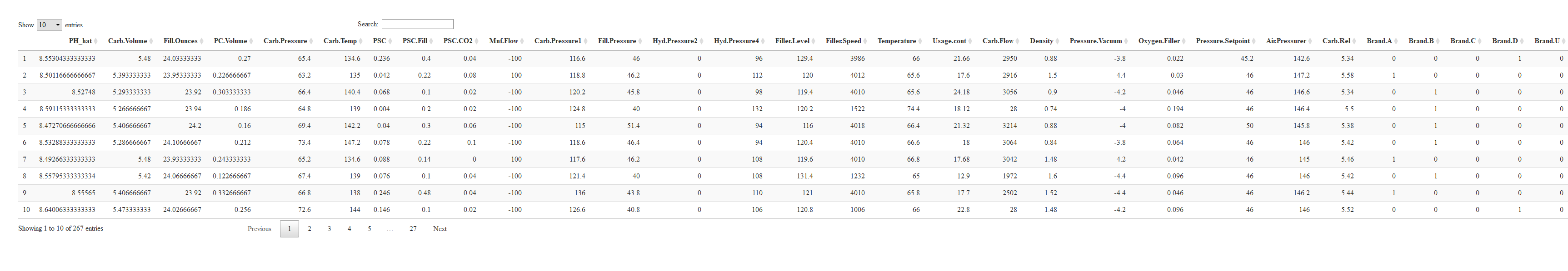
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| Carb.Volume | 0 | 1 | 5.37 | 0.11 | 5.15 | 5.28 | 5.34 | 5.46 | 5.67 | ▂▇▃▅▁ |
| Fill.Ounces | 0 | 1 | 23.97 | 0.08 | 23.75 | 23.92 | 23.97 | 24.02 | 24.20 | ▁▅▇▃▁ |
| PC.Volume | 0 | 1 | 0.28 | 0.06 | 0.10 | 0.23 | 0.28 | 0.32 | 0.46 | ▁▆▇▅▁ |
| Carb.Pressure | 0 | 1 | 68.25 | 3.86 | 60.20 | 65.30 | 68.00 | 70.60 | 77.60 | ▃▆▇▃▂ |
| Carb.Temp | 0 | 1 | 141.27 | 4.34 | 130.00 | 138.40 | 140.80 | 143.90 | 154.00 | ▁▆▇▃▁ |
| PSC | 0 | 1 | 0.09 | 0.05 | 0.00 | 0.05 | 0.08 | 0.11 | 0.25 | ▆▇▃▂▁ |
| PSC.Fill | 0 | 1 | 0.19 | 0.11 | 0.02 | 0.10 | 0.18 | 0.26 | 0.62 | ▆▇▃▁▁ |
| PSC.CO2 | 0 | 1 | 0.05 | 0.04 | 0.00 | 0.02 | 0.04 | 0.06 | 0.24 | ▇▃▂▁▁ |
| Mnf.Flow | 0 | 1 | 21.03 | 117.76 | -100.20 | -100.00 | 0.20 | 141.30 | 220.40 | ▇▁▁▆▂ |
| Carb.Pressure1 | 0 | 1 | 122.99 | 4.45 | 113.00 | 120.10 | 123.40 | 125.50 | 136.00 | ▃▃▇▂▁ |
| Fill.Pressure | 0 | 1 | 48.13 | 3.47 | 37.80 | 46.00 | 47.80 | 50.20 | 60.20 | ▁▇▇▂▁ |
| Hyd.Pressure2 | 0 | 1 | 20.04 | 17.23 | -50.00 | 0.00 | 26.80 | 34.80 | 61.40 | ▁▁▆▇▁ |
| Hyd.Pressure4 | 0 | 1 | 98.18 | 14.13 | 68.00 | 90.00 | 98.00 | 104.00 | 140.00 | ▅▆▇▂▁ |
| Filler.Level | 0 | 1 | 110.37 | 15.47 | 69.20 | 100.60 | 118.60 | 120.20 | 153.20 | ▂▃▇▇▁ |
| Filler.Speed | 0 | 1 | 3501.01 | 987.44 | 1006.00 | 3795.00 | 3918.00 | 3996.00 | 4020.00 | ▂▁▁▁▇ |
| Temperature | 0 | 1 | 66.22 | 1.69 | 63.80 | 65.40 | 65.80 | 66.60 | 75.40 | ▇▃▁▁▁ |
| Usage.cont | 0 | 1 | 20.88 | 3.00 | 12.90 | 18.10 | 21.40 | 23.74 | 24.60 | ▁▃▃▃▇ |
| Carb.Flow | 0 | 1 | 2408.64 | 1161.36 | 0.00 | 1083.00 | 3038.00 | 3215.00 | 3858.00 | ▂▃▁▆▇ |
| Density | 0 | 1 | 1.17 | 0.38 | 0.06 | 0.91 | 0.98 | 1.60 | 1.84 | ▁▁▇▁▅ |
| Pressure.Vacuum | 0 | 1 | -5.17 | 0.58 | -6.40 | -5.60 | -5.20 | -4.80 | -3.60 | ▁▇▆▃▁ |
| Oxygen.Filler | 0 | 1 | 0.05 | 0.05 | 0.00 | 0.02 | 0.03 | 0.06 | 0.40 | ▇▁▁▁▁ |
| Pressure.Setpoint | 0 | 1 | 47.73 | 2.06 | 44.00 | 46.00 | 46.00 | 50.00 | 52.00 | ▁▇▁▆▁ |
| Air.Pressurer | 0 | 1 | 142.83 | 1.23 | 141.20 | 142.20 | 142.60 | 142.80 | 147.20 | ▅▇▁▁▁ |
| Carb.Rel | 0 | 1 | 5.44 | 0.13 | 5.18 | 5.34 | 5.40 | 5.56 | 5.74 | ▂▇▂▃▂ |
| Brand.A | 0 | 1 | 0.13 | 0.34 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▁ |
| Brand.B | 0 | 1 | 0.48 | 0.50 | 0.00 | 0.00 | 0.00 | 1.00 | 1.00 | ▇▁▁▁▇ |
| Brand.C | 0 | 1 | 0.12 | 0.32 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▁ |
| Brand.D | 0 | 1 | 0.24 | 0.43 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▂ |
| Brand.U | 0 | 1 | 0.03 | 0.17 | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | ▇▁▁▁▁ |

### Use Selected Model to Predict PH on student\_eval

The selected model is the random forest model. This model is used to predict response variable PH on the student\_data.

### Predicted PH

Below is a preview of the predicted PH.



### Overview of Predicted PH

The predicted PH values do not show any values that fall outside the expected PH range of values.

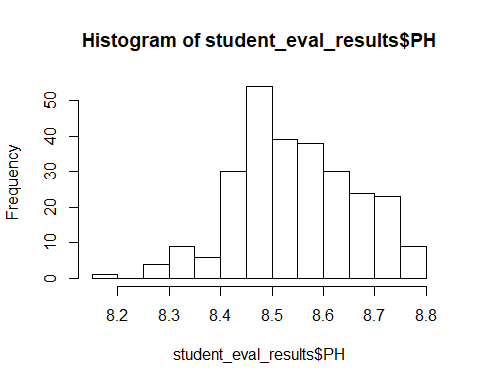
Data summary

|  |  |
| --- | --- |
| Name | student\_eval\_results$PH\_h… |
| Number of rows | 267 |
| Number of columns | 1 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Column type frequency: |  |
| numeric | 1 |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Group variables | None |

**Variable type: numeric**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| skim\_variable | n\_missing | complete\_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
| data | 0 | 1 | 8.55 | 0.11 | 8.17 | 8.47 | 8.53 | 8.63 | 8.79 | ▁▂▇▆▃ |

The general shape of the distribution of the predicted PH is approximately normal. There are only 267 rows.



## 8. Conclusion

Nine models were considered to build a predictive model for company ABC. The purpose of this model is to predict PH levels of beverages that company ABC manufactures. The selected model is a tree-based model, specifically a random forest model. This model has the lowest RMSE and also happens to have the highest r-squared as well. We were given a student evaluation file. The selected model was used to predict the PH levels for this evaluation data set. The results of the prediction show predicted values that are within the expected range. The general shahpe of the distribution of the predicted values is approximately normal as expected.

## Code Appendix

knitr::opts\_chunk$set(echo = TRUE)  
  
library(tidyverse)  
library(mice)  
library(caret)  
library(e1071)  
library(psych)  
library(DataExplorer)  
library(RANN)  
library(MASS)  
library(ggplot2)  
library(elasticnet)  
library(randomForest)  
library(skimr)  
library(DataExplorer)  
library(GGally)  
library(corrplot)  
library(DT)  
library(usdm)  
library(randomForest)  
library(vip)  
library(gbm)  
library(parallel)  
library(kernlab)  
library(doParallel)  
library(earth)  
  
student\_data<-read.csv("https://raw.githubusercontent.com/Luz917/data624project2/master/StudentData.csv",stringsAsFactors = FALSE)  
student\_eval<-read.csv("https://raw.githubusercontent.com/Luz917/data624project2/master/StudentEvaluation.csv", stringsAsFactors = FALSE)  
  
skim(student\_data)  
  
table(student\_data$ï..Brand.Code)  
  
names(student\_data)[names(student\_data) == "ï..Brand.Code"] <- "Brand.Code"  
names(student\_eval)[names(student\_eval) == "ï..Brand.Code"] <- "Brand.Code"  
  
hist(student\_data$PH)  
  
plot\_histogram((student\_data[-c(26)]))  
  
pairs(student\_data[c(26,2:11)], col="grey40")  
pairs(student\_data[c(26,12:22)], col="grey40")  
pairs(student\_data[c(26,23:25,27:33)], col="grey40")  
  
corr\_data =cor(student\_data[c(26,2:25,27:33)], use="pairwise.complete.obs", method = "pearson")  
corrplot(corr\_data, method = "color",type = "upper", order = "original", number.cex = .7,addCoef.col = "black", #Add coefficient of correlation  
 tl.srt = 90,# Text label color and rotation  
 diag = TRUE)# hide correlation coefficient on the principal diagonal  
  
(vif\_result <- vifcor(na.omit(student\_data[c(2:25,27:33)])))  
  
temp <- student\_data[c(2:25,27:33)]  
temp$Balling <- NULL  
temp$Bowl.Setpoint <- NULL  
temp$Balling.Lvl <- NULL  
temp$MFR <- NULL  
temp$Hyd.Pressure3 <- NULL  
temp$Alch.Rel <- NULL  
vifcor(temp)  
  
x <- data.frame(scale(na.omit(student\_data[c(2:33)])))  
ggplot(stack(x), aes(x= ind, y = values)) +   
 geom\_boxplot(outlier.colour="blue", outlier.shape=1, outlier.size=2, aes(fill=ind)) + theme\_minimal() + coord\_flip()   
  
plot\_histogram(student\_data[c(23, 18, 27, 30, 19, 21)])  
  
#student\_data  
student\_data[student\_data$Brand.Code=="",]$Brand.Code <- NA  
#student\_eval  
student\_eval[student\_eval$Brand.Code=="",]$Brand.Code <- NA  
plot\_missing(student\_data)  
  
result <- nearZeroVar(student\_data, saveMetrics= TRUE)  
datatable(result[order(-result$nzv),])  
  
#copy of original student\_data  
student\_data2 <- student\_data  
#copy of original student\_eval  
student\_eval2 <- student\_eval  
  
#drop for student\_data  
student\_data$Hyd.Pressure1 <- NULL  
#drop for student\_eval  
student\_eval$Hyd.Pressure1 <- NULL  
  
#drop for student\_data  
student\_data$Balling <- NULL  
student\_data$Bowl.Setpoint <- NULL  
student\_data$Balling.Lvl <- NULL  
student\_data$MFR <- NULL  
student\_data$Hyd.Pressure3 <- NULL  
student\_data$Alch.Rel <- NULL  
#drop for student\_eval  
student\_eval$Balling <- NULL  
student\_eval$Bowl.Setpoint <- NULL  
student\_eval$Balling.Lvl <- NULL  
student\_eval$MFR <- NULL  
student\_eval$Hyd.Pressure3 <- NULL  
student\_eval$Alch.Rel <- NULL  
  
#drop for student\_data  
student\_data <- student\_data[!is.na(student\_data$PH), ]  
#note: student\_eval all observations do not have PH values  
  
#student\_data  
student\_data[is.na(student\_data$Brand.Code),]$Brand.Code <- 'U'  
#student\_eval  
student\_eval[is.na(student\_eval$Brand.Code),]$Brand.Code <- 'U'  
  
boxplot(formula=PH~`Brand.Code`, data=student\_data)  
  
#student\_data  
s3obj\_mice = mice(student\_data, print = FALSE, seed = 360)  
#student\_eval  
s3obj\_mice\_eval = mice(student\_eval, print = FALSE, seed = 360)  
  
densityplot(s3obj\_mice)  
  
#student\_data  
student\_data = complete(s3obj\_mice)  
#student\_eval  
student\_eval = complete(s3obj\_mice\_eval)  
  
skim(student\_data)  
  
# student\_data  
Brand.A <- ifelse(student\_data$Brand.Code == 'A', 1, 0)  
Brand.B <- ifelse(student\_data$Brand.Code == 'B', 1, 0)  
Brand.C <- ifelse(student\_data$Brand.Code == 'C', 1, 0)  
Brand.D <- ifelse(student\_data$Brand.Code == 'D', 1, 0)  
Brand.U <- ifelse(student\_data$Brand.Code == 'U', 1, 0)  
# add dummy columns to data frame  
student\_data$Brand.A <- Brand.A  
student\_data$Brand.B <- Brand.B  
student\_data$Brand.C <- Brand.C  
student\_data$Brand.D <- Brand.D  
student\_data$Brand.U <- Brand.U  
# remove categorical variable  
student\_data <- subset(student\_data, select = -c(Brand.Code))  
#student\_eval  
Brand.A <- ifelse(student\_eval$Brand.Code == 'A', 1, 0)  
Brand.B <- ifelse(student\_eval$Brand.Code == 'B', 1, 0)  
Brand.C <- ifelse(student\_eval$Brand.Code == 'C', 1, 0)  
Brand.D <- ifelse(student\_eval$Brand.Code == 'D', 1, 0)  
Brand.U <- ifelse(student\_eval$Brand.Code == 'U', 1, 0)  
# add dummy columns to data frame  
student\_eval$Brand.A <- Brand.A  
student\_eval$Brand.B <- Brand.B  
student\_eval$Brand.C <- Brand.C  
student\_eval$Brand.D <- Brand.D  
student\_eval$Brand.U <- Brand.U  
# remove categorical variable  
student\_eval <- subset(student\_eval, select = -c(Brand.Code))  
  
skim(student\_data)  
  
# Create training and testing split from training data  
set.seed(525)  
trainrow = createDataPartition(student\_data$PH, p = 0.80, list = FALSE)  
student\_data\_train <- student\_data[trainrow, ]  
student\_data\_test <- student\_data[-trainrow, ]  
colPH <- which(colnames(student\_data) == "PH")  
train\_X <- student\_data\_train[, -colPH]  
train\_Y <- student\_data\_train$PH  
test\_X <- student\_data\_test[, -colPH]  
test\_Y <- student\_data\_test$PH  
  
#collect results of training and test results on models  
perf\_train <- data.frame(Model=character(), RMSE=double(), RSquared=double(), MAE=double())  
perf\_test <- data.frame(Model=character(), RMSE=double(), RSquared=double(), MAE=double())  
  
set.seed(1)  
PLS\_model <- train(x=train\_X,  
 y=train\_Y,   
 method='pls',  
 metric='Rsquared',  
 tuneLength=20,  
 trControl=trainControl(method='cv'),  
 preProcess=c('center', 'scale')  
 )  
PLS\_model  
  
plot(PLS\_model)  
  
PLS\_model\_pred <- predict(PLS\_model, newdata=test\_X)  
(PLS\_test <- postResample(pred=PLS\_model\_pred, obs=test\_Y))  
  
#PLS  
#save training and test perf  
r <- PLS\_model$results  
s <- r[which(r$ncomp == PLS\_model$bestTune$ncomp),]  
perf\_train[nrow(perf\_train) + 1,] = c("PLS", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("PLS", PLS\_test[1], PLS\_test[2], PLS\_test[3])  
  
## Define the candidate set of values  
ridgeGrid <- data.frame(.lambda = seq(0, 1, by=0.1))  
set.seed(1)  
ridge\_model <- train(x=train\_X,  
 y=train\_Y,  
 method = "ridge",  
 tuneGrid = ridgeGrid,  
 trControl = trainControl(method='cv') ,  
 preProc = c("center", "scale")  
 )  
ridge\_model  
  
plot(ridge\_model)  
  
ridge\_model\_pred <- predict(ridge\_model, newdata=test\_X)  
(ridge\_test <- postResample(pred=ridge\_model\_pred, obs=test\_Y))  
  
#RIDGE  
#save training and test perf  
r <- ridge\_model$results  
s <- r[which(r$lambda == ridge\_model$bestTune$lambda),]  
perf\_train[nrow(perf\_train) + 1,] = c("Ridge", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("Ridge", ridge\_test[1], ridge\_test[2], ridge\_test[3])  
  
set.seed(1)  
enet\_model <- train(x=train\_X,  
 y= train\_Y,  
 method = "enet",  
 tuneGrid=expand.grid(.fraction = seq(0, 1, by=0.1),   
 .lambda = seq(0, 1, by=0.1)),  
 trControl = trainControl(method='cv') ,  
 preProc = c("center", "scale")  
 )  
enet\_model  
  
plot(enet\_model)  
  
enet\_model\_pred <- predict(enet\_model, newdata=test\_X)  
(enet\_test <- postResample(pred=enet\_model\_pred, obs=test\_Y))  
  
#Enet  
#save training and test perf  
r <- enet\_model$results  
s <- r[which(r$lambda == enet\_model$bestTune$lambda & r$fraction == enet\_model$bestTune$fraction),]  
perf\_train[nrow(perf\_train) + 1,] = c("Enet", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("Enet", enet\_test[1], enet\_test[2], enet\_test[3])  
  
set.seed(1)  
lasso\_model <- train(x=train\_X,  
 y=train\_Y,  
 method='lasso',  
 metric='Rsquared',  
 tuneGrid=data.frame(.fraction = seq(0, 0.5, by=0.05)),  
 trControl=trainControl(method='cv'),  
 preProcess=c('center','scale')  
 )  
lasso\_model  
  
plot(lasso\_model)  
  
lasso\_model\_pred <- predict(lasso\_model, newdata=test\_X)  
(lasso\_test <- postResample(pred=lasso\_model\_pred, obs=test\_Y))  
  
#LASSo  
#save training and test perf  
r <- lasso\_model$results  
s <- r[which(r$fraction == lasso\_model$bestTune$fraction),]  
perf\_train[nrow(perf\_train) + 1,] = c("Lasso", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("Lasso", lasso\_test[1], lasso\_test[2], lasso\_test[3])  
  
set.seed(1)  
knn\_model <- train(x=train\_X,   
 y=train\_Y,   
 method="knn",   
 tuneLength=20,   
 trainControl=trainControl(method = "repeatedcv", repeats = 5),  
 preProc = c("center", "scale"))  
knn\_model  
  
plot(knn\_model)  
  
knn\_pred <- predict(knn\_model, newdata=test\_X)  
(knn\_test <- postResample(pred=knn\_pred,test\_Y))  
  
#KNN  
#save training and test perf  
r <- knn\_model$results  
s <- r[which(r$k == knn\_model$bestTune$k),]  
perf\_train[nrow(perf\_train) + 1,] = c("KNN", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("KNN", knn\_test[1], knn\_test[2], knn\_test[3])  
  
set.seed(1)  
mars\_model <- train(x=train\_X, y=train\_Y,   
 method="earth",  
 tuneLength=20,  
 preProc = c("center", "scale"))  
mars\_model  
  
plot(mars\_model)  
  
mars\_pred <- predict(mars\_model, newdata=test\_X)  
(mars\_test <- postResample(pred=mars\_pred, test\_Y))  
  
#MARS  
#save training and test perf  
r <- mars\_model$results  
s <- r[which(r$nprune == mars\_model$bestTune$nprune & r$degree == mars\_model$bestTune$degree),]  
perf\_train[nrow(perf\_train) + 1,] = c("MARS", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("MARS", mars\_test[1], mars\_test[2], mars\_test[3])  
  
set.seed(1)  
svm\_model <- train(train\_X,   
 train\_Y,   
 method="svmRadial",  
 tuneLength=10,   
 trainControl=trainControl(method = "repeatedcv", repeats = 5),  
 preProc = c("center", "scale"))  
svm\_model  
  
plot(svm\_model)  
  
svm\_pred <- predict(svm\_model, newdata=test\_X)  
(svm\_test <- postResample(pred=svm\_pred,test\_Y))  
  
#SVM  
#save training and test perf  
r <- svm\_model$results  
s <- r[which(r$sigma == svm\_model$bestTune$sigma & r$C == svm\_model$bestTune$C),]  
perf\_train[nrow(perf\_train) + 1,] = c("SVM", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("SVM", svm\_test[1], svm\_test[2], svm\_test[3])  
  
set.seed(1)  
rf\_model <- randomForest(x = train\_X, y = train\_Y, ntree = 100)  
rf\_model  
  
rf\_model\_pred <- predict(rf\_model, test\_X)  
(rf\_test <- postResample(pred = rf\_model\_pred, obs = test\_Y))  
  
#Random Forest  
#save training and test perf  
perf\_test[nrow(perf\_test) + 1,] = c("Random Forest", rf\_test[1], rf\_test[2], rf\_test[3])  
  
set.seed(1)  
gbmGrid = expand.grid(interaction.depth = seq(1,5, by=2), n.trees = seq(100, 1000, by = 100), shrinkage = 0.1, n.minobsinnode = 5)  
gbm\_model <- train(train\_X, train\_Y, tuneGrid = gbmGrid, verbose = FALSE, method = 'gbm' )  
gbm\_model  
  
gbm\_model\_pred <- predict(gbm\_model, test\_X)  
(gbm\_test <- postResample(pred = gbm\_model\_pred, obs = test\_Y))  
  
#GBM  
#save training and test perf  
r <- gbm\_model$results  
t <- gbm\_model$bestTune  
s <- r[which(r$n.trees == t$n.trees & r$interaction.depth == t$interaction.depth &   
 r$shrinkage == t$shrinkage & r$n.minobsinnode == t$n.minobsinnode),]  
perf\_train[nrow(perf\_train) + 1,] = c("GBM", s$RMSE, s$Rsquared, s$MAE)  
perf\_test[nrow(perf\_test) + 1,] = c("GBM", gbm\_test[1], gbm\_test[2], gbm\_test[3])  
  
p1 <- plot(varImp(PLS\_model), main = "PLS Model")   
p2 <- plot(varImp(enet\_model), main = "ENET Model")  
p3 <- plot(varImp(ridge\_model), main = "Ridge Model")  
p4 <- plot(varImp(lasso\_model), main = "Lasso Model")  
  
gridExtra::grid.arrange(p1,p2, p3,p4, nrow=2)  
  
p1 <- plot(varImp(knn\_model), main = "KNN Model")   
p2 <- plot(varImp(mars\_model), main = "MARS Model")  
p3 <- plot(varImp(svm\_model), main = "SVM Model")  
  
gridExtra::grid.arrange(p1,p2, p3, nrow=2)  
  
rfImp1 <- rf\_model$importance   
## p1\_trees <- vip(rf\_model) + ggtitle('Random Forest Var Imp')  
  
gbmImp1 <- gbm\_model$importance   
## p2\_trees<-vip(gbm\_model) + ggtitle('GBM Var Imp')  
  
## gridExtra::grid.arrange(p1\_trees,p2\_trees, nrow=1)  
  
datatable(perf\_test[order(perf\_test$RMSE),])  
  
datatable(perf\_train[order(perf\_train$RMSE),])  
  
rf\_model  
  
skim(student\_eval)  
  
student\_eval\_predictors <- subset(student\_eval, select = -PH)  
#Prediction is run on random forest model   
PH\_hat <- predict(rf\_model, student\_eval\_predictors)  
student\_eval\_results <- cbind(PH\_hat, student\_eval\_predictors)  
#save copy to csv  
write.csv(student\_eval\_results, "Student\_Eval\_Predictions.csv")  
  
### Predicted `PH`  
  
datatable(student\_eval\_results)  
  
skim(student\_eval\_results$PH\_hat)  
  
hist(student\_eval\_results$PH)