Data 624 Project 2

CUNY SPS MSDS
Dr. Scott Burk
Summary 2022
Group 4:

Melvin Matanos Claire Meyer Chinedu Onyeka
Euclid Zhang Jie Zou

7/3/2022

Table of Contents

Executive Summary

The objective of this analysis is to utilize data model to predict the PH of beverage products and identify key factors in the manufacturing process that can be used to improve the PH of the products. The data contains manufacturing variables such as temperature, hydraulic pressures and filler speed. Upon data exploration, we found most of the variable have missing values. We used the MICE method (a set of chained regression models) to impute the missing values. Two extreme outliers that seem to be unreasonable were removed. Variables that are highly correlated were removed to reduce time building models and reduce model bias. Additionally, we applied Yeo Johnson transformation, centering and scaling to the predictors to improve model performance and unbiasness. We then splt the data into a training data set and a small test data set. The training data was used to create 10 models of different types and the model parameters were tuned based on the RMSE performance measure. The 10-fold cross-validation with 10 repeats resampling was used in the tuning process to mitigate the effect of overfiting. The test data set was used to evaluate the models' performance on unseen data. The Random Forest model and Cubist model have the best and comparable performance. We rebuilt the two models using the whole data and found that the Cubist model has the best performance without changing the model parameters. We select the Cubist model as our optimal model and perform prediction. Addtionally, we identified the most important factors in the manufacturing process from the models. The key focus is to increase **Mnf Flow** and lower **Pressure Vacuum** and lower **Bowl Setpoint** to lower the **PH** of the beverage products.

Project Description

The production of beverage must conform several regulations issued by the Food and Drug Administration (FDA). The potential for hydrogen (pH) is a measure of acidity/alkalinity, it must conform in a critical range to be considered safe to the customers. Hence, it is important to understand the key factors in manufacturing process that have significant effect to the PH of the products. In this analysis, we are provided a data set of 2571 cases with 32 independent variables. We would use the data to create models to predict the PH of the produced beverages and identify the most important factors to help improving the manufacturing process.

Data Exploration

Model Summary

The following is a summary of all predictor variables. It shows the number of missing values, the percentage of data missing, the mean and standard deviation, the median, and the ranges of the variables.

The **Brand Code** is a categorical variable with values **A**, **B**, **C** and **D**. The other variables are numeric.

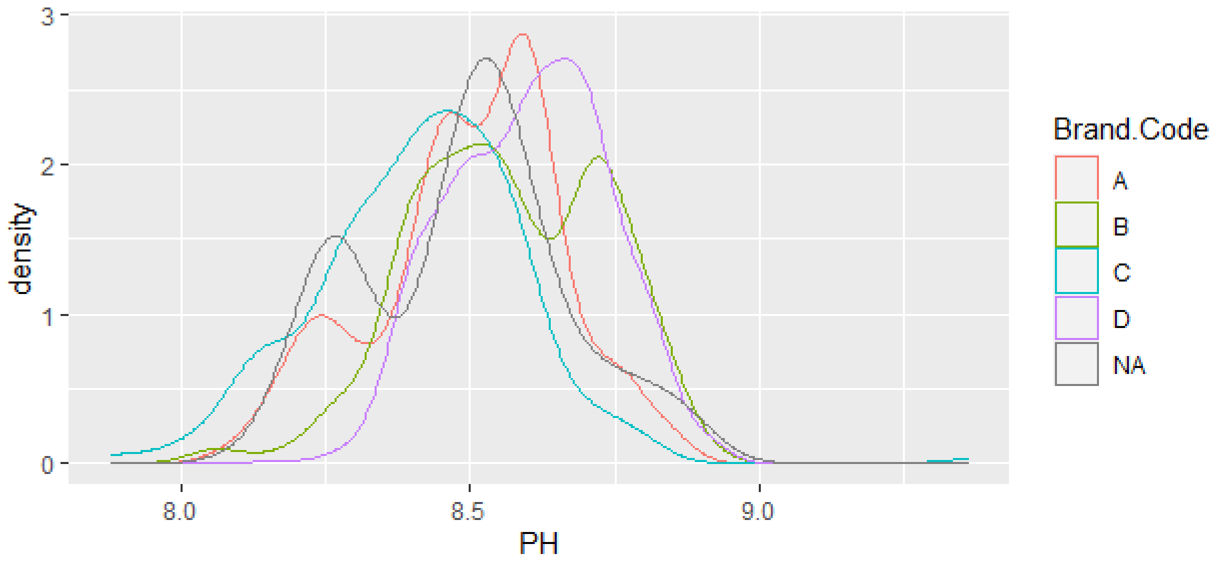
var	n_NA	pct_NA	mean	sd	median	min	max
Brand.Code*	120	4.67%	0.00	0.00	0.00	0.00	0.00
Carb.Volume	10	0.39%	5.37	0.11	5.35	5.04	5.70
Fill.Ounces	38	1.48%	23.97	0.09	23.97	23.63	24.32
PC.Volume	39	1.52%	0.28	0.06	0.27	0.08	0.48
Carb.Pressure	27	1.05%	68.19	3.54	68.20	57.00	79.40
Carb.Temp	26	1.01%	141.09	4.03	140.80	128.60	154.00
PSC	33	1.29%	0.08	0.05	0.08	0.00	0.27
PSC.Fill	23	0.9%	0.20	0.12	0.18	0.00	0.62
PSC.CO2	39	1.52%	0.06	0.04	0.04	0.00	0.24
Mnf.Flow	0	0%	24.63	119.50	70.20	-100.20	229.40
Carb.Pressure1	32	1.25%	122.57	4.73	123.20	105.60	140.20
Fill.Pressure	18	0.7%	47.92	3.18	46.40	34.60	60.40
Hyd.Pressure1	11	0.43%	12.46	12.43	11.40	-0.80	58.00
Hyd.Pressure2	15	0.58%	20.99	16.38	28.60	0.00	59.40
Hyd.Pressure3	15	0.58%	20.48	15.97	27.60	-1.20	50.00
Hyd.Pressure4	28	1.09%	96.31	13.10	96.00	62.00	142.00
Filler.Level	16	0.62%	109.25	15.70	118.40	55.80	161.20
Filler.Speed	54	2.1%	3,688.11	769.63	3,982.00	998.00	4,030.00
Temperature	12	0.47%	65.96	1.38	65.60	63.60	76.20
Usage.cont	5	0.19%	20.99	2.98	21.79	12.08	25.90
Carb.Flow	2	0.08%	2,472.05	1,070.43	3,030.00	26.00	5,104.00
Density	0	0%	1.17	0.38	0.98	0.24	1.92
MFR	208	8.1%	704.05	73.90	724.00	31.40	868.60
Balling	0	0%	2.20	0.93	1.65	0.16	4.01
Pressure.Vacuum	0	0%	-5.22	0.57	-5.40	-6.60	-3.60
PH	0	0%	8.55	0.17	8.54	7.88	9.36
Oxygen.Filler	11	0.43%	0.05	0.05	0.03	0.00	0.40
Bowl.Setpoint	2	0.08%	109.35	15.29	120.00	70.00	140.00
Pressure.Setpoint	12	0.47%	47.61	2.04	46.00	44.00	52.00

var	n_NA	pct_NA	mean	sd	median	min	max
Air.Pressurer	0	0%	142.83	1.21	142.60	140.80	148.20
Alch.Rel	7	0.27%	6.90	0.51	6.56	5.28	8.62
Carb.Rel	8	0.31%	5.44	0.13	5.40	4.96	6.06
Balling.Lvl	1	0.04%	2.05	0.87	1.48	0.00	3.66

Missing Values

Brand Code and **MFR** have high percentage of missing values. **MFR** is numeric that the missing value can be imputed using regression models. **Brand Code** can be imputed by classification models, but we would like to know if the values are missing for a specific reason. That is, if the PH of the cases with missing **Brand Code** have a significantly different distribution from the cases with provided **Brand Code**, we can flag the cases with missing **Brand Code** instead of performing imputation.

The following plot shows the distributions of **PH** by **Brand Code**.



The PH of the cases with missing **Brand Code does not** have a significantly different distribution. Hence, we will impute the missing values using classification models.

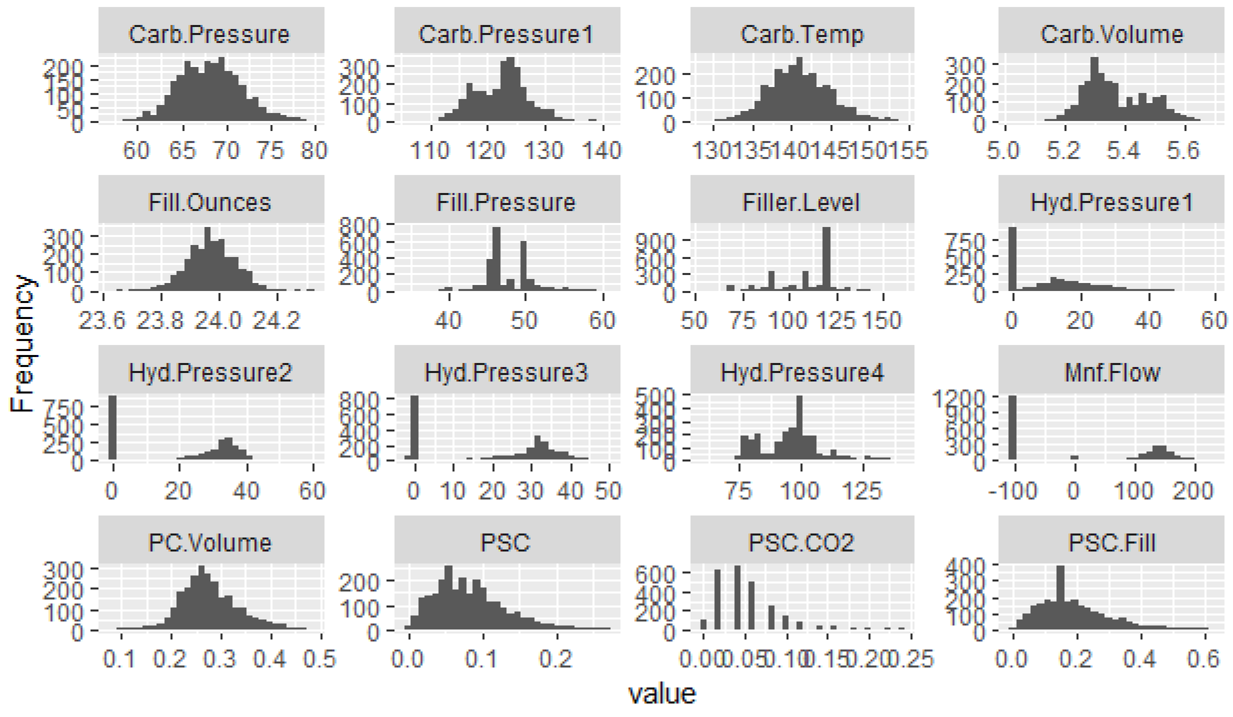
Distributions of Numeric Variables

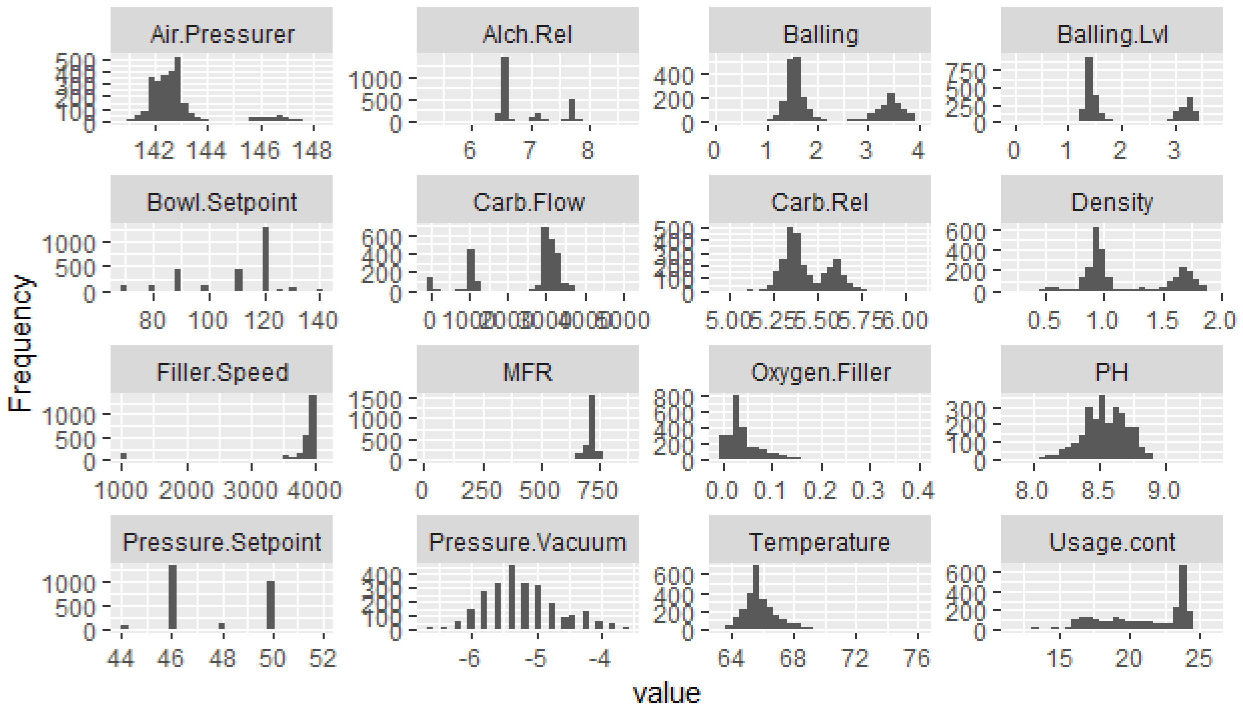
We can check the distributions of the variables to see if any transformations are needed.

From the following plots, some of the variables are highly skewed (**PSC** for example). We will apply transformations to the predictors so that the distributions of the data would be close to the normal distribution. Since there are negative values in some variables, the **Yeo Johnson**

Transformation would be suitable to do the job. Reducing skewness would help models fitting the data better.

We also see that the variables are in different scales. We will apply scaling and centering to the predictors. Some models may be biased toward variables with higher variance. Scaling helps reducing biases in the predictor importance evaluation.

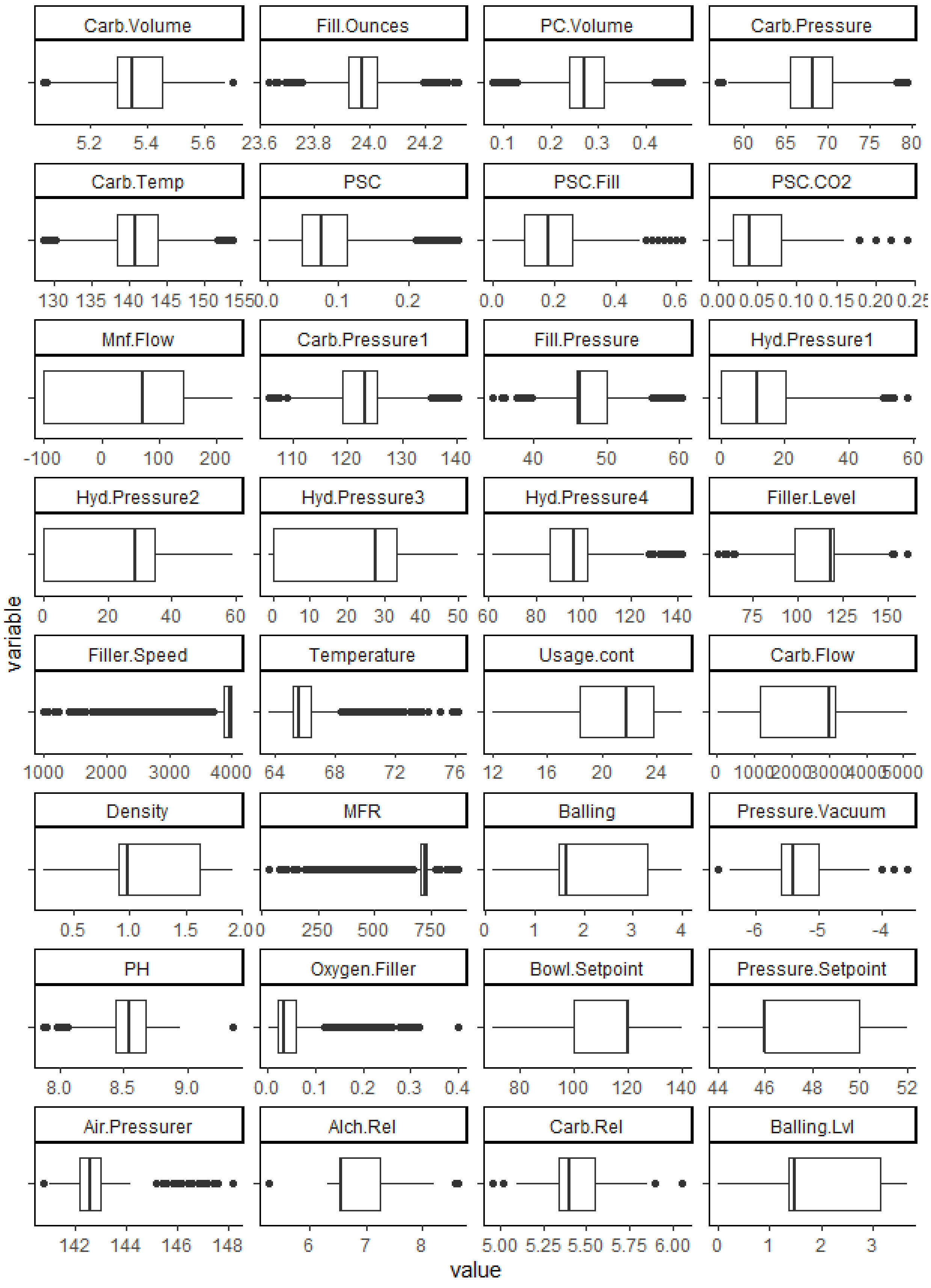




Extreme Outliers

We check the box-plots of the variables to identify extreme outliers. A single point that is far away from all other data is considered as an extreme outlier. Extreme outliers with high leverage have strong influence to models and decrease the unbiasness.

From the following plots, we find that **PH** and **Alch Rel** have one extreme outlier. We will remove the case with extremely high **PH** from our observations. For **Alch Rel**, we will remove the value and impute it with a reasonable value along with other missing values.

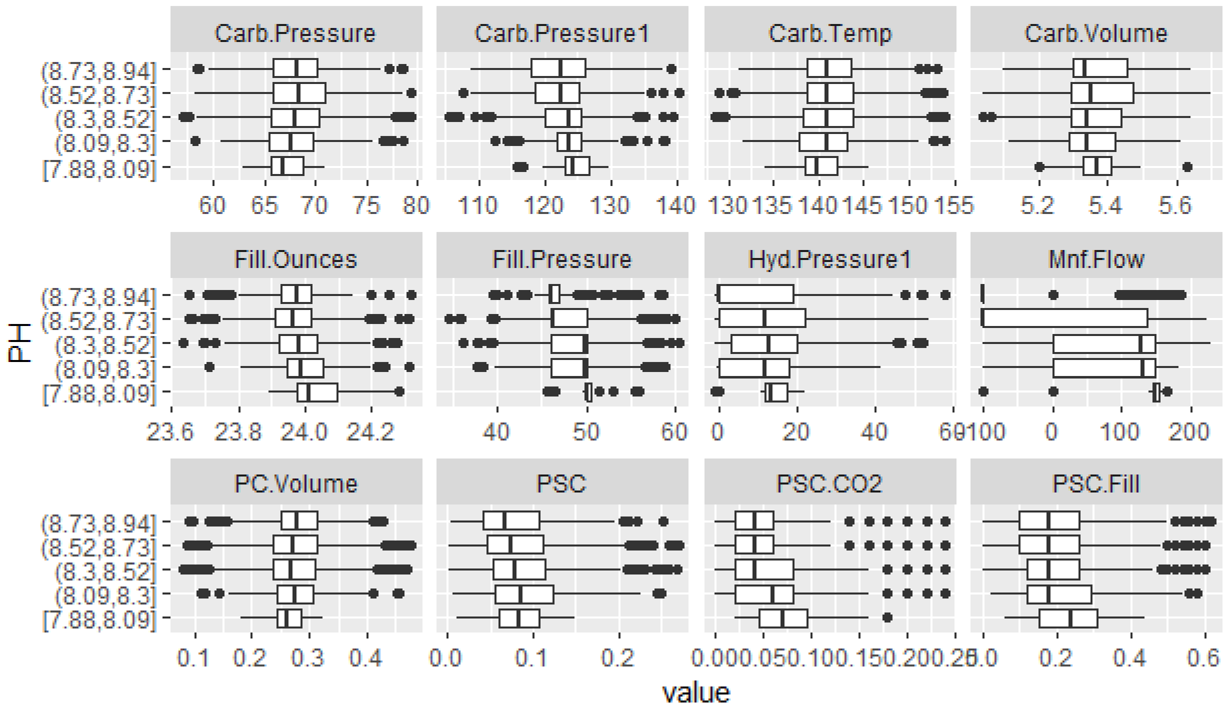

Carb.Volume
Fill.Ounces
PC.Volume
Carb.Pressure
Carb.Temp
PSC
PSC.Fill
PSC.CO2
Mnf.Flow
Carb.Pressure1
Fill.Pressure
Hyd.Pressure1
Hyd.Pressure2
Hyd.Pressure3
Hyd.Pressure4
Filler.Level
Filler.Speed
Temperature
Usage.cont
Carb.Flow
Density
MFR
Balling
Pressure.Vacuum
PH
Oxygen.Filler
Bowl.Setpoint
Pressure.Setpoint
Air.Pressurer
Alch.Rel
Carb.Rel
Balling.Lvl
variable
value

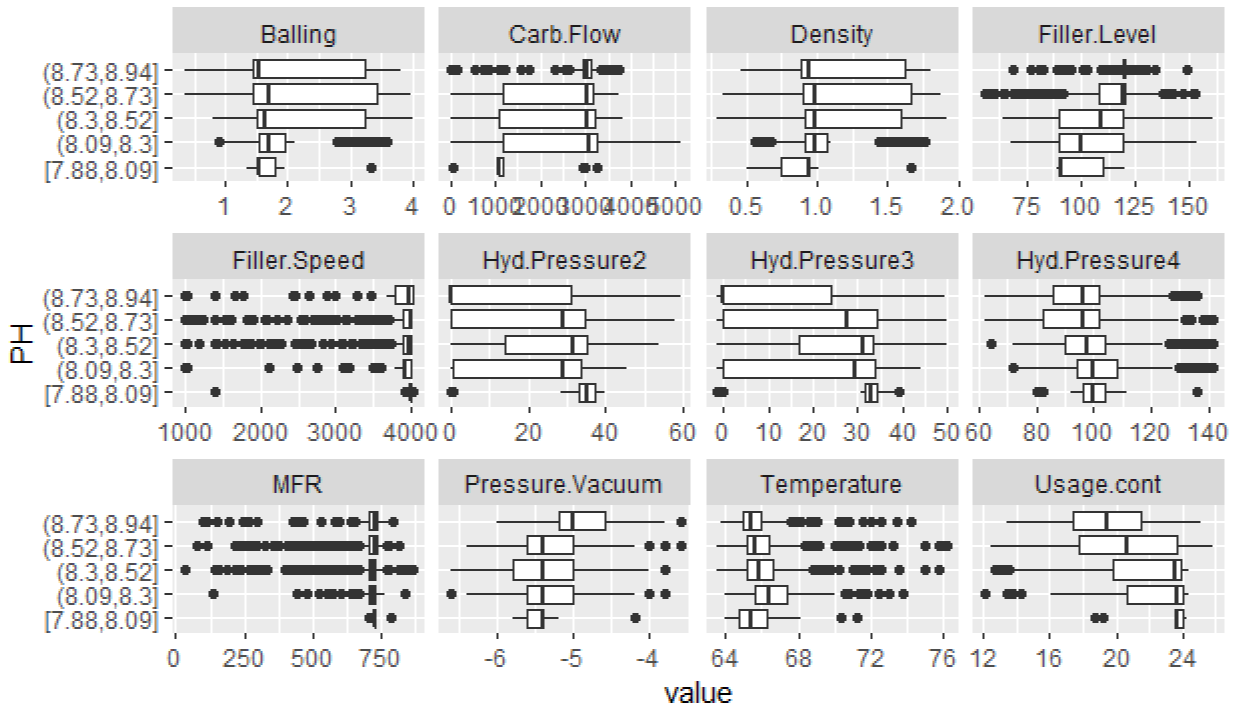
PH Vs. Predictor distributions

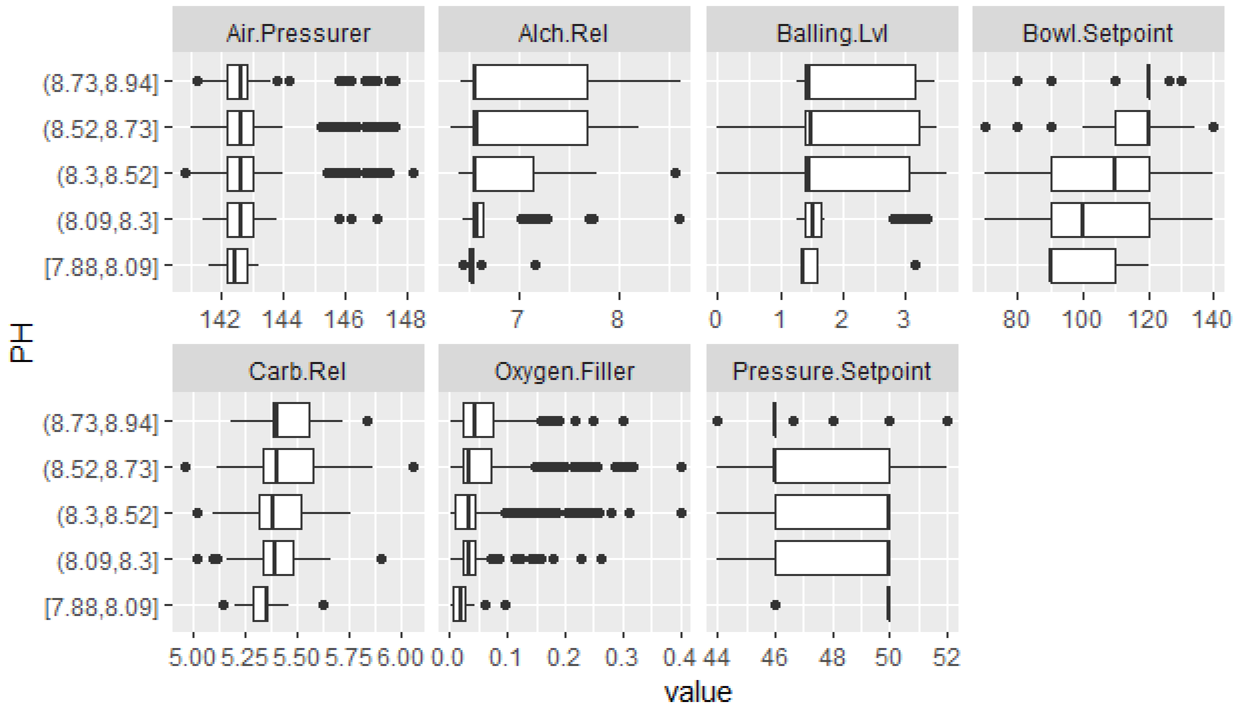
We can perform a preliminary identification of the key predictors by comparing the distributions of the numeric variables by different ranges of PH.

For example, **Bowl Setpoint** is a good predictor. Higher range of **Bowl Setpoint** has higher range of **PH**. Higher range of **Filler Level** also has higher range of **PH**. Variables like them may be highly correlated so we may only need to include one of them in our model. We will check the correlations between predictors to determine the variables to be excluded.

Variable such as **PSC** is not a good predictor as we can see from the plot that the distribution does not vary a lot across different ranges of **PH**




Balling
Carb.Flow
Density
Filler.Level
Filler.Speed
Hyd.Pressure2
Hyd.Pressure3
Hyd.Pressure4
MFR
Pressure.Vacuum
Temperature
Usage.cont
PH
(8.73,8.94]
(8.52,8.73]
(8.3,8.52]
(8.09,8.3]
[7.88,8.09]
value


Air.Pressurer
Alch.Rel
Balling.Lvl
Bowl.Setpoint
Carb.Rel
Oxygen.Filler
Pressure.Setpoint
PH
(8.73,8.94]
(8.52,8.73]
(8.3,8.52]
(8.09,8.3]
[7.88,8.09]
value

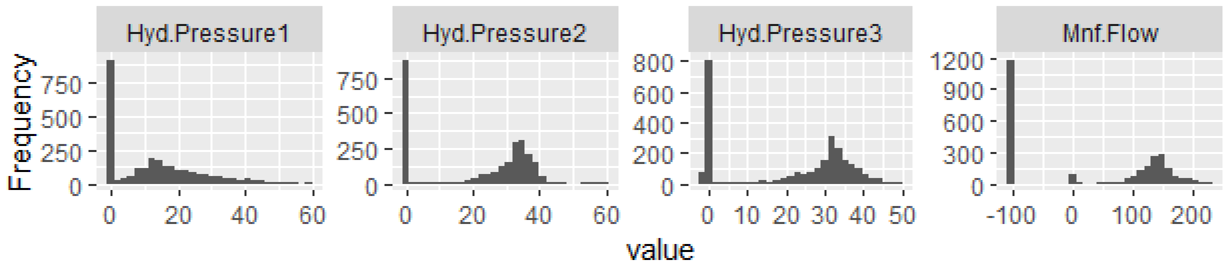
Near-zero Variance Variable

A variable that have a single value for the vast majority of the samples is considered as a near-zero variance variable. near-zero variance variables usually are not informative in predictors. A rule to identify the near-zero variance variables is to calculate the

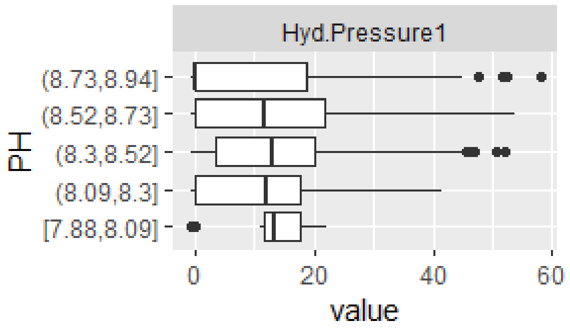
The following variable is identified as near-zero variance by a logic that calculates the ratio of the frequency of the most prevalent value to the frequency of the second most prevalent value. But should we simply remove the variable?

```
## [1] "Hyd.Pressure1"
```

We review variables with similar distributions. **Hyd Pressure2**, **Hyd Pressure3**, and **Mnf Flow** are not considered as near-zero variance since the values are not spread-out as **Hyd Pressure1**. If we separate the values of **Hyd Pressure1** into bins then the variable will not considered as near-zero variance.



Additionally, **Hyd Pressure1** is useful for predicting high **PH** value by looking at the **Hyd Pressure1 vs. PH distribution plot**



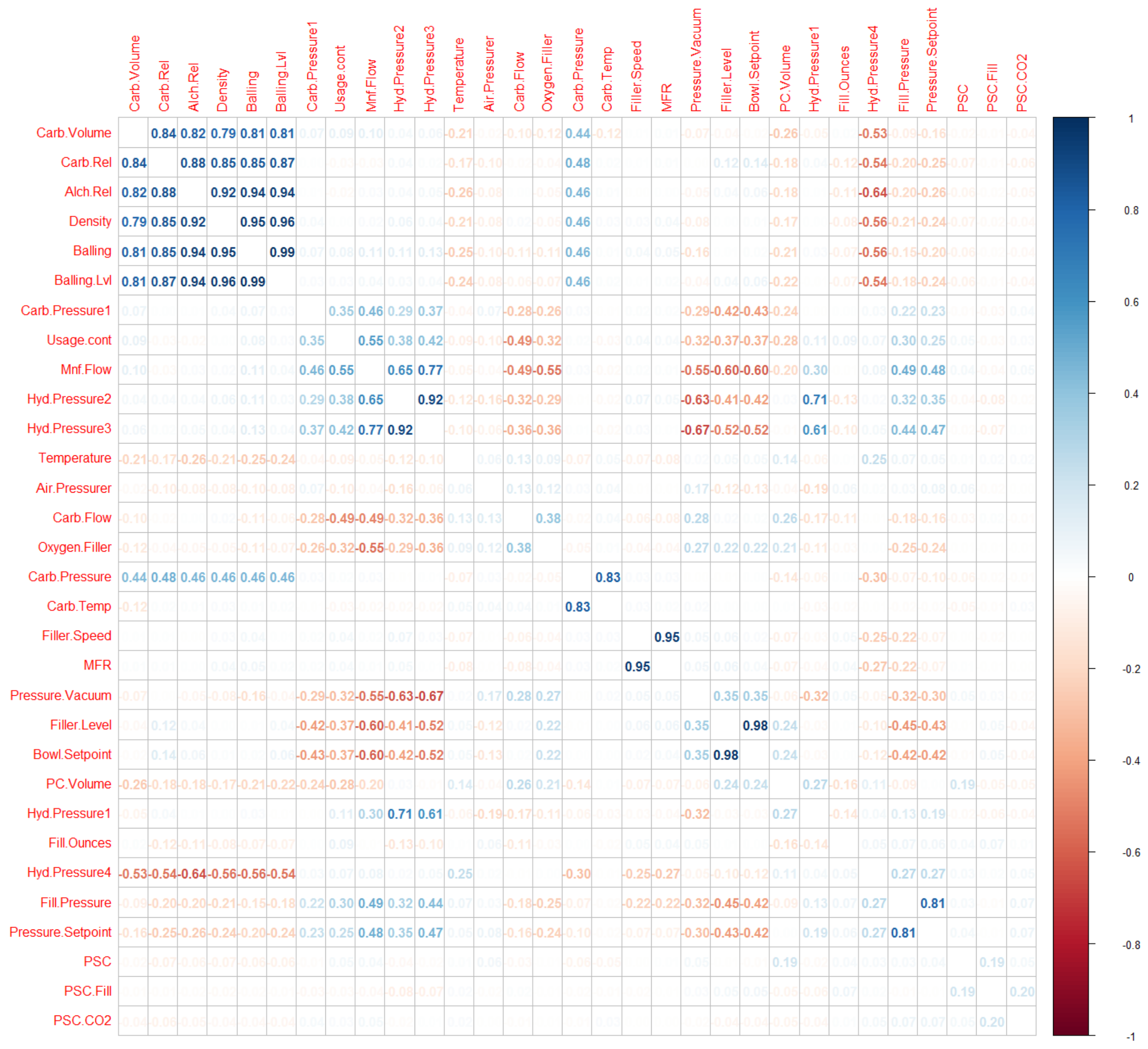
As a conclusion, we will keep the variable **Hyd Pressure1**.

Correlations Between Variables

As mentioned before, we only need to keep one of the variables that are highly correlated and exclude the other ones. There are several benefits for doing so:

- Reduce time for modeling.
- Variables carrying mostly the same information compete each other in the importance evaluation. The resulting importance for the variables are lower than they should.
- Some models such as ordinary least squares regression do not produce accurate inference result if there is multicollinearity in the data.

We can from the following plot that there are some variables with very high correlation.



We will remove variables with correlations equal or greater than 0.8 but keeping one. The following variables will be removed **after missing value imputation is done**. These variables may be useful for imputing the missing values.

```
##  [1] "Balling"       "Hyd.Pressure3" "Balling.Lvl"   "Alch.Rel"
##  [5] "Density"       "Carb.Rel"      "Fill.Pressure" "Filler.Level"
##  [9] "Filler.Speed"  "Carb.Temp"
```

Data Preprocessing

After reviewing the data, we will perform data processing using the following steps

- separate the predictors into one data frame and the response variable into another one.
- impute the missing values using Multiple imputation by chained equations (mice). The method mice uses multiple chained regression models to impute missing data.
- apply Yeo Johnson transformation, centering and scaling to the predictors.
- remove highly correlated variables but keeping one as identified above.
- create dummy variables for Brand Code. Some models do not handle categorical variables automatically. Hence, replacing categorical variables by dummy variables is needed. In this case, Brand A is used as the reference group so it is removed.
- We split the data into training set (75%) and test set (25%). The purpose of the test set is to evaluate the models' performance on unseen data.

Building Models

We select to build the following models. Linear models, non-linear models, and tree models are included.

- ordinary least squares
- partial least squares
- k-nearest neighbors
- support-vector machine
- neural network
- multivariate adaptive regression splines
- classification and regression trees
- random forests
- gradient boosting
- cubist

Model Tuning Parameters

All models, except ordinary least squares, have parameters that can be tuned to achieve higher performance. The **RMSE** (Root Mean Square Error) is used as the performance measure in the tuning process. Additionally, we resample the training data in the process using **10-fold cross-**

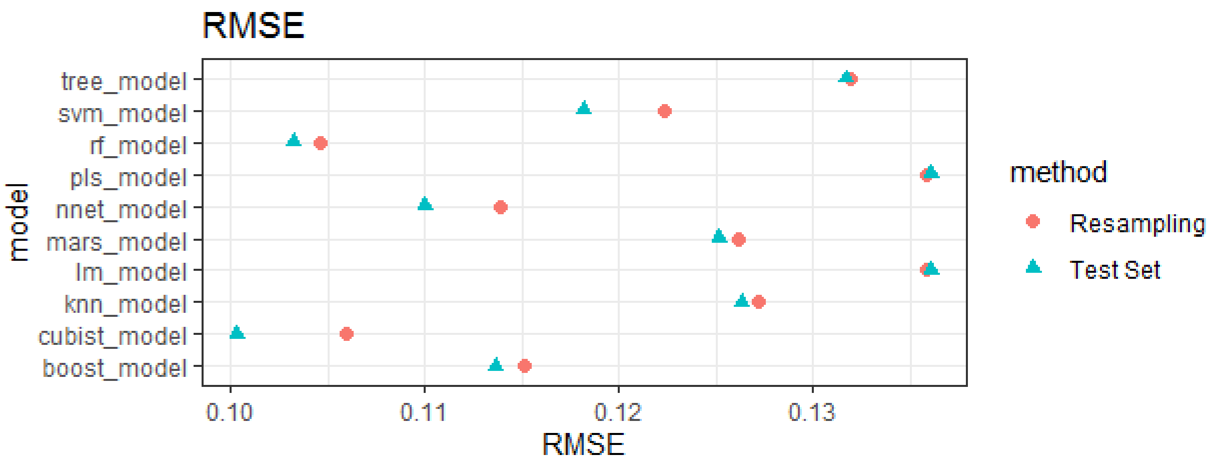
validation with 10 repeats. The averaging of the performance by using different samples can mitigate the effect of overfitting. A summary of tuning parameters for our models and the optimal values is showed below:

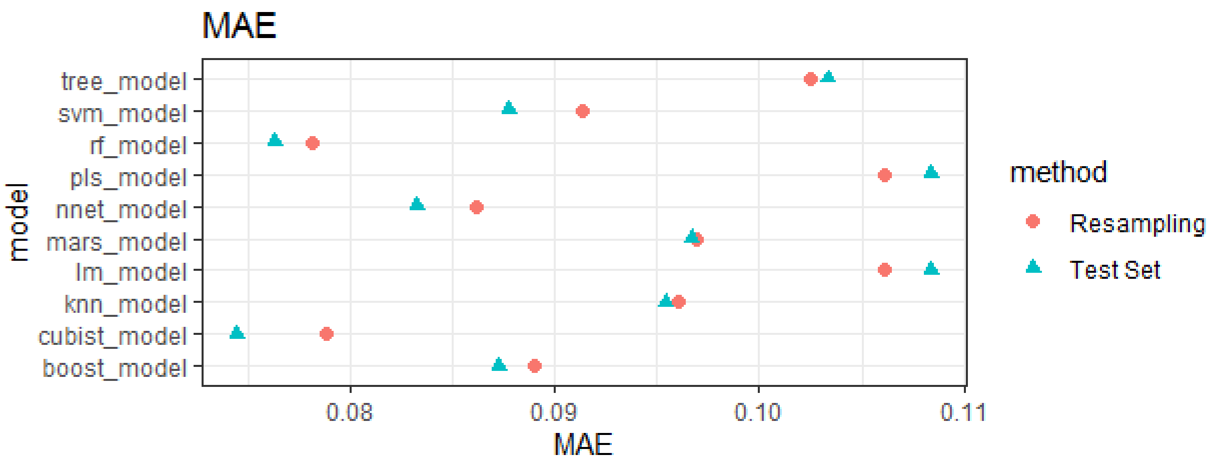
Model.Name	Model.Type	Tuning.Parameter	Tuning.Parameter.Description	Optimal.Value
pls_model	partial least squares	ncomp	number of components in the model	13.0000
knn_model	k-nearest neighbors	k	number of neighbors used for each prediction	9.0000
svm_model	support-vector machine	sigma	width of the decision boundary	0.0285
svm_model	support-vector machine	C	size of the penalty of misclassifications	2.0000
nnet_model	neural network	size	number of units (components) in hidden layer	10.0000
nnet_model	neural network	decay	regularization parameter to lower overfitting	0.0100
mars_model	multivariate adaptive regression splines	degree	maximum degree of interaction (between predictors)	2.0000
mars_model	multivariate adaptive regression splines	nprune	maximum number of terms in the pruned model	20.0000
tree_model	classification and regression trees	maxdepth	maximum depth of the tree	11.0000
rf_model	random forests	mtry	number of predictors sampled in each tree	15.0000
boost_model	gradient boosting	n.tees	the number of trees (boosting iterations)	1,000.0000
boost_model	gradient boosting	interaction.depth	the number of splits in each tree	7.0000
boost_model	gradient boosting	shrinkage	the porportion of data learned in each tree	0.0100
cubist_model	cubist	committees	the numebr of committees (boosting iterations)	20.0000
cubist_model	cubist	neighbors	the number of neighbors used to adjust the prediction	9.0000

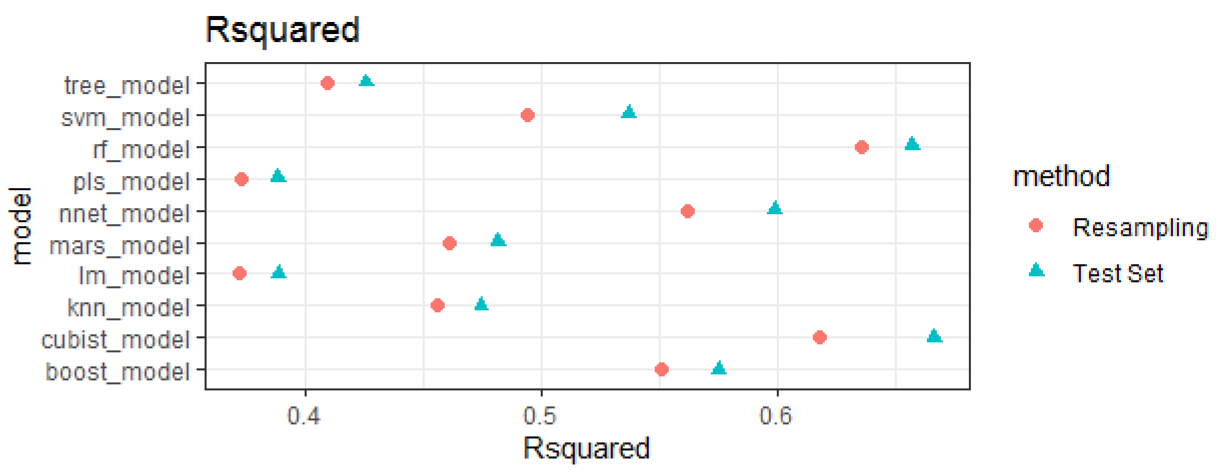
Model Evaluation

We then calculate three different performance measures (RMSE, MAE, Rsquared) of the models with the optimal tuning parameters, using the resampling / training data and test data separately.

- RMSE (root mean square error): a measure of the distances between the fitting values and the actual values. The distances are squared in the calculation so it is more sensitive to large error than MAE.
- MAE (mean absolute error): also a measure of the distances between the fitting values and the actual values. The absolute values of the distances are used instead of the squared value in RMSE.
- Rsquared: a measure of the percentage of the variance of the data explained by the model. A higher Rsquared implies that the model is able to predict higher percentage of the data correctly.







ordinary least squares and **Partial Least Squares** have the worst performance. This implies that the relation between PH and the predictors are mostly non-linear. We may transform the variables to better fit the model but that is a time consuming process and will not be done in this analysis.

Random Forests has the best resampling performance (lowest RMSE and MAR, highest Rsquared).

Cubist has the best has the best test set performance.

The resampling performance and test set performance of the two models are comparable.

To determine the optimal model, let's retune both models using the whole data set and compare the resampling performance.

model	RMSE	Rsquared	MAE
rf_model	0.10041014	0.6657363	0.07395117
cubist_model	0.09860941	0.6710419	0.07278599

Now **Cubist** has the best performance using the whole data set.

Additionally, we can compare the optimal tuning parameter(s) of the models using the training data only and using the whole data set.

The optimal tuning parameters of the **Random Forests** model are

data	mtry
splited_data	15
all_data	17

The optimal tuning parameters of the **Cubist** model are

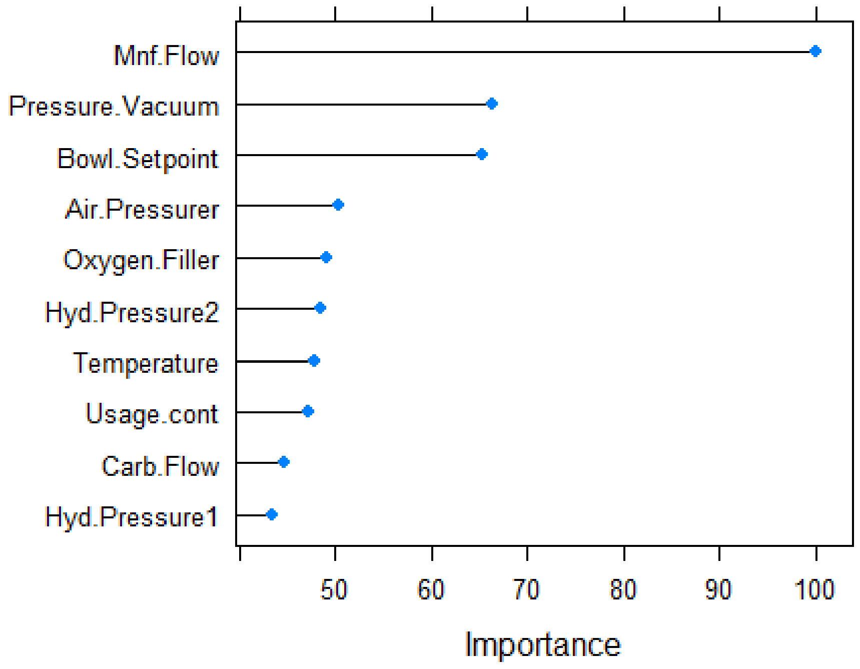
committees	neighbors	data
20	9	splited_data
20	9	all_data

The optimal tuning parameter of the **Cubist** model is more stable. If we are given a larger data set in the future to rebuild the model, we can save the time tuning the model and reuse the same parameters. Considering this and the performance of the model, we select the **Cubist** model as our optimal model.

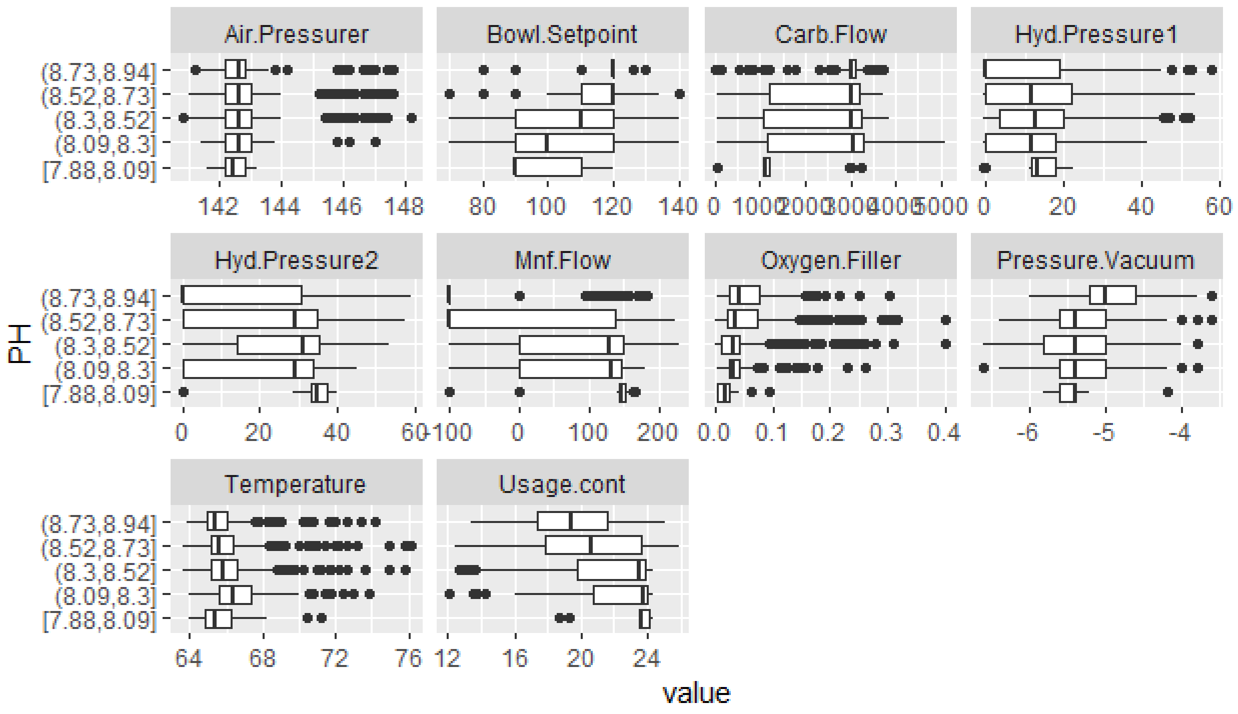
Conclusion

Now let's find the most important factors that can be used to help improving our manufacturing process.

The following plots shows the 10 most important predictors from our **Cubist** model.



We can check how these factors affect PH by viewing the PH vs. predictor distributions again.



From the plots, we have the following findings:

- We can increase **Mnf Flow**, **Hyd Pressure2**, **Temperature**, **Usage Cont**, and **Hyd Pressure1** to lower **PH**
- we can lower **Pressure Vacuum**, **Bowl Setpoint**, **Air Pressurer**, **Oxygen Filler**, and **Carb Flow** to lower **PH**
- If resouce is limited, we suggest to focus on the top 3 most important factors: **Mnf Flow**, **Pressure Vacuum**, **Bowl Setpoint**

Factors that are highly correlated with the key factors can also be used to adjust the manufacturing process, though they were removed in the modeling process.

New Data Prediction

We load a new set of data for prediction.

First we identify the data values that are not within the ranges of the predictors in the training data.

Hyd.Pressure1	Hyd.Pressure2	Hyd.Pressure3	Carb.Flow	Density	MFR
-50.0	-50.0	-50.0	0	0.06	
32.6	61.4	34.8	3,286	0.90	706.0
0.0	0.2	0.0	44	1.60	15.6
-1.0	0.2	-1.2	38		582.2

The lower and upper bounds of the predictors in the training data are

variable	min	max
Hyd.Pressure1	-0.8	58.00
Hyd.Pressure2	0.0	59.40
Hyd.Pressure3	-1.2	50.00
Carb.Flow	26.0	5,104.00
Density	0.3	1.92
MFR	31.4	868.60

In the first record, **Hyd Pressure1**, **Hyd Pressure2**, and **Hyd Pressure3** are far away from the boundaries. A **Carb Flow** of 0 is also seems abnormal. It can be data collection error, or a malfunction in the manufacturing process. If it is a malfunction, then it would be meaningless to predict the PH as the products must be examined and most likely destroyed. We will remove these unreasonable values and the numbers will be replaced by imputation along with other missing values.

For the other 3 records, the values are reasonable as they are close to the boundaries. We can keep them as they are and perform prediction. Tree models such as **Random Forests** and rule based models such as **Cubist** can handle these unseen values.

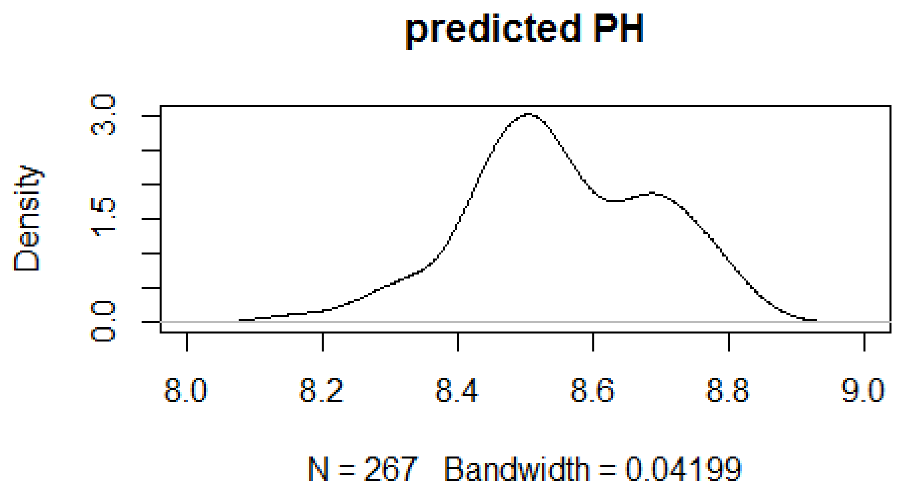
The following shows the replaced values of the unreasonable values.

Hyd.Pressure1	Hyd.Pressure2	Hyd.Pressure3	Carb.Flow	Density	MFR
0.0	0.0	38.6	1,174	0.06	735.0
32.6	61.4	34.8	3,286	0.90	706.0
0.0	0.2	0.0	44	1.60	15.6
-1.0	0.2	-1.2	38	0.98	582.2

Hyd Pressure1, **Hyd Pressure2**, and **Hyd Pressure3** are close to the lower boundaries and **Carb Flow** is comparatively lower (considering the range is 26-5140 in the training data).

We impute the missing values using the same MICE models created when imputing the training data.

We then apply the same Yeo Johnson transformation, centering and scaling to the test data as we did to the training data. The data then can be used to perform prediction. The distribution of the predicted PH is showed below.



The distribution of the predicted PH looks plausible so we can conclude that our model produce valid prediction.

Coding

```
#Loading libraries
library(psych)  #describe function for data summary
library(stringr)
library(dplyr)
library(tidyr)
library(DataExplorer) #data distribution plots
library(caret)  #data processing and modeling
library(reshape2)
library(corrplot) #correlation plot
library(RANN) #required for knn
library(mice) #data imputation
library(NADIA) #reuse mice
library(officer)  #flextable boarder

library(flextable)  #displaying tables in nice format
set_flextable_defaults(font.size = 9)


#loading data
raw_df <- readxl::read_xlsx("StudentData - TO MODEL.xlsx", skip = 0, sheet = 'Subset')
raw_df <- as.data.frame(raw_df)
raw_df$`Brand Code` <- as.factor(raw_df$`Brand Code`) #convert categorical variable to factor
raw_df <- raw_df %>% filter(!is.na(PH)) #filter out records with response variable missing
names(raw_df)<-str_replace_all(names(raw_df), c(" " = ".")) #replace space character in variabl
e names


#raw data summary
raw_summary <- describe(raw_df)
n_records <- nrow(raw_df)
n_var <- raw_summary$n
raw_summary["Brand.Code*",] <- 0
raw_summary$n <- NULL
raw_summary$n_NA <- n_records - n_var
raw_summary$pct_NA <- 100*raw_summary$n_NA / n_records
raw_summary[,-1] <- round(raw_summary[,-1],2)
raw_summary$pct_NA <- paste0(as.character(raw_summary$pct_NA), "%")
raw_summary$var <- rownames(raw_summary)
raw_summary %>% select(var, n_NA, pct_NA, mean, sd, median, min, max) %>%
  flextable() %>% autofit() %>% border(border = fp_border(color = "black"))


#Distribution of PH by Brand Code
```

```
ggplot(raw_df, aes(x=PH, color=`Brand.Code`)) + geom_density()

#Distribution of numeric variables
plot_histogram(raw_df)


#Box plot of numeric variable for identifying extreme outliers
data.m <- melt(raw_df[-1])
ggplot(data.m, aes(x = variable, y = value)) + geom_boxplot() + coord_flip() +
  facet_wrap(~ variable, scales = 'free',ncol=4) + theme_classic() +
  theme(axis.text.y=element_blank())

#remove extreme outliers
raw_df <- raw_df[-which.max(raw_df$PH),]
raw_df$Alch.Rel[which.min(raw_df$Alch.Rel)] <- NA


#Comparing the distributions of the numeric variables by different ranges of PH
plot_boxplot(raw_df, by = "PH")


#Finding zero variance variable
colnames(raw_df)[nearZeroVar(raw_df)]


#Correlation plot of the predictors
cor_matrix <- cor(subset(raw_df, select=-c(Brand.Code,PH)), use = "na.or.complete")
corrplot::corrplot(cor_matrix, method = 'number',  order='hclust',
         diag = FALSE)


#Identify the predictors to be removed due to multicollinearity
#These predictors will be removed after missing value imputation since they may be useful for th
e imputation
high_corr <- findCorrelation(cor_matrix, cutoff = 0.8, names = TRUE)
high_corr


#separate the predictors into one data frame and the response variable into another one
processed_x <- subset(raw_df, select=-c(PH))
```

```
processed_y <- raw_df$PH


#save the imputation models to impute the test data set later
mickey <- parlmice(processed_x, maxit = 5, m = 1, printFlag = FALSE, seed = 624,
                   cluster.seed = 624)

#save the imputation result
processed_x <- complete(mickey,1)


#Yeo Johnson transformation, centering and scaling
set.seed(624)
data_processor <- preProcess(processed_x, method = c("YeoJohnson", "center", "scale"))
processed_x <- predict(data_processor, processed_x)


#remove the highly correlated predictors identified above
processed_x <- processed_x[,!names(processed_x) %in% high_corr]


#Create dummy variables for Brand Code. Brand A is used as the reference group so it is remov
ed.
dummy_gen <- dummyVars(~.,
                      data = processed_x,
                      levelsOnly = TRUE)
processed_x <- as.data.frame(predict(dummy_gen, processed_x))
#remove brand A
processed_x <- processed_x[-1]


#Split the data into training set (75%) and test set (25%)
set.seed(624)
split <- createDataPartition(processed_y, p=0.75, list=F)
x_train <- processed_x[split, ]
y_train <- processed_y[split]
x_test <- processed_x[-split, ]
y_test <- processed_y[-split]
```

```
#Set up resampling control. We will use 10-fold cross-validation with 10 repeats
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)

#The model tunning is time consuming. We saved the tuned models in the
#tuned_models.RData file. The tunning reuslt can be loaded from the file
#or running the tunning process in the else block
if (file.exists("tuned_models.RData")) {
  load("tuned_models.RData")
} else {

  #ordinary linear regression
  set.seed(624)
  lm_model <- train(x = x_train,
              y = y_train,
              method= 'lm',
              metric = "RMSE",
              trControl = ctrl)

  #partial least squares
  set.seed(624)
  pls_model <- train(x = x_train,
                     y = y_train,
                     method = "pls",
                     metric = "RMSE",
                     tuneLength = 30,
                     trControl=ctrl)

  #k-nearest neighbors
  set.seed(624)
  knn_model <- train(x = x_train,
                     y = y_train,
                     method = "knn",
                     metric = "RMSE",
                     tuneLength = 30,
                     trControl = ctrl)

  #support-vector machine
  set.seed(624)
  svm_model <- train(x = x_train,
                    y = y_train,
```

```r
                 method = "svmRadial",
                 metric = "RMSE",
                 tuneLength = 30,
                 trControl = ctrl)

#neural network
nnetGrid <- expand.grid(.decay = c(0, 0.01, 0.1),
                        .size = c(1:10),
                        .bag = FALSE)
set.seed(624)
nnet_model <- train(x = x_train,
                    y = y_train,
                    method = 'avNNet',
                    metric = "RMSE",
                    tuneGrid = nnetGrid,
                    linout = TRUE,
                    trace = FALSE,
                    MaxNWts = 10 * (ncol(x_train) + 1) + 10 + 1,
                    maxit = 500,
                    trControl = ctrl,
                    verbose = FALSE)

#multivariate adaptive regression splines
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:20)
set.seed(624)
mars_model <- train(x = x_train,
                    y = y_train,
                    method = 'earth',
                    metric = "RMSE",
                    tuneGrid = marsGrid,
                    trControl = ctrl)

#classification and regression trees
set.seed(624)
tree_model <- train(x = x_train,
                    y = y_train,
                    method = "rpart2",
                    metric = "RMSE",
                    tuneLength = 20,
                    trControl = ctrl)
```

```
 #random forests
 set.seed(624)
 rf_model <- train(x = x_train,
                   y = y_train,
                   method = 'rf',
                   metric = "RMSE",
                   tuneLength = 20,
                   trControl = ctrl)

 #gradient boosting
 boostGrid <- expand.grid(n.trees=seq(100, 1000, by = 100),
                          interaction.depth=seq(1, 7, by = 1),
                          shrinkage=c(0.01, 0.1, 0.5),
                          n.minobsinnode = 10)
 set.seed(624)
 boost_model <- train(x = x_train,
                      y = y_train,
                      method = 'gbm',
                      metric = "RMSE",
                      tuneGrid = boostGrid,
                      trControl = ctrl,
                      verbose = FALSE)

 #cubist
 set.seed(624)
 cubist_model <- train(x = x_train,
                       y = y_train,
                       method = 'cubist',
                       metric = "RMSE",
                       trControl = ctrl)

 #Save the model results
 save(lm_model, pls_model, knn_model, svm_model, nnet_model, mars_model, tree_model,
      rf_model, boost_model, cubist_model, file = "tuned_models.RData")

}


#Model tunning parameters summary
```

```
tuning_para <- read.csv(file = 'https://raw.githubusercontent.com/ezaccountz/DATA_624/main/P
roject2/Model%20Tuning%20Parameters.csv')

tuning_para$Optimal.Value <- c(pls_model$bestTune$ncomp,
                              knn_model$bestTune$k,
                              svm_model$bestTune$sigma,
                              svm_model$bestTune$C,
                              nnet_model$bestTune$size,
                              nnet_model$bestTune$decay,
                              mars_model$bestTune$degree,
                              mars_model$bestTune$nprune,
                              tree_model$bestTune$maxdepth,
                              rf_model$bestTune$mtry,
                              boost_model$bestTune$n.trees,
                              boost_model$bestTune$interaction.depth,
                              boost_model$bestTune$shrinkage,
                              cubist_model$bestTune$committees,
                              cubist_model$bestTune$neighbors)
tuning_para$Optimal.Value <- round(tuning_para$Optimal.Value,4)
wd_table <- tuning_para %>% flextable() %>% autofit() %>% border(border = fp_border(color
= "black"))
pgwidth = 6.5
width(wd_table, width = dim(wd_table)$widths*pgwidth /(flextable_dim(wd_table)$widths))


#Resampling performance
resample_perform <- rbind(
  lm_model=lm_model$results[which.min(lm_model$results$RMSE),c("RMSE","Rsquared","M
AE")],
  pls_model=pls_model$results[which.min(pls_model$results$RMSE),c("RMSE","Rsquared","
MAE")],
  knn_model=knn_model$results[which.min(knn_model$results$RMSE),c("RMSE","Rsquared",
"MAE")],
  svm_model=svm_model$results[which.min(svm_model$results$RMSE),c("RMSE","Rsquared
","MAE")],
  nnet_model=nnet_model$results[which.min(nnet_model$results$RMSE),c("RMSE","Rsquared
","MAE")],
  mars_model=mars_model$results[which.min(mars_model$results$RMSE),c("RMSE","Rsquar
ed","MAE")],
  tree_model=tree_model$results[which.min(tree_model$results$RMSE),c("RMSE","Rsquared",
"MAE")],
  rf_model=rf_model$results[which.min(rf_model$results$RMSE),c("RMSE","Rsquared","MA
E")],
  boost_model=boost_model$results[which.min(boost_model$results$RMSE),c("RMSE","Rsqua
red","MAE")],
  cubist_model=cubist_model$results[which.min(cubist_model$results$RMSE),c("RMSE","Rsq
uared","MAE")]
)


#Test data performance
lm_pred <- predict(lm_model, newdata = x_test)
pls_pred <- predict(pls_model, newdata = x_test)
knn_red <- predict(knn_model, newdata = x_test)
svm_pred <- predict(svm_model, newdata = x_test)
nnet_pred <- predict(nnet_model, newdata = x_test)
mars_pred <- predict(mars_model, newdata = x_test)
tree_pred <- predict(tree_model, newdata = x_test)
rf_pred <- predict(rf_model, newdata = x_test)
boost_pred <- predict(boost_model, newdata = x_test)
cubist_pred <- predict(cubist_model, newdata = x_test)

test_perform <- as.data.frame(rbind(
  lm_model=postResample(pred = lm_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  pls_model=postResample(pred = pls_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  knn_model=postResample(pred = knn_red, obs = y_test)[c("RMSE","Rsquared","MAE")],
  svm_model=postResample(pred = svm_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  nnet_model=postResample(pred = nnet_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  mars_model=postResample(pred = mars_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  tree_model=postResample(pred = tree_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  rf_model=postResample(pred = rf_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  boost_model=postResample(pred = boost_pred, obs = y_test)[c("RMSE","Rsquared","MAE")],
  cubist_model=postResample(pred = cubist_pred, obs = y_test)[c("RMSE","Rsquared","MAE")]
))


#RMSE comparison plot
ggplot(data=data.frame(model=c(row.names(resample_perform), row.names(test_perform)),
                       method=c(rep("Resampling",nrow(resample_perform)),rep("Test Set",nrow(test_p
erform))),
                    RMSE=c(resample_perform$RMSE,test_perform$RMSE)),
      aes(x=RMSE, y=model, color=method, shape=method)
) + geom_point(size=2) + theme_bw() + ggtitle("RMSE")


#MAE comparison plot
ggplot(data=data.frame(model=c(row.names(resample_perform), row.names(test_perform)),
                    method=c(rep("Resampling",nrow(resample_perform)),rep("Test Set",nrow(test_p
erform))),
                    MAE=c(resample_perform$MAE,test_perform$MAE)),
      aes(x=MAE, y=model, color=method, shape=method)
) + geom_point(size=2) + theme_bw() + ggtitle("MAE")


#Rsquared comparison plot
ggplot(data=data.frame(model=c(row.names(resample_perform), row.names(test_perform)),
                    method=c(rep("Resampling",nrow(resample_perform)),rep("Test Set",nrow(test_p
erform))),
                    Rsquared=c(resample_perform$Rsquared,test_perform$Rsquared)),
      aes(x=Rsquared, y=model, color=method, shape=method)
) + geom_point(size=2) + theme_bw() + ggtitle("Rsquared")


#final tunning with all data
if (file.exists("tuned_models2.RData")) {
  load("tuned_models2.RData")
} else {

  #random forests
  set.seed(624)
  rf_model2 <- train(x = processed_x,
                  y = processed_y,
                  method = 'rf',
                  metric = "RMSE",
                  tuneLength = 20,
                  trControl = ctrl)

  #cubist
  set.seed(624)
  cubist_model2 <- train(x = processed_x,
                        y = processed_y,
                        method = 'cubist',
                        metric = "RMSE",
                        trControl = ctrl)

  save(rf_model2, cubist_model2, file = "tuned_models2.RData")
}


#Performance of final tuning
resample_perform2 <- rbind(
  rf_model=rf_model2$results[which.min(rf_model2$results$RMSE),c("RMSE","Rsquared","M
AE")],
  cubist_model=cubist_model2$results[which.min(cubist_model2$results$RMSE),c("RMSE","R
squared","MAE")]
)
wd_table <- resample_perform2
wd_table$model <- rownames(wd_table)
wd_table[c(4,1,2,3)] %>% flextable() %>% autofit() %>% border(border = fp_border(color = "b
lack"))


#Compare the optimal values of the tuning parameters of the random forests model, with the spli
t data and all data
wd_table <- rbind(splited_data=rf_model$bestTune, all_data=rf_model2$bestTune)
wd_table$data <- rownames(wd_table)
wd_table[c(2,1)] %>% flextable() %>% autofit() %>% border(border = fp_border(color = "black
"))


#Compare the optimal values of the tuning parameters of the cubist model, with the split data an
d all data
wd_table <- rbind(splited_data=cubist_model$bestTune, all_data=cubist_model2$bestTune)
wd_table$data <- rownames(wd_table)
wd_table %>% flextable() %>% autofit() %>% border(border = fp_border(color = "black"))


#Predictor importance of the final cubist model
model_imp <- varImp(cubist_model2)
plot(varImp(cubist_model2), top=10)
```

```
#Review of the box-plots of the top 10 most important predictors vs. PH
top_10 <- as.data.frame(model_imp$importance) %>% arrange(desc(Overall)) %>%  top_n(10)
 %>% rownames()
plot_boxplot(subset(raw_df, select=c("PH", top_10)), by = "PH")


#Load the evaluation data
test_df <- readxl::read_xlsx("StudentEvaluation- TO PREDICT.xlsx", skip = 0, sheet = 'Subset
(2)')
test_df <- as.data.frame(test_df)
test_df$`Brand Code` <- as.factor(test_df$`Brand Code`)
names(test_df)<-str_replace_all(names(test_df), c(" " = "."))
test_df_x <- subset(test_df, select=-c(PH))


#Identify data values that are not within the ranges of the predictors in the training data.
#Find the columuns and rows contain those data
train_min <- apply(subset(raw_df, select=-c(Brand.Code, PH)), 2, min, na.rm=TRUE)
train_max <- apply(subset(raw_df, select=-c(Brand.Code, PH)), 2, max, na.rm=TRUE)
out_of_bounds <- apply(test_df_x[-1], 1, function(x) x<train_min | x>train_max)
abnormal_col <- apply(out_of_bounds,1,any)
abnormal_col[is.na(abnormal_col)] <- FALSE
abnormal_row <- apply(out_of_bounds,2,any)
abnormal_row[is.na(abnormal_row)] <- FALSE

#data values out of the boundaries
test_df_x[-1][abnormal_row, abnormal_col] %>% flextable() %>% autofit() %>% border(border
 = fp_border(color = "black"))

#lower and upper bounds of the predictors in the training data
wd_table <- data.frame(min=train_min[abnormal_col], max=train_max[abnormal_col])
wd_table$variable <- rownames(wd_table)
wd_table[c(3,1,2)] %>% flextable() %>% autofit() %>% border(border = fp_border(color = "bla
ck"))

#remove the unreasonable values. The values will be replaced by imputation along with other mi
ssing values
test_df_x[-1][abnormal_row, abnormal_col][1,c(1:4)] <- NA
```

```
#missing value imputation, using MICE models from the training data
processed_test <- mice.reuse(mickey, test_df_x, maxit = 5, printFlag = FALSE, seed = 624)[[1]]

#result of the imputation for the removed unreasonable values
processed_test[-1][abnormal_row, abnormal_col] %>% flextable() %>% autofit() %>% border(b
order = fp_border(color = "black"))

#apply the same Yeo Johnson transformation, centering and scaling to the test data
set.seed(624)
processed_test <- predict(data_processor, processed_test)

#remove highly correlated variables
processed_test <- processed_test[,!names(processed_test) %in% high_corr]

#create dummy variables
dummy_gen <- dummyVars(~.,
                    data = processed_test,
                    levelsOnly = TRUE)
processed_test <- as.data.frame(predict(dummy_gen, processed_test))
processed_test <- processed_test[-1]

#prediction using the evaluation data
PH_pred <- predict(cubist_model2, newdata = processed_test)

#Distribution of the predicted PH
plot(density(PH_pred), main="predicted PH")

#save the prediction in an csv file
#write.csv(PH_pred,"PH_pred.csv", row.names = TRUE)
```