# DATA 624 - Predictive Analytics

Fall 2020 - Project #2

Vijaya Cherukuri, Samantha Deokinanan, Habib Khan, Priya Shaji

December 11, 2020

**Abstract**

An analysis of ABC Beverage to determine the pH level of the beverages and evaluate the accuracy of the predictive model with rigorous statistical testings.

## Contents

### Overview

Due to new regulations by abc beverage, the company leadership requires that the production team had a better understanding of the manufacturing process, the predictive factors and their relationship to the ph of the beverages. Therefore, this project was an effort to find the optimal predictive variables related to the ph of the beverages and evaluate the accuracy of the predictive model for ph of beverages with rigorous statistical testing.

the selection of a method depends on many factors - the context of the predictions, the relevance of the historical data given, the degree of accuracy, etc. As a result, these factors were cross-validated and examined for the potentially greater accuracy model at a minimal cost

### R Packages

The statistical tool that will be used to fascinate in the modeling of the data was `r`. The main packages used for data wrangling, visualization, and graphics were listed in the `code appendix`. Any other minor packages for analysis will be listed when needed

**Data Exploration**

The data set was historic data containing predictors associated with the `pH` and was provided in an excel file. We will utilize this historic data set to analyze and predict the pH of beverages.

**Predictive Variables** There were 2571 observations of 31 numeric predictor variables and 1 factor predictor variable, namely `Brand.Code`. All other variables provide measurements for the manufacturing process of each brand of beverage

**Summary Statistic** Based on the summary statistic for the beverage brands, `table 1`, we made some initial observations. The data set did not have complete cases, thus, there was a need for imputation. Some variables were highly skewed, such as `MFR`, `Temperature`, and `Oxygen.Filler`, which will need data transformation to satisfy the assumption of normality. Lastly, the `Hyd.Pressure` variables appeared to be near-zero variance predictors given that zero accounts for more than 30% of these variables. Such a zero variance predictor will never be chosen for a split since it offers no possible predictive information

**Missing Data** The graph below indicates the amount of missing data the training data contains. It appears that more than 8% of the missing data was from the `MFR` variable. This further suggests that 79% were complete. There were no missingness patterns, and their overall proportion was not extreme. This was good because for some imputation methods, such as certain types of multiple imputations, having fewer missingness patterns was helpful, as it requires fitting fewer models

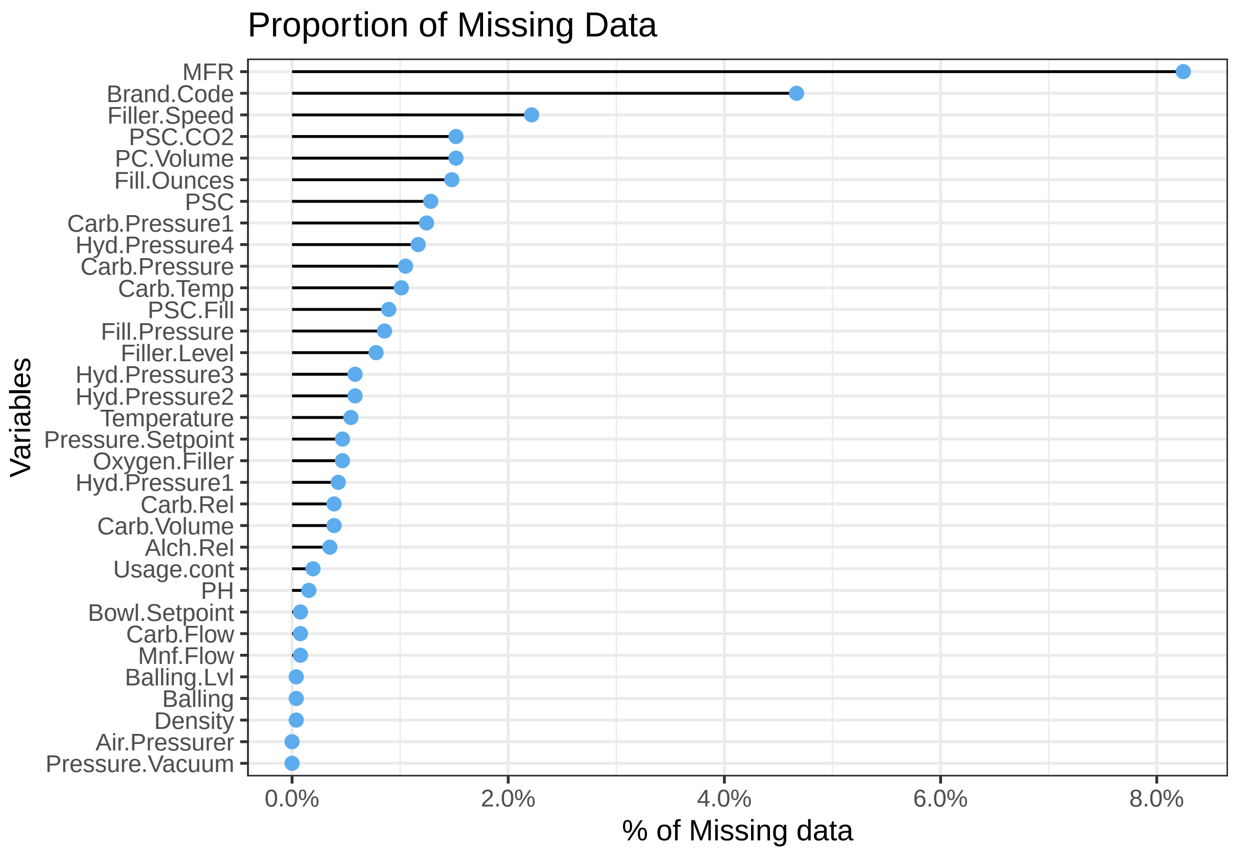


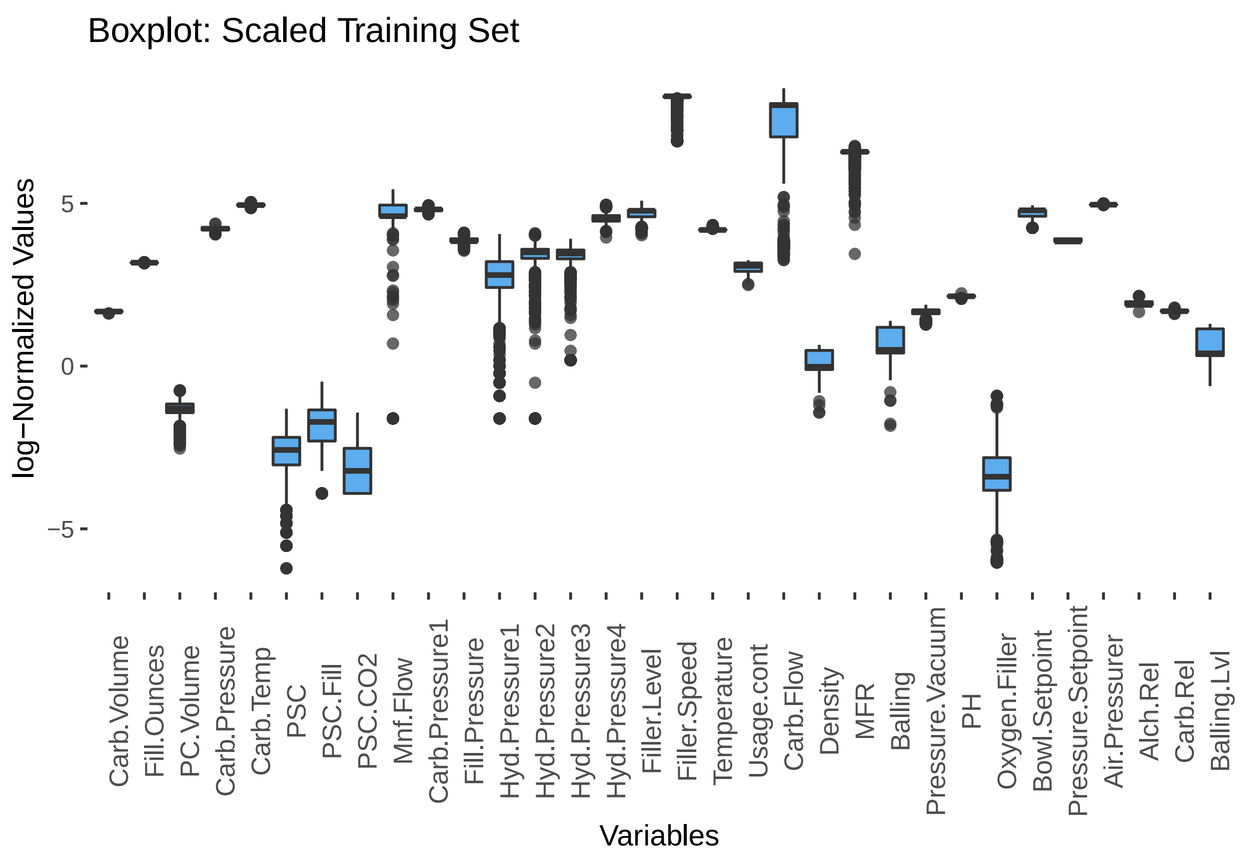
Table 1: Descriptive Statistics for All Brand Code

| | n | mean | sd | median | min | max | range | skew | kurtosis |
|---|---|---|---|---|---|---|---|---|---|
| Brand.Code* | 2451 | 2.51 | 1.00 | 2.00 | 1.00 | 4.00 | 3.00 | 0.38 | -1.06 |
| Carb.Volume | 2561 | 5.37 | 0.11 | 5.35 | 5.04 | 5.70 | 0.66 | 0.39 | -0.47 |
| Fill.Ounces | 2533 | 23.97 | 0.09 | 23.97 | 23.63 | 24.32 | 0.69 | -0.02 | 0.86 |
| PC.Volume | 2532 | 0.28 | 0.06 | 0.27 | 0.08 | 0.48 | 0.40 | 0.34 | 0.67 |
| Carb.Pressure | 2544 | 68.19 | 3.54 | 68.20 | 57.00 | 79.40 | 22.40 | 0.18 | -0.01 |
| Carb.Temp | 2545 | 141.09 | 4.04 | 140.80 | 128.60 | 154.00 | 25.40 | 0.25 | 0.24 |
| PSC | 2538 | 0.08 | 0.05 | 0.08 | 0.00 | 0.27 | 0.27 | 0.85 | 0.65 |
| PSC.Fill | 2548 | 0.20 | 0.12 | 0.18 | 0.00 | 0.62 | 0.62 | 0.93 | 0.77 |
| PSC.CO2 | 2532 | 0.06 | 0.04 | 0.04 | 0.00 | 0.24 | 0.24 | 1.73 | 3.73 |
| Mnf.Flow | 2569 | 24.57 | 119.48 | 65.20 | -100.20 | 229.40 | 329.60 | 0.00 | -1.87 |
| Carb.Pressure1 | 2539 | 122.59 | 4.74 | 123.20 | 105.60 | 140.20 | 34.60 | 0.05 | 0.14 |
| Fill.Pressure | 2549 | 47.92 | 3.18 | 46.40 | 34.60 | 60.40 | 25.80 | 0.55 | 1.41 |
| Hyd.Pressure1 | 2560 | 12.44 | 12.43 | 11.40 | -0.80 | 58.00 | 58.80 | 0.78 | -0.14 |
| Hyd.Pressure2 | 2556 | 20.96 | 16.39 | 28.60 | 0.00 | 59.40 | 59.40 | -0.30 | -1.56 |
| Hyd.Pressure3 | 2556 | 20.46 | 15.98 | 27.60 | -1.20 | 50.00 | 51.20 | -0.32 | -1.57 |
| Hyd.Pressure4 | 2541 | 96.29 | 13.12 | 96.00 | 52.00 | 142.00 | 90.00 | 0.55 | 0.63 |
| Filler.Level | 2551 | 109.25 | 15.70 | 118.40 | 55.80 | 161.20 | 105.40 | -0.85 | 0.05 |
| Filler.Speed | 2514 | 3687.20 | 770.82 | 3982.00 | 998.00 | 4030.00 | 3032.00 | -2.87 | 6.71 |
| Temperature | 2557 | 65.97 | 1.38 | 65.60 | 63.60 | 76.20 | 12.60 | 2.39 | 10.16 |
| Usage.cont | 2566 | 20.99 | 2.98 | 21.79 | 12.08 | 25.90 | 13.82 | -0.54 | -1.02 |
| Carb.Flow | 2569 | 2468.35 | 1073.70 | 3028.00 | 26.00 | 5104.00 | 5078.00 | -0.99 | -0.58 |
| Density | 2570 | 1.17 | 0.38 | 0.98 | 0.24 | 1.92 | 1.68 | 0.53 | -1.20 |
| MFR | 2359 | 704.05 | 73.90 | 724.00 | 31.40 | 868.60 | 837.20 | -5.09 | 30.46 |
| Balling | 2570 | 2.20 | 0.93 | 1.65 | -0.17 | 4.01 | 4.18 | 0.59 | -1.39 |
| Pressure.Vacuum | 2571 | -5.22 | 0.57 | -5.40 | -6.60 | -3.60 | 3.00 | 0.53 | -0.03 |
| PH | 2567 | 8.55 | 0.17 | 8.54 | 7.88 | 9.36 | 1.48 | -0.29 | 0.06 |
| Oxygen.Filler | 2559 | 0.05 | 0.05 | 0.03 | 0.00 | 0.40 | 0.40 | 2.66 | 11.09 |
| Bowl.Setpoint | 2569 | 109.33 | 15.30 | 120.00 | 70.00 | 140.00 | 70.00 | -0.97 | -0.06 |
| Pressure.Setpoint | 2559 | 47.62 | 2.04 | 46.00 | 44.00 | 52.00 | 8.00 | 0.20 | -1.60 |
| Air.Pressurer | 2571 | 142.83 | 1.21 | 142.60 | 140.80 | 148.20 | 7.40 | 2.25 | 4.73 |
| Alch.Rel | 2562 | 6.90 | 0.51 | 6.56 | 5.28 | 8.62 | 3.34 | 0.88 | -0.85 |
| Carb.Rel | 2561 | 5.44 | 0.13 | 5.40 | 4.96 | 6.06 | 1.10 | 0.50 | -0.29 |
| Balling.Lvl | 2570 | 2.05 | 0.87 | 1.48 | 0.00 | 3.66 | 3.66 | 0.59 | -1.49 |

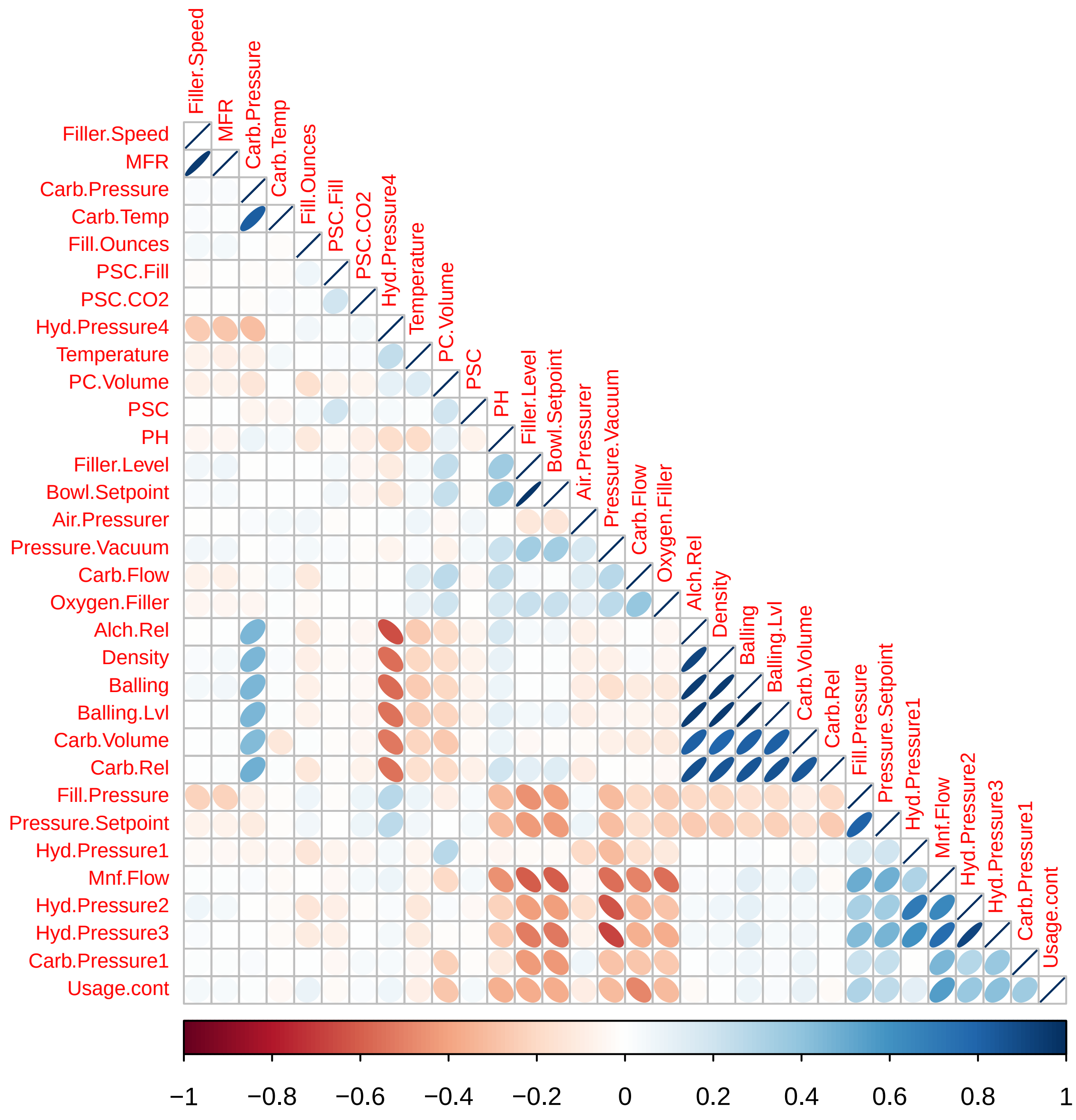
**Outlier** Further exploration revealed that some variables may be strongly influenced by outliers. An outlier is an observation that lies an abnormal distance from other values in a random sample. Outliers in the data could distort predictions and affect the accuracy, therefore, these were corrected by imputation.

Table 2: Predictive Variables with Outlier

| variables | outlier |
|---|---|
| Carb.Volume | 5 |
| Fill.Ounces | 55 |
| PC.Volume | 85 |
| Carb.Pressure | 14 |
| Carb.Temp | 36 |
| PSC | 52 |
| PSC.Fill | 70 |
| PSC.CO2 | 77 |
| Carb.Pressure1 | 24 |
| Fill.Pressure | 79 |
| Hyd.Pressure1 | 7 |
| Hyd.Pressure4 | 83 |
| Filler.Level | 9 |
| Filler.Speed | 439 |
| Temperature | 123 |
| MFR | 238 |
| Pressure.Vacuum | 125 |
| PH | 18 |
| Oxygen.Filler | 189 |
| Air.Pressurer | 220 |
| Alch.Rel | 6 |
| Carb.Rel | 7 |



**Correlation** The corrgram below graphically represents the correlations between the numeric predictor variables, when ignoring the missing variables. Most of the numeric variables were uncorrelated with one another, but there were a few highly correlated pairs.

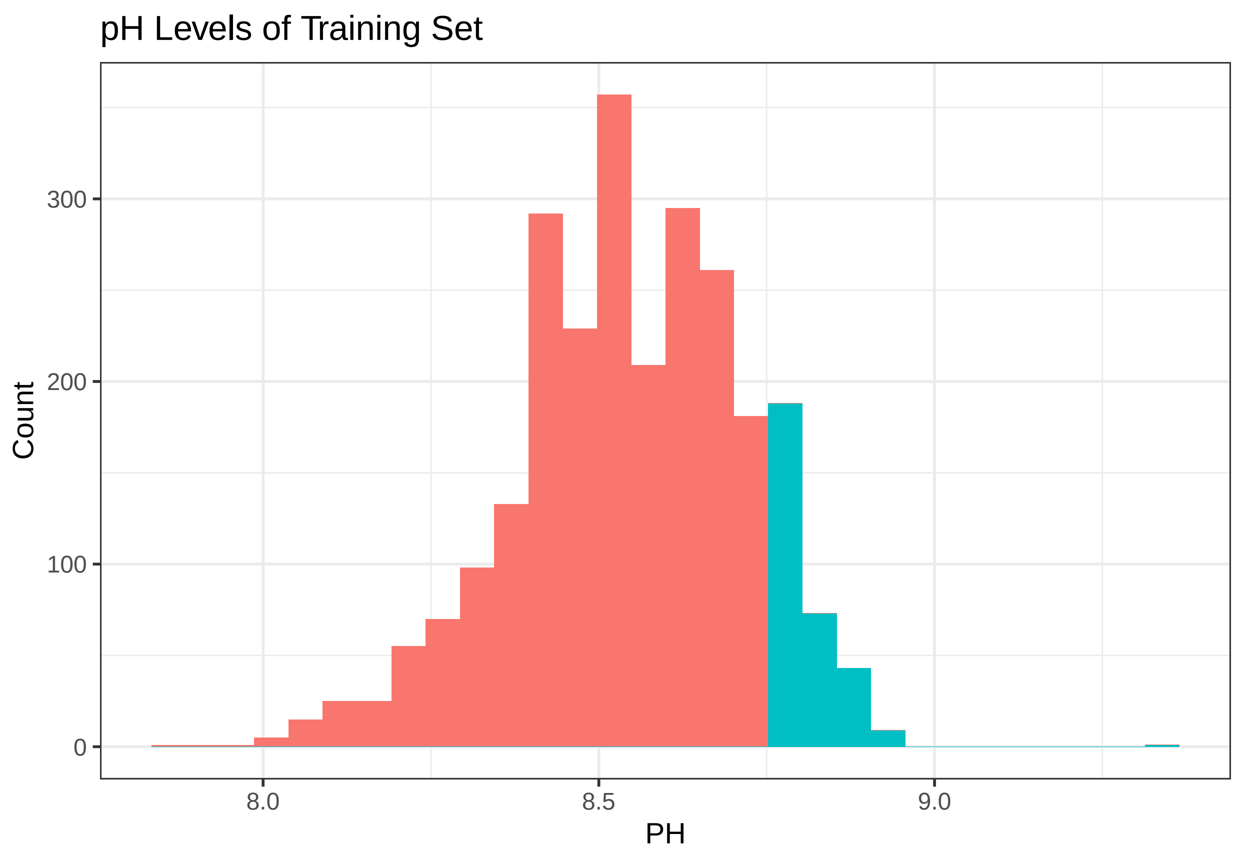


Moreover, to build a smaller model without predictors with extremely high correlations, it was best to reduce the number of predictors such that there were no absolute pairwise correlations above 0.90. The list below shows only significant correlations (at 5% level) for the top 10 highest correlations by the correlation coefficient. The results show that these ten had a correlation of greater than 0.95.

Table 3: Top 10 Highly Correlated Predictor Candidates

| | X1 | X2 | value |
|---|---|---|---|
| 736 | Balling.Lvl | Balling | 0.9876727 |
| 1015 | Balling | Balling.Lvl | 0.9876727 |
| 507 | Bowl.Setpoint | Filler.Level | 0.9773811 |
| 848 | Filler.Level | Bowl.Setpoint | 0.9773811 |
| 672 | Balling.Lvl | Density | 0.9550900 |
| 1013 | Density | Balling.Lvl | 0.9550900 |
| 663 | Balling | Density | 0.9523125 |
| 725 | Density | Balling | 0.9523125 |
| 534 | MFR | Filler.Speed | 0.9514224 |
| 689 | Filler.Speed | MFR | 0.9514224 |

**Target Variable (pH)** The response variable, `pH` had a couple of missing values. There was also a detection of outliers, which explained the skewness in the variable. The plot below highlights that a majority of the pH levels was less than 8.75. This suggested that the beverages are alkaline based.



**Data Preparation**

**Pre-Processing of Predictors** Firstly, we treated missing data and outlier by imputing them. The random forest (RF) missing data algorithms was implemented because this could handle mixed types of missing data, adaptable to interactions and non-linearity, and they had the potential to scale to big data settings. This would help to account for the uncertainty in the individual imputations. In addition, near-zero variable diagnostics were performed on the predictors to identify if they had very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value was large. If true, these were removed.

**Creating Dummy Variables** `Brand.Code` was the only categorical variable within this data set. It represents the brand code of the beverages with values A, B, C, and D. For the purpose of modeling, it was converted into a set of dummy variables.

**Correlation** Next, to filter out highly correlated predictors, were removed those that have an absolute correlation coefficient greater than 0.90. This resulted in 25 of the 32 predictor variables are kept.

**Normality** The data was then pre-processed to fulfill the assumption of normality using the Yeo-Johnson transformation (Yeo and Johnson, 2000). This technique attempts to find the value of lambda that minimizes the Kullback-Leibler distance between the normal distribution and the transformed distribution. This method had the advantage of working without having to worry about the domain of x.

**Training & Testing Split** All the models were trained on the same approximately 70% of the training set, reserving 30% for validation of which model to select for the pH estimation on the supplied evaluation set.

## Building the Models

The relationship between these manufacturing processes with pH is not known. Therefore, we investigated various regression and classification families.

**Model #1: Baseline Model** We started with a simple linear model to serve as a baseline. This included all variables in the training data set.

We immediately saw that a few variables exceeded the 0.05 p-value threshold for significance. The intercept itself suggested that with no information on the manufacturing processes, a beverage is likely to have a pH balance of a killer 369! This baseline model also only account for 40.4% of the variability of the data. Therefore, there was a need for a better model. We focused and selected our model based on how well it explains that data and the accuracy of the results.

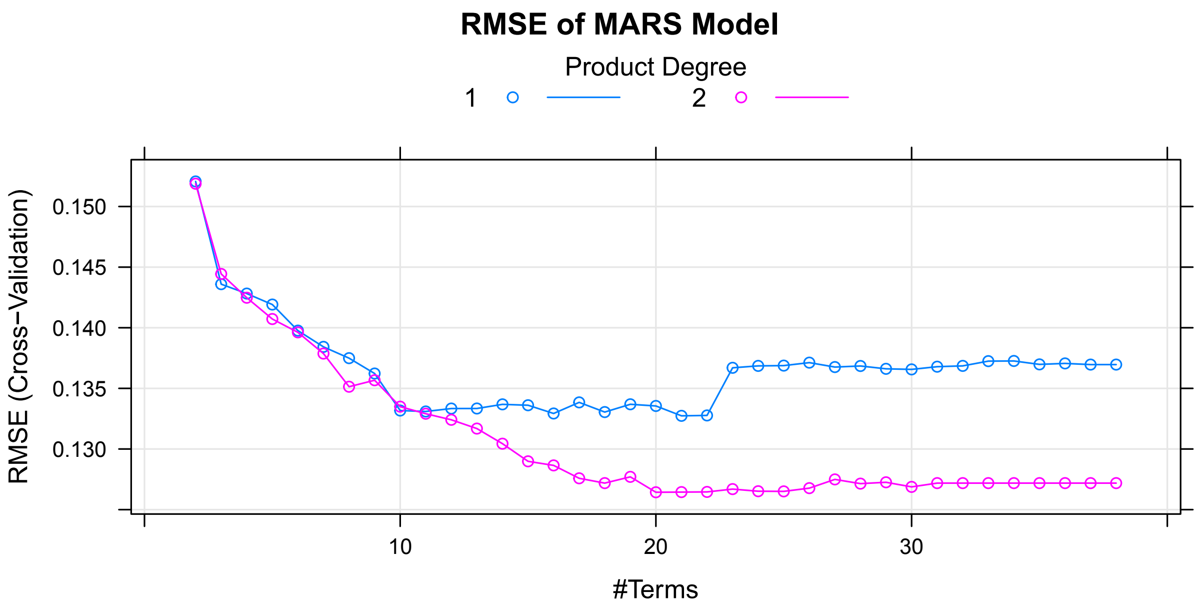
Table 4: Model 1 - Baseline Linear Regression Output

| | Estimate | Std. Error | t value | Pr(>\|t\|) |
|---|---:|---:|---:|---:|
| (Intercept) | 368.587 | 6220.216 | 0.059 | 0.953 |
| Brand.Code.A | -0.082 | 0.018 | -4.646 | 0.000 |
| Brand.Code.B | -0.002 | 0.027 | -0.089 | 0.929 |
| Brand.Code.C | -0.138 | 0.028 | -4.922 | 0.000 |
| Carb.Volume | -0.112 | 0.090 | -1.241 | 0.215 |
| Fill.Ounces | 0.000 | 0.000 | -1.477 | 0.140 |
| PC.Volume | -0.033 | 0.099 | -0.336 | 0.737 |
| Carb.Pressure | 0.324 | 0.968 | 0.335 | 0.738 |
| Carb.Temp | -756.771 | 12852.974 | -0.059 | 0.953 |
| PSC | -0.115 | 0.069 | -1.659 | 0.097 |
| PSC.Fill | -0.074 | 0.060 | -1.247 | 0.213 |
| PSC.CO2 | -0.027 | 0.075 | -0.355 | 0.723 |
| Mnf.Flow | -0.001 | 0.000 | -12.051 | 0.000 |
| Carb.Pressure1 | 0.020 | 0.002 | 8.391 | 0.000 |
| Fill.Pressure | 4.793 | 1.244 | 3.851 | 0.000 |
| Hyd.Pressure2 | 0.006 | 0.001 | 4.649 | 0.000 |
| Hyd.Pressure4 | 0.037 | 0.120 | 0.305 | 0.761 |
| Filler.Speed | 0.000 | 0.000 | -0.516 | 0.606 |
| Temperature | -0.014 | 0.003 | -5.359 | 0.000 |
| Usage.cont | -0.007 | 0.001 | -5.395 | 0.000 |
| Carb.Flow | 0.000 | 0.000 | 2.416 | 0.016 |
| MFR | 0.000 | 0.000 | -0.424 | 0.672 |
| Pressure.Vacuum | 0.000 | 0.000 | 0.543 | 0.587 |
| Oxygen.Filler | -0.272 | 0.079 | -3.434 | 0.001 |
| Bowl.Setpoint | 0.002 | 0.000 | 6.696 | 0.000 |
| Pressure.Setpoint | -0.008 | 0.002 | -3.453 | 0.001 |
| Air.Pressurer | 0.000 | 0.003 | -0.009 | 0.993 |
| Alch.Rel | 0.003 | 0.022 | 0.131 | 0.896 |
| Carb.Rel | 0.135 | 0.054 | 2.475 | 0.013 |

**Model #2: Multivariate Adaptive Regression Splines** MARS is an algorithm that essentially creates a piecewise linear model which provides an intuitive stepping block into non-linearity after grasping the concept of linear regression and other intrinsically linear models. Two tuning parameters associated with the MARS model was done to identify the optimal combination of these hyperparameters that minimize prediction error.

Table 5: Model 2: MARS Model Coefficient

| | y |
|---|---|
| (Intercept) | 8.40 |
| h(Mnf.Flow-0.199347) | 0.00 |
| h(0.199347-Mnf.Flow) | 0.00 |
| h(0.199347-Mnf.Flow)*h(Pressure.Vacuum–78.7763) | 0.00 |
| h(0.199347-Mnf.Flow)*h(-78.7763-Pressure.Vacuum) | 0.00 |
| h(Alch.Rel-7.04) | 0.13 |
| h(Mnf.Flow-0.199347)*h(145.4-Air.Pressurer) | 0.00 |
| h(Bowl.Setpoint-90)*h(7.04-Alch.Rel) | 0.01 |
| h(0.199347-Mnf.Flow)*h(Air.Pressurer-143.8) | 0.00 |
| Brand.Code.C*h(Hyd.Pressure2-0.190033) | -0.01 |
| Brand.Code.C*h(0.190033-Hyd.Pressure2) | -1.21 |
| Brand.Code.C*h(65.4-Temperature) | -0.34 |
| h(68.2-Temperature) | 0.03 |
| h(Mnf.Flow-0.199347)*h(Usage.cont-20.28) | 0.00 |
| h(Mnf.Flow-0.199347)*h(20.28-Usage.cont) | 0.00 |
| h(Carb.Pressure1-56.5916) | -0.10 |
| h(56.5916-Carb.Pressure1) | -0.01 |
| h(Mnf.Flow-0.199347)*h(Filler.Speed-3888) | 0.00 |
| h(PC.Volume-0.145941)*h(Carb.Pressure1-56.5916) | 1.55 |
| h(Mnf.Flow-0.199347)*h(Filler.Speed-3990) | 0.00 |

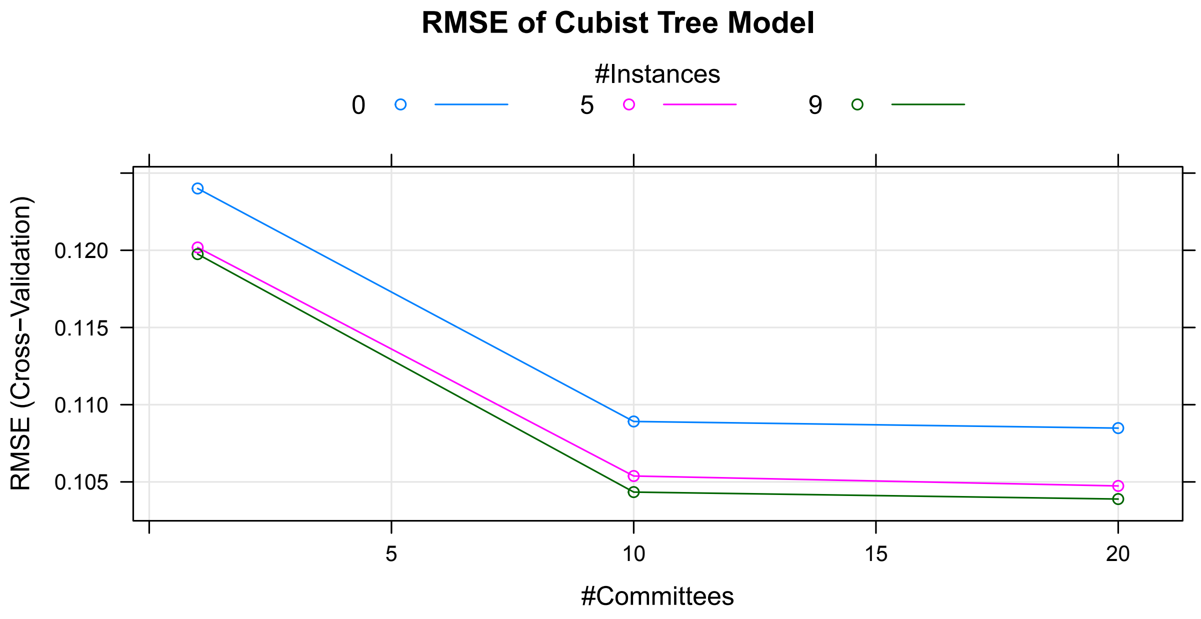


RMSE was used to select the optimal MARS model using the smallest value. The best tune for the MARS model which resulted in the smallest root mean squared error was with 2 degrees of interactions and the number of retained terms of 20. It had RMSE = 0.126, and $R^2$ = 0.468. In this case, it did account for the largest portion of the variability in the data than all other variables, and it produced the smallest error.

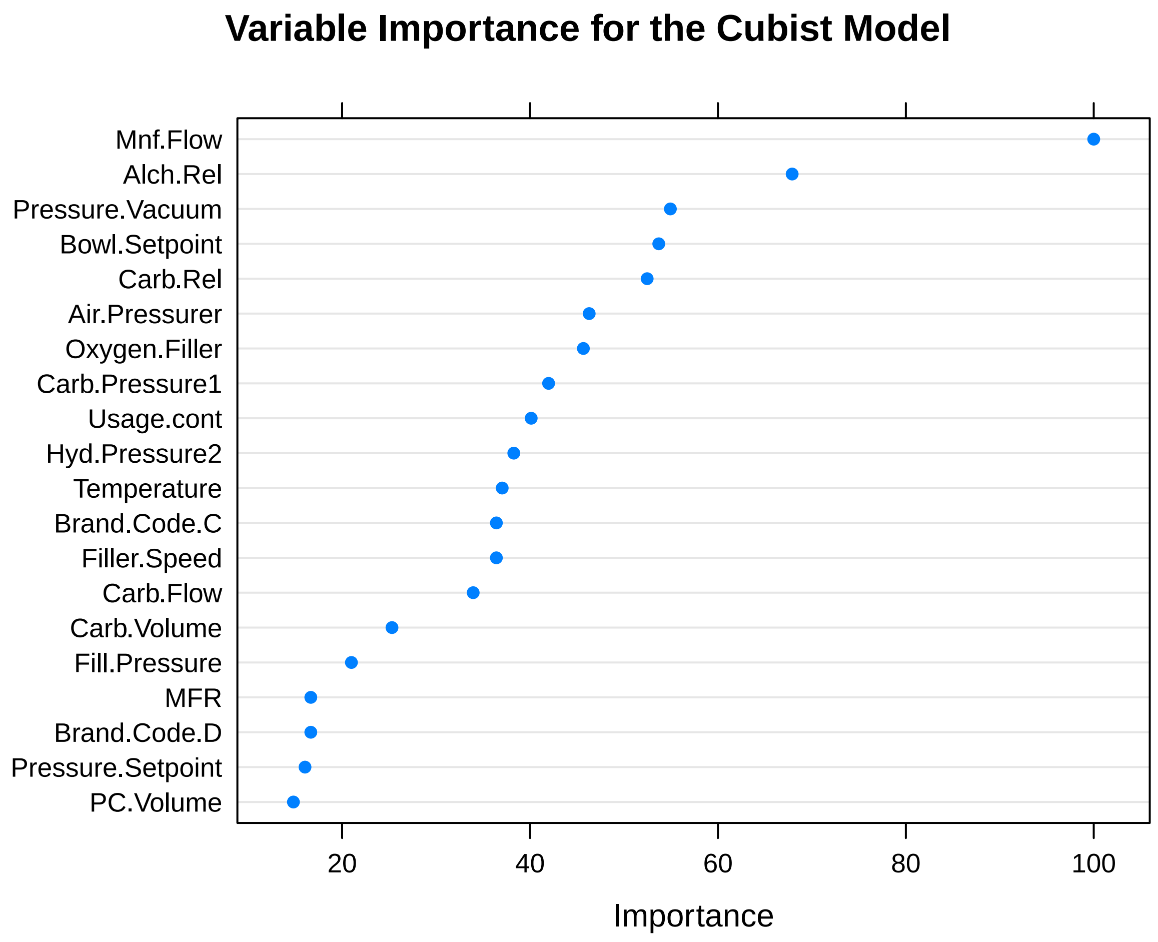
Having no other information, the MARS model expects a beverage pH level of about 8.40. This meant that the model identified factors that lean more to decreasing the pH than to increase it. It was also less than the expected value with a difference of 0.15.

Contributing coefficients are those which lie within the 95% level of significance. A positive coefficient indicates that as the value of the predictor increases, the response variable also tends to increase. A negative coefficient suggests that as the predictor increases, the response variable tends to decrease. In addition to pruning the number of knots, the potential interactions between different hinge functions are illustrated. There were quite a few interaction terms between multiple hinge functions.

**Model #3: Cubist** This is a prediction-oriented regression model that initially creates a tree structure, and then collapses each path through the tree into a rule. A regression model is fit for each rule based on the data subset defined by the rules. The collection of rules are pruned or combined, and the candidate variables for the models are the predictors that were pruned away.

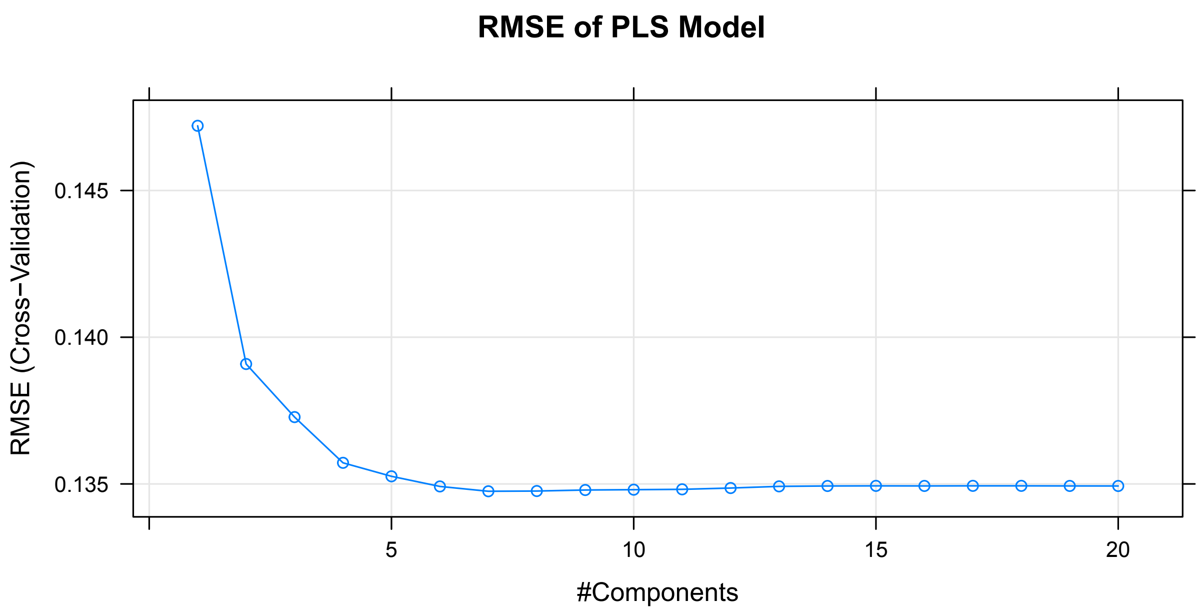


RMSE was used to select the optimal model using the smallest value. The best tune for the cubist model which resulted in the smallest root mean squared error was with 20 committees and correct the prediction using the 9-nearest neighbors. It had RMSE = 0.104, and $R^2$ = 0.635. In this case, it did account for the largest portion of the variability in the data than all other variables, and it produced the smallest error which makes it the best fit.



The Cubist output provided the percentage of times where each variable was used in the condition or the linear model. At each split of the tree, Cubist saved a linear model after feature selection. The variable importance used here was a linear combination of the usage in the rule conditions and the model. From the plot, the most important variable was `Mnf.Flow` followed by `Alch.Rel` and `Pressure.Vacuum`.

**Model #4: Partial Least Squares** Partial Least Squares is a technique which minimizes the predictors to a smaller set of uncorrelated components and performs least squares regression on these components instead of on the original data. It is useful when the predictors have high correlation. Another advantage of PLS is that it does not assume that the predictors are fixed and predictors are measured with errors. It makes the model more robust. It was ideal to use PLS in this situation as there were many predictors and not enough sample size. Also, there were high correlations among some of the predictors. In this model, we trained the data and later we evaluated the performance on the test set to see how the model was performing based on RMSE and r-squared values. The data was also centered and scaled through preProcess function.



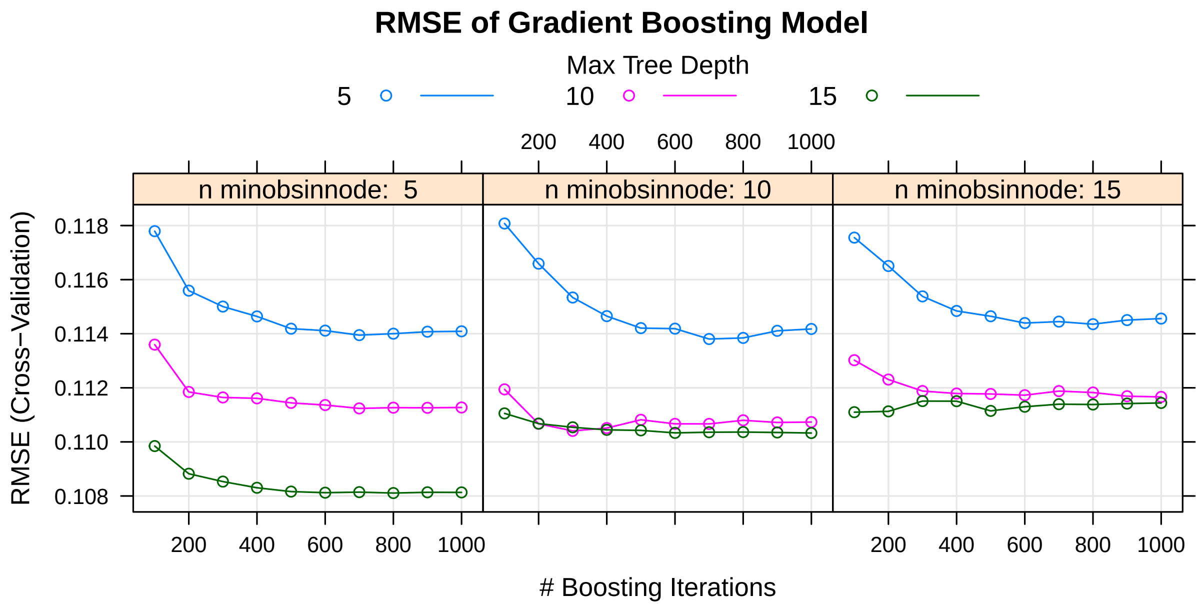
**Model #5: Gradient Boosting** Gradient boosting is one of the most powerful techniques for building predictive models. Gradient boosting involves three elements:

1. A loss function to be optimized.

2. A weak learner to make predictions.

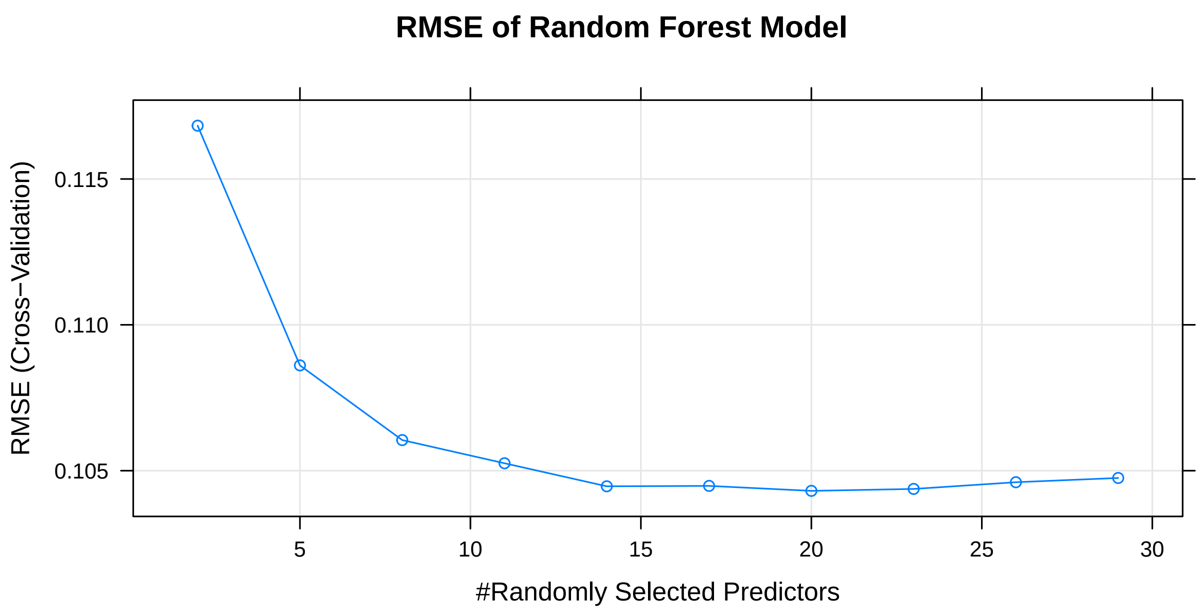
3. An additive model to add weak learners to minimize the loss function.

A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function but at the same time it is a greedy algorithm and it can over-fit a training data set quickly. It is quite fascinating for accuracy and speed especially with large and complex data and one of the most used techniques among the data scientists that is why we used it and evaluated how well it performed in in our situation.



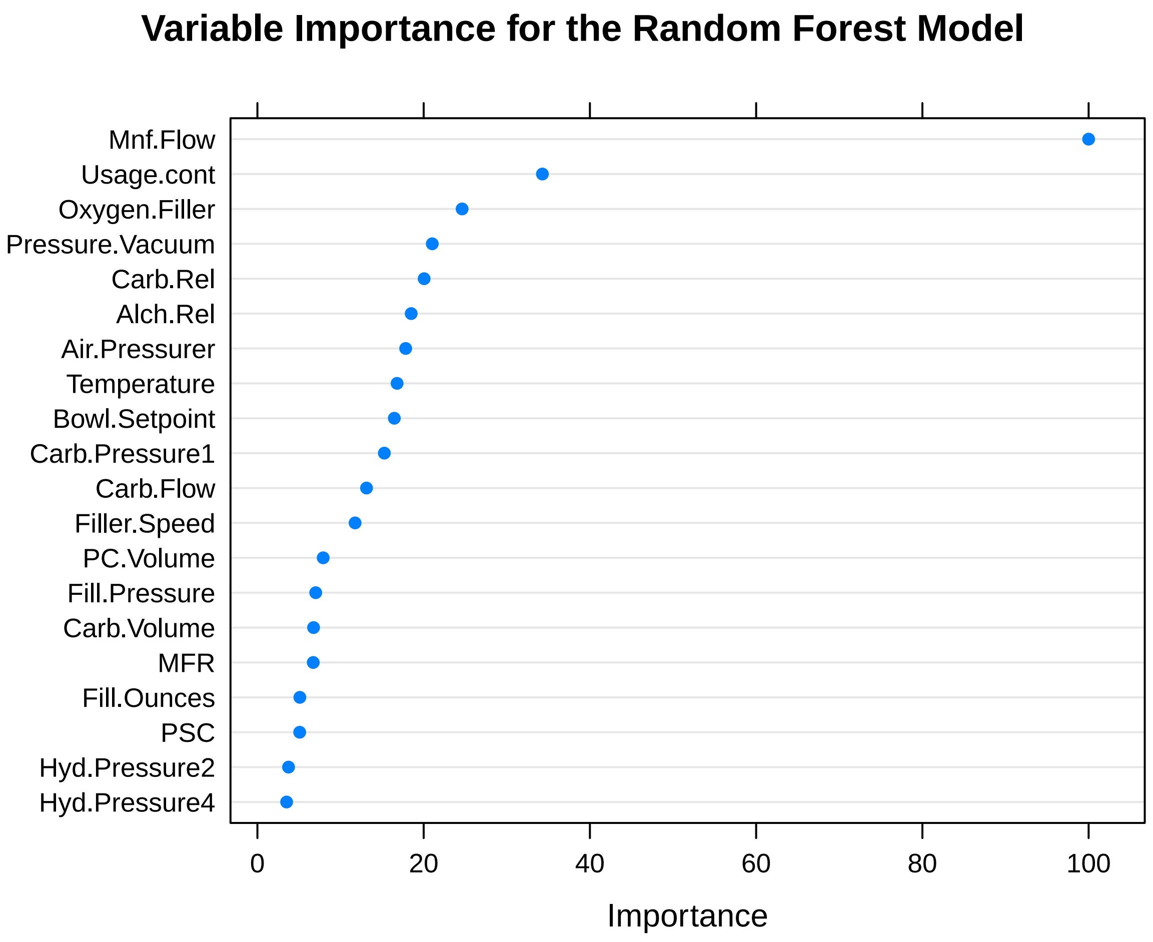
Tuning parameter 'shrinkage' was held constant at a value of 0.1. RMSE was used to select the optimal model using the smallest value. The final values used for the model were n.trees = 800, interaction.depth = 15, and the minimum number of observations in trees' terminal nodes is 5.

**Model #6: Random Forest** Random forest is a machine learning algorithm that contains a forest with number of trees. These trees are called decision trees therefore it consists of random collection of forest trees. It can be used for both classification and regression easily. The best part of using random forest is that it provides higher accuracy through cross validation. It handles the missing values and maintains the accuracy of a large proportion of data. It would not allow over-fitting if there are more trees in the model and it has the power to handle any data set no matter if it is small or large with any dimensionality. It seemed like a wonderful algorithm to use in our situation. We used random forest regression and then compared its performance with the other models to select the best model out of them. It is highly used in medicines, chemistry, stock market, banking sector and e-commerce.



The best tune for the random forest model which resulted in the smallest root mean squared error was with the optimal number of randomly selected predictors to choose from at each split being 20. It had RMSE = 0.104, and $R^2 = 0.65$.

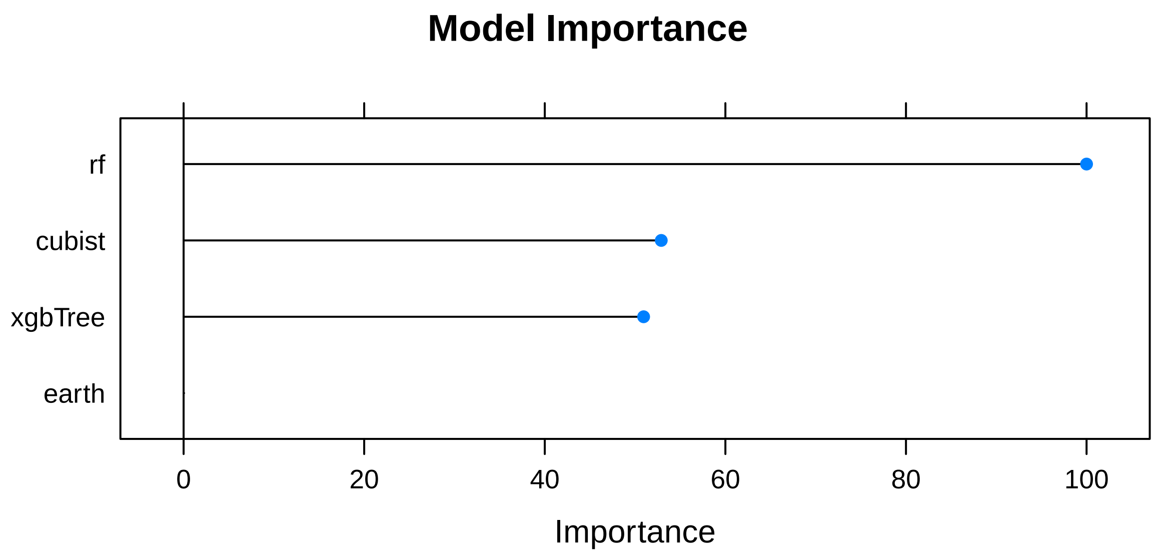
Let us display informative variables picked by Random Forest model using `varImp`. Random Forest model picked the most informative variables among which top three variables are `Mnf.Flow`, `Usage.cont`, `Oxygen.Filter` and `Pressure.Vacuum`. These are the variables which were most related to the pH of beverages.



Therefore, `Mnf.Flow` is the most important variable followed by `Brand.Code` , affecting pH value of beverages.

**Model #7: Ensemble Regression** The last model built was an ensemble regression. The goal of ensemble regression was to combine several models to improve the prediction accuracy in learning problems with a numerical target variable. There were three phases of processing ensemble learning. These include the generation phase, the pruning phase, and the integration phase. Given a list of caret models, we built a function that can be used to specify a higher-order model to learn how to best combine the predictions of sub-models. The 4 sub-models were the best four models created thus far based on their RMSEs. More specifically, these were:

- Multivariate Adaptive Regression Splines
- Cubist
- Gradient Boosting
- Random Forest



From the plot, it is clear that the random forest model plays a major role in the prediction model, while MARS did not at all. This seems rational as it was the model with the smallest RMSE and largest $R^2$.

### Model Evaluation

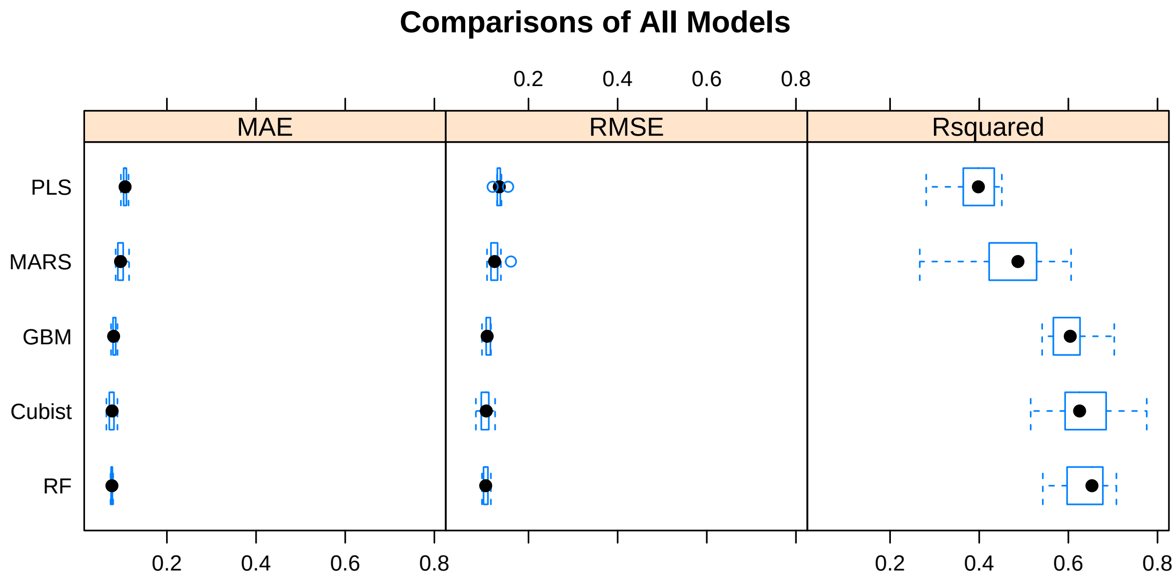
**Model Selection Criteria** To select the best model to make predictions, we looked at the following goodness-of-fit metrics:

1. $R^2$, which represents the proportion of the variance explained by the model
2. *Root Mean Squared Error* (RMSE), which is the square root of the mean squared difference between the observation and the fitted value.
3. *Mean Absolute Error* (MAE), which is the average of all absolute errors. Absolute Error is the amount of error in the measurements. It is the difference between the measured value and 'true' value.

We looked at $R^2$ and RMSE as our primary metrics and selected the model which performed best on both. If no model performs best on both, the MAE of the model results on the training set were used to break the tie. The remaining 30% of the historical data (i.e. predictors `test.p` and response `test.r`) was used to determine the model performance.

Table 6: Performance Metric for All Models

| | RMSE | Rsquared | MAE |
|---|---|---|---|
| MARS | 0.126 | 0.476 | 0.096 |
| CUBIST | 0.103 | 0.649 | 0.075 |
| PLS | 0.139 | 0.367 | 0.108 |
| GBM | 0.109 | 0.612 | 0.080 |
| RF | 0.102 | 0.665 | 0.072 |
| Ensemble | 0.100 | 0.674 | 0.071 |



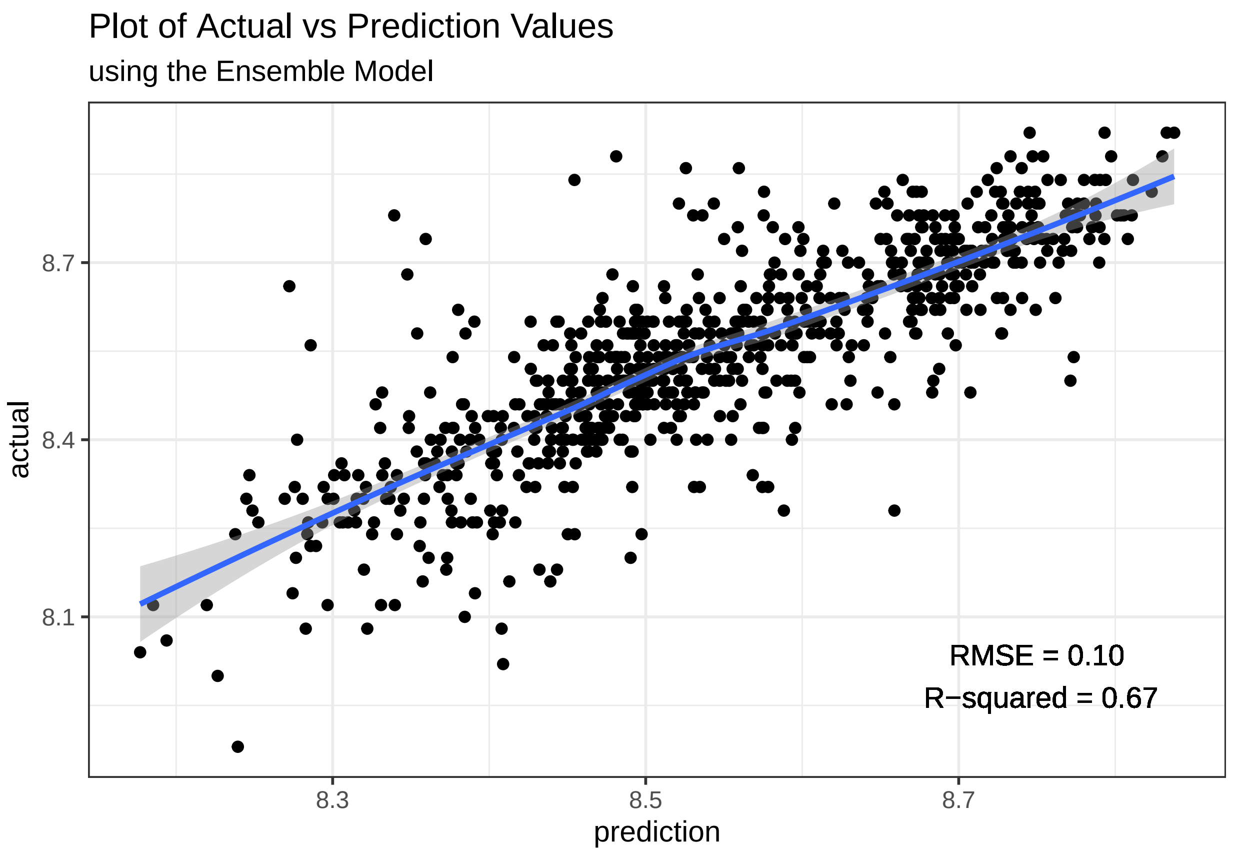
After fitting a multi-linear, MARS, cubist, partial least squared, GBM, and random forest model to the training data, it was evident that a random forest model is well-suited for modeling the data. This model outperformed the other linear and tree-based models in every resampling performance metric. The RMSE and $R^2$ resampling performance metrics of the Random Forest model was superior to all other model.

Adding the `ensemble` model to this list, the table highlights the performance criteria for all the models built. For exceptional prediction quality, a meta- algorithm such as the `ensemble` method proves to be beneficial since it combine several learning techniques into one predictive model in order to decrease variance, bias, and improve predictions. This meta-model outperformed the other linear, and tree-based models in every resampling performance metric, and it was superior to the `Random Forest`, the best single model.

Typically, for predicting physical processes, $R^2$ should be greater than 50%, as this only explains about 29% of the standard deviation. The model with both the smallest errors and account for the largest proportion of the data variability was the `ensemble` model, with RMSE = 0.100 and $R^2 = 0.674$. From the descriptive statistic, it was clear that the predictions are similar to the actual values. The predictions are consistent throughout the range of pH.

Table 7: Comparison of the Test Set Actual and Predicted Values

| | n | mean | sd | median | min | max | range | skew | kurtosis | se |
|---|---|---|---|---|---|---|---|---|---|---|
| test_actual | 769 | 8.54 | 0.17 | 8.54 | 7.88 | 8.92 | 1.04 | -0.35 | -0.06 | 0.01 |
| test_prediction | 769 | 8.54 | 0.14 | 8.53 | 8.18 | 8.84 | 0.66 | -0.02 | -0.70 | 0.00 |

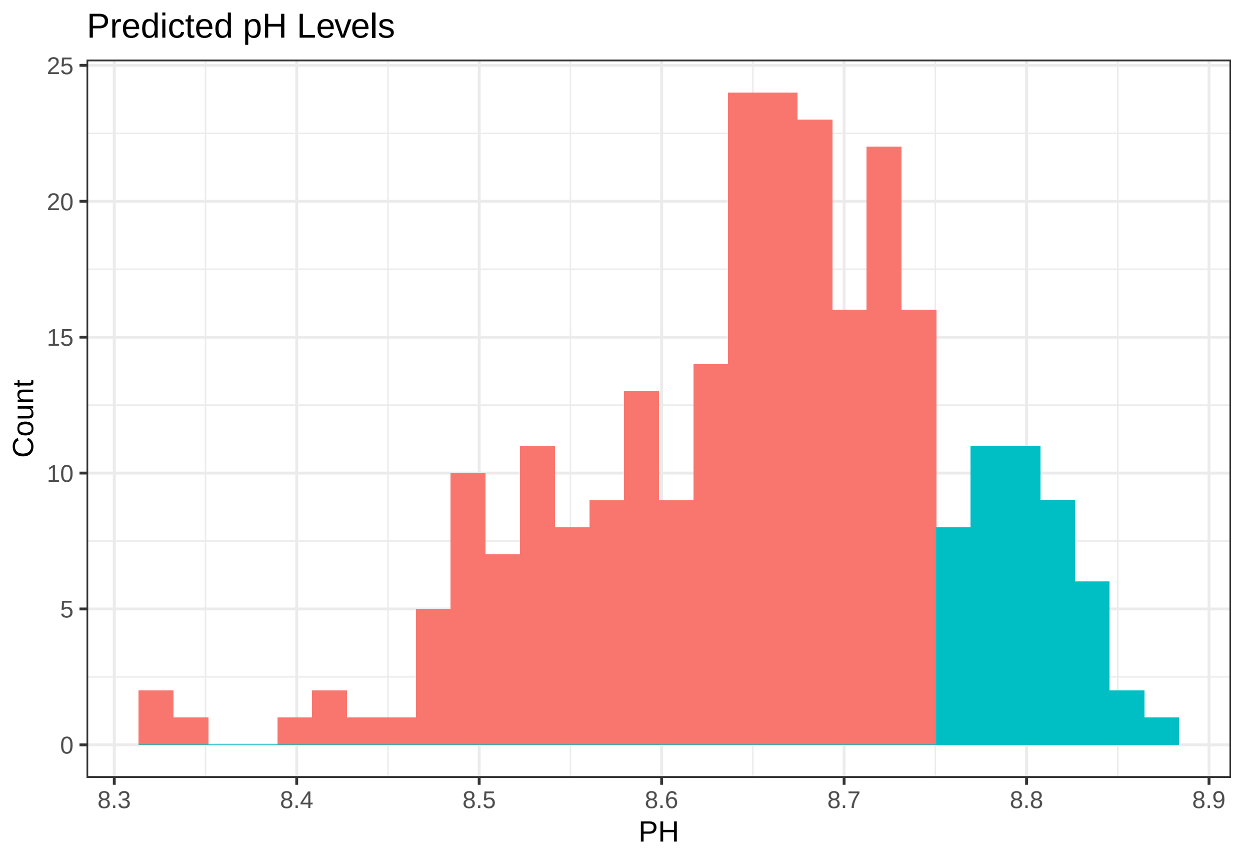


Because the models are weighted such that predictions are maximized, consistent quality is expected. If a model predicts better at lower pH, the `ensemble` model will be up-weighted over that specific range. Therefore, according to this data set and performance metrics, we confidently say that `Model #7 - Ensemble Model` was the best model based on all criteria that were used to evaluate its fit, and was used to predict the pH levels of the evaluation set.

**The Optimal Model** With the optimal model decided, we re-trained the model on the entire training set before we made the predictions on the evaluation data.

### Prediction Result

Based on the performance test, it was decided that the `ensemble` model was the optimal model. With the evaluation set pre-processed identically to that of the training set, it was used to make the pH prediction of beverages.



**Save pH Predictions** The pH predictions are save to our group's GitHub repository.

**Conclusion**

ABC Beverage company is a beverage manufacturer that most likely produces alkaline beverages. Our team was given historical data on its manufacturing processes for some of these beverages, where we were tasked to determine the pH level. The data was prepared for missing values, which is a crucial step for data analysis. After missing values were dealt with, the data needed to be normalize. Seven models were created, these included a base model, MARS, Cubist, Partial Least Square, Gradient Boosting, Random Forest and Ensemble models. The data was split into training and test data sets, 70% and 30% respectively. We noticed that all the models have overall almost close results for RMSE, R-squared and MAE. Random Forest found to be a best model among the rest. PLS performed worst among the others but still the values were not significantly.

Knowing the performance of these models, an Ensemble model was built as a mean to improve the prediction accuracy by learning the problems with the target variable and combining several models. In the end, the main processes that can likely be the cause for a great shift in the pH levels are `Mnf.Flow`, `Usage.cont`, `Oxygen.Filler`, and `Pressure.Vacuum` when altered. Therefore, readjustment to the desired balance can be achieve by carefully monitoring these processes. Moreover, future manufacturing should carefully consider these processes as a priority during evaluation of the pH levels for the currently produced beverages. These features can also save on operational cost for the trial and error phase during the creation of new beverages.

## Works Cited

1. Max Kuhn and Kjell Johnson. Applied Predictive Modeling. Springer, New York, 2013.
2. Yeo, I., & Johnson, R. (2000). A New Family of Power Transformations to Improve Normality or Symmetry. Biometrika, 87(4), 954–959. Retrieved November 26, 2020, from http://www.jstor.org/stable/2673623

## Code Appendix

```r
# Required R packages
library(tidyverse)
library(kableExtra)
library(psych)
library(caret)
library(mice)
library(corrplot)
library(earth)
library(xgboost)
library(Cubist)
library(caretEnsemble)
library(vip)
```

```r
# Set master seed
set.seed(52508)

# Set filepaths for data ingestion
urlRemote  = "https://github.com/"
pathGithub = "greeneyefirefly/DATA624-Project2/blob/main/"
fileTrain = "StudentData.xlsx"
fileTest = "StudentEvaluation.xlsx"

# Read training file
tempfile_1 = tempfile(fileext = ".xlsx")
tempfile_2 = tempfile(fileext = ".xlsx")

# Load training dataset
download.file(url = paste0(urlRemote, pathGithub, fileTrain, "?raw=true"),
              destfile = tempfile_1,
              mode = "wb",
              quiet = TRUE)
train_df = data.frame(readxl::read_excel(tempfile_1,skip=0))

# Load test dataset
download.file(url =  paste0(urlRemote, pathGithub, fileTest, "?raw=true"),
              destfile = tempfile_2,
              mode = "wb",
              quiet = TRUE)
eval_df = data.frame(readxl::read_excel(tempfile_2,skip=0))

# Number of training observations
ntrobs = dim(train_df)[[1]]
```

```r
# Transform Brand.Code to factor
train_df$Brand.Code = as.factor(train_df$Brand.Code)
eval_df$Brand.Code = as.factor(eval_df$Brand.Code)
```

```r
kable(describe(train_df)[,-c(1,6,7,13)],
      caption = "Descriptive Statistics for All Brand Code",
      digit = 2L)
```

```r
na.counts = as.data.frame(((sapply(train_df,
                                   function(x) sum(is.na(x))))/nrow(train_df))*100)
names(na.counts) = "counts"
na.counts = cbind(variables = rownames(na.counts),
                  data.frame(na.counts, row.names = NULL))

na.counts %>% arrange(counts) %>% mutate(name = factor(variables, levels = variables)) %>%
  ggplot(aes(x = name, y = counts)) + geom_segment( aes(xend = name, yend = 0)) +
  geom_point(size = 2, color = "steelblue2") + coord_flip() + theme_bw() +
  labs(title = "Proportion of Missing Data", x = "Variables", y = "% of Missing data") +
  scale_y_continuous(labels = scales::percent_format(scale = 1))
```

```r
temp = data.frame(variables = NA, outlier = NA)
for (i in 2:33){
  temp = rbind(temp, c(names(train_df)[i],
                             length(boxplot(train_df[i], plot = FALSE)$out)))
}
remove = apply(temp, 1, function(row) all(row !=0 ))
temp = na.omit(temp[remove,])
row.names(temp) = NULL
kable(temp, caption = "Predictive Variables with Outlier")

ggplot(data = reshape2::melt(train_df) , aes(x = variable, y = log(abs(value)))) +
geom_boxplot(fill = 'steelblue2', outlier.alpha = 0.75) +
  labs(title = 'Boxplot: Scaled Training Set',
       x = 'Variables',
       y = 'log-Normalized Values') +
  theme(panel.background = element_rect(fill = 'white'),
        axis.text.x = element_text(size = 10, angle = 90))
```

```r
corrplot::corrplot(cor(train_df[,-1], use = 'complete.obs'),
         method = 'ellipse', type = 'lower', order = 'hclust',
         hclust.method = 'ward.D2', tl.cex = 0.7)
```

```r
corr = cor(train_df[,-1], use = 'complete.obs')
corr[corr == 1] = NA
corr[abs(corr) < 0.90] = NA
corr = na.omit(reshape::melt(corr))
kable(head(corr[order(-abs(corr$value)),], 10),
      caption = "Top 10 Highly Correlated Predictor Candidates")
```

```r
train_df %>%
  ggplot(aes(PH, fill = PH > 8.75)) +
  geom_histogram(bins = 30) +
  theme_bw() +
  theme(legend.position = 'center') +
  labs(y = 'Count', title = 'pH Levels of Training Set')
```

```r
set.seed(525)
# Train set
processed_train_df = mice(train_df, method = 'rf', print = FALSE, m = 3, maxit = 3)
train_df_cleaned = complete(processed_train_df)

predictors = nearZeroVar(train_df_cleaned)
train_df_cleaned = train_df_cleaned[,-predictors]

# Evaluation set
processed_eval_df = mice(eval_df, method = 'rf', print = FALSE, m = 3, maxit = 3)
eval_df_cleaned = complete(processed_eval_df)
```

```r
set.seed(525)
# Train set
dummy.brand.code = dummyVars(PH ~ Brand.Code, data = train_df_cleaned)
train_dummy = predict(dummy.brand.code, train_df_cleaned)
train_df_cleaned = cbind(train_dummy, train_df_cleaned) %>% select(-Brand.Code)

# Evaluation set
eval_df_cleaned$PH = 1
dummy.brand.code = dummyVars(PH ~ Brand.Code, data = eval_df_cleaned)
eval_dummy = predict(dummy.brand.code, eval_df_cleaned)
eval_df_cleaned = cbind(eval_dummy, eval_df_cleaned) %>% select(-Brand.Code, -PH)
```

```r
# Train set
tooHigh = findCorrelation(cor(train_df_cleaned), 0.90)
train_df_cleaned = train_df_cleaned[, -tooHigh]
```

```r
set.seed(525)
# Train set
processed_train_df = preProcess(train_df_cleaned, method = c("YeoJohnson"))
train_df_cleaned =  predict(processed_train_df, train_df_cleaned)

# Evaluation set
processed_eval_df = preProcess(eval_df_cleaned, method = c("YeoJohnson"))
eval_df_cleaned =  predict(processed_eval_df, eval_df_cleaned)
```

```r
# Create training and testing split from training data
set.seed(525)
intrain = createDataPartition(train_df_cleaned$PH, p = 0.70, list = FALSE)

# Train & Test predictor variables
train.p = train_df_cleaned[intrain, ] %>% select(-PH)
test.p = train_df_cleaned[-intrain, ] %>% select(-PH)
```

```r
# Train & Test response variable (pH)
train.r = train_df_cleaned$PH[intrain]
test.r = train_df_cleaned$PH[-intrain]
```

```r
# Baseline linear model
set.seed(525)
baseline = lm(train.r ~ ., data = train.p)
kable(summary(baseline)$coefficients, digits = 3L,
      caption = 'Model 1 - Baseline Linear Regression Output')
```

```r
set.seed(525)
marsGrid = expand.grid(.degree = 1:2,
                       .nprune = 2:38)
marsModel = train(x = train.p,
                  y = train.r,
                  method = "earth",
                  tuneGrid = marsGrid,
                  trControl = trainControl(method = "cv",
                                           number = 10))
```

```r
plot(marsModel, main = "RMSE of MARS Model")
```

```r
kable(summary(marsModel$finalModel)$coefficients,
      digits = 2L,
      caption = "Model 2: MARS Model Coefficient")
```

```r
set.seed(525)
cubModel = train(x = train.p,
                 y = train.r,
                 method = "cubist",
                 tuneLength = 10,
                 trControl = trainControl(method = "cv",
                                          repeats = 5))
```

```r
plot(cubModel, main = "RMSE of Cubist Tree Model")
```

```r
dotPlot(varImp(cubModel), main = 'Variable Importance for the Cubist Model')
```

```r
set.seed(393)
plsModel = train(x = train.p,
                 y = train.r,
                 method = "pls",
                 tuneLength = 20,
                 trControl= trainControl(method="cv"),
                 preProcess=c('center','scale'))
```

```r
plot(plsModel, main = "RMSE of PLS Model")
```

```r
set.seed(20350)
grid <- expand.grid(n.trees=seq(100, 1000, by = 100),
                    interaction.depth=c(5, 10, 15),
                    shrinkage=0.1,
                    n.minobsinnode=c(5, 10, 15))
gbm_m <- train(x = train.p,
               y = train.r,
               method = 'gbm',
               tuneGrid = grid,
               trControl = trainControl(method = "cv",
                                        repeats = 5),
               verbose = FALSE)
```

```r
plot(gbm_m, main = "RMSE of Gradient Boosting Model")
```

```r
set.seed(9988)
rfModel <- train(x = train.p,
                 y = train.r,
                 method = "rf",
                 tuneLength = 10,
                 trControl = trainControl(method = "cv",
                                          repeats = 5))
```

```r
plot(rfModel, main = "RMSE of Random Forest Model")
```

```r
dotPlot(rfModel, main = 'Variable Importance for the Random Forest
Model')
```

```r
set.seed(525)
# Model Tuning Grids
marsGrid = expand.grid(.degree = 2, .nprune = 20)
cubGrid = expand.grid(.committees = 20, .neighbors = 9)
rfGrid = expand.grid(mtry = 20)
xgbGrid = expand.grid(eta = 0.01, nrounds = 1000, max_depth = 6,
                      gamma = 0, colsample_bytree = 0.8,
                      min_child_weight = 0.8, subsample = 0.8)
# List of Algorithms to use in Ensemble
tuning_list = list(caretModelSpec(method = "earth", tuneGrid = marsGrid),
                   caretModelSpec(method = "cubist", tuneGrid = cubGrid),
                   caretModelSpec(method = "rf", tuneGrid = rfGrid,
                                  importance = TRUE),
                   caretModelSpec(method = "xgbTree", tuneGrid = xgbGrid))
# Adding the train controls
control = trainControl(method = 'cv',
                       number = 5,
                       savePredictions = 'final',
                       index = createFolds(train.r, 5),
                       allowParallel = TRUE)
model_list = caretList(x = train.p,
                       y = train.r,
                       trControl = control,
                        tuneList = tuning_list)
# Combine several predictive models via stacking
ensembleModel = caretStack(model_list,
                           method = "glmnet",
                           metric = "RMSE",
                           trControl=trainControl(method = "cv",
                                                  number = 5,
                                                  savePredictions = "final"))
```

```r
plot(varImp(ensembleModel$ens_model), main = "Model Importance")
```

```r
fits = list(MARS = marsModel,
            Cubist = cubModel,
            PLS = plsModel,
            GBM = gbm_m,
            RF = rfModel)
bwplot(resamples(fits), main = "Comparisons of All Models")
```

```r
set.seed(525)
marsPred = predict(marsModel, newdata = test.p)
cubPred = predict(cubModel , newdata = test.p)
plsPred = predict(plsModel, newdata = test.p)
gradPred = predict(gbm_m , newdata = test.p)
rfPred = predict(rfModel, newdata = test.p)
ensemblePred = predict(ensembleModel, newdata = test.p)

compTable = data.frame(rbind(MARS = postResample(pred = marsPred, obs = test.r),
                             CUBIST = postResample(pred = cubPred, obs = test.r),
                             PLS = postResample(pred = plsPred, obs = test.r),
                             GBM = postResample(pred = gradPred, obs = test.r),
                             RF = postResample(pred = rfPred, obs = test.r),
                             Ensemble = postResample(pred = ensemblePred, obs = test.r)))

kable(compTable, digits = 3L, caption = "Performance Metric for All Models")
```

```r
rbind(test_actual = describe(test.r)[,-c(1,6,7)],
      test_prediction = describe(predict(ensembleModel,
                                         newdata = test.p))[,-c(1,6,7)]) %>%
  kable(digits = 2L,
        caption = "Comparison of the Test Set Actual and Predicted Values")
```

```r
ggplot(data.frame(prediction = predict(ensembleModel, newdata = test.p),
                  actual = test.r)) +
  geom_point(aes(x = prediction, y = actual)) +
  geom_smooth(aes(x = prediction, y = actual)) +
  labs(title = "Plot of Actual vs Prediction Values",
       subtitle = "using the Ensemble Model") +
  geom_text(x = 8.75, y = 8, label = "RMSE = 0.10\n R-squared = 0.67") +
  theme_bw()
```

```r
set.seed(525)
# Full, clean training data
Xtrain = train_df_cleaned %>% select(-PH)
yTrain = train_df_cleaned$PH

# Model Tuning Grids
marsGrid = expand.grid(.degree = 2, .nprune = 27)
cubGrid = expand.grid(.committees = 20, .neighbors = 9)
rfGrid = expand.grid(mtry = 20)
xgbGrid = expand.grid(eta = 0.01, nrounds = 1000, max_depth = 6,
                      gamma = 0, colsample_bytree = 0.8,
                      min_child_weight = 0.8, subsample = 0.8)
# List of Algorithms to use in Ensemble
tuning_list = list(caretModelSpec(method = "earth", tuneGrid = marsGrid),
                   caretModelSpec(method = "cubist", tuneGrid = cubGrid),
                   caretModelSpec(method = "rf", tuneGrid = rfGrid,
                                  importance = TRUE),
                   caretModelSpec(method = "xgbTree", tuneGrid = xgbGrid))
# Control parameters for train functions
control = trainControl(method = 'cv',
                       number = 5,
                       savePredictions = 'final',
                       index = createFolds(yTrain, 5),
                       allowParallel = TRUE)
model_list = caretList(x = Xtrain,
                       y = yTrain,
                       trControl = control,
                       tuneList = tuning_list)
# Combine several predictive models via stacking
ensembleModel_final = caretStack(model_list,
                                 method = "glmnet",
                                 metric = "RMSE",
                                 trControl=trainControl(method = "cv",
                                                        number = 5,
                                                        savePredictions = "final"))
```

```r
eval_df_cleaned$PH = predict(ensembleModel_final, newdata = eval_df_cleaned)

eval_df_cleaned %>%
  ggplot(aes(PH, fill = PH > 8.75)) +
  geom_histogram(bins = 30) +
  theme_bw() +
  theme(legend.position = 'center') +
  labs(y = 'Count', title = 'Predicted pH Levels')
```

```r
# write.csv(eval_df_cleaned$PH, "StudentEvaluation_Predictions.csv")
```