DATA 624 PROJECT 2: Regression modeling - PH prediction

DATA 624 - Predictive Analytics

Group 2

Group Members:

Sang Yoon (Andy) Hwang

10 December 2019

Contents

Introduction

The goal of this project is to predict PH, a measure of acidity/alkalinity, using train data set from a beverage company which consists of 2571 rows of data and 33 variables. After creating models based on training data, we will test on scoring set of 267 rows with 32 variables (excluding target variable which is PH in our training set)

As a group project, each member of the group is responsible for creating their own models of choice. For instance, my own selections were PLS and Bagged Tree. However, the choice of models can be altered after careful review of data exploration - it may require different type of model in case data suffers from outliers or any other data related issues.

Explaining why some necessary steps were applied before modeling and model A was preferred to Model B is often a topic in academic papers which is a meaningful topic that helps audience learn the concept of bagged regression and least square method better.

The final version of report will contain all of our approaches with results of MAPE for each model with detailed explanation of why/what/how each model of choice was constructed.

1 Data Exploration

Data dictionary

The table below describes the variables in the train data set.

Table 1.1: Data dictionary

Name	Type	MD code	Length	Measurement Type	Excluded
Test Time	Double	-9999	8	m/d/yy h:mm Am/Pm	Auto
Brand Code	Double	-9999	8	General	Categorical
Carb Volume	Double	-9999	8	General	Auto
Fill Ounces	Double	-9999	8	General	Auto
PC Volume	Double	-9999	8	General	Auto
Carb Pressure	Double	-9999	8	General	Auto
Carb Temp	Double	-9999	8	General	Auto
PSC	Double	-9999	8	General	Auto
PSC Fill	Double	-9999	8	General	Auto
PSC CO2	Double	-9999	8	General	Auto
Mnf Flow	Double	-9999	8	General	Auto
Carb Pressure1	Double	-9999	8	General	Auto
Fill Pressure	Double	-9999	8	General	Auto
Hyd Pressure1	Double	-9999	8	General	Auto
Hyd Pressure2	Double	-9999	8	General	Auto
Hyd Pressure3	Double	-9999	8	General	Auto
Hyd Pressure4	Double	-9999	8	General	Auto
Filler Level	Double	-9999	8	General	Auto
Filler Speed	Double	-9999	8	General	Auto
Temperature	Double	-9999	8	General	Auto
Usage cont	Double	-9999	8	General	Auto
Carb Flow	Double	-9999	8	General	Auto
Density	Double	-9999	8	General	Auto
MFR	Double	-9999	8	General	Auto
Balling	Double	-9999	8	General	Auto
Pressure Vacuum	Double	-9999	8	General	Auto
PH	Double	-9999	8	General	Auto
Oxygen Filler	Double	-9999	8	General	Auto
Bowl Setpoint	Double	-9999	8	General	Auto
Pressure Setpoint	Double	-9999	8	General	Auto
Air Pressurer	Double	-9999	8	General	Auto
Alch Rel	Double	-9999	8	General	Auto
Carb Rel	Double	-9999	8	General	Auto
Balling Lvl	Double	-9999	8	General	Auto
sample	Double	-999999998	8	General	Auto

Summary statistics

Since we are implementing models which do not require hard assumptions of joint distribution of variables, normality assumption is not required. We will focus on how to handle missing values only.

It is apparent, from below, that we have variables that are missing values - we will impute NULLs with MICE later on.

Table 1.2: Summary statistics

metric	missing	min	Q1	mean	median	Q3	max	sd
AirPressurer	0	140.8000000	142.2000000	142.8339946	142.6000000	143.000000	148.200	1.2119170
AlchRel	9	5.2800000	6.5400000	6.8974161	6.5600000	7.240000	8.620	0.5052753
Balling	1	-0.1700000	1.4960000	2.1977696	1.6480000	3.292000	4.012	0.9310914
BallingLvl	1	0.0000000	1.3800000	2.0500078	1.4800000	3.140000	3.660	0.8703089
BowlSetpoint	2	70.0000000	100.0000000	109.3265862	120.0000000	120.000000	140.000	15.3031541
CarbFlow	2	26.0000000	1144.0000000	2468.3542234	3028.0000000	3186.000000	5104.000	1073.6964743
CarbPressure	27	57.0000000	65.6000000	68.1895755	68.2000000	70.600000	79.400	3.5382039
CarbPressure1	32	105.6000000	119.0000000	122.5863726	123.2000000	125.400000	140.200	4.7428819
CarbRel	10	4.9600000	5.3400000	5.4367825	5.4000000	5.540000	6.060	0.1287183
CarbTemp	26	128.6000000	138.4000000	141.0949234	140.8000000	143.800000	154.000	4.0373861
CarbVolume	10	5.0400000	5.2933333	5.3701978	5.3466667	5.453333	5.700	0.1063852
Density	1	0.2400000	0.9000000	1.1736498	0.9800000	1.620000	1.920	0.3775269
FillerLevel	20	55.8000000	98.3000000	109.2523716	118.4000000	120.000000	161.200	15.6984241
FillerSpeed	57	998.0000000	3888.0000000	3687.1988862	3982.0000000	3998.000000	4030.000	770.8200208
FillOunces	38	23.6333333	23.9200000	23.9747546	23.9733333	24.026667	24.320	0.0875299
FillPressure	22	34.6000000	46.0000000	47.9221656	46.4000000	50.000000	60.400	3.1775457
HydPressure1	11	-0.8000000	0.0000000	12.4375781	11.4000000	20.200000	58.000	12.4332538
HydPressure2	15	0.0000000	0.0000000	20.9610329	28.6000000	34.600000	59.400	16.3863066
HydPressure3	15	-1.2000000	0.0000000	20.4584507	27.6000000	33.400000	50.000	15.9757236
HydPressure4	30	52.0000000	86.0000000	96.2888627	96.0000000	102.000000	142.000	13.1225594
MFR	212	31.4000000	706.3000000	704.0492582	724.0000000	731.000000	868.600	73.8983094
MnfFlow	2	-100.2000000	-100.0000000	24.5689373	65.2000000	140.800000	229.400	119.4811263
OxygenFiller	12	0.0024000	0.0220000	0.0468426	0.0334000	0.060000	0.400	0.0466436
PCVolume	39	0.0793333	0.2391667	0.2771187	0.2713333	0.312000	0.478	0.0606953
PH	4	7.8800000	8.4400000	8.5456486	8.5400000	8.680000	9.360	0.1725162
PressureSetpoint	12	44.0000000	46.0000000	47.6153966	46.0000000	50.000000	52.000	2.0390474
PressureVacuum	0	-6.6000000	-5.6000000	-5.2161027	-5.4000000	-5.000000	-3.600	0.5699933
PSC	33	0.0020000	0.0480000	0.0845737	0.0760000	0.112000	0.270	0.0492690
PSCCO2	39	0.0000000	0.0200000	0.0564139	0.0400000	0.080000	0.240	0.0430387
PSCFill	23	0.0000000	0.1000000	0.1953689	0.1800000	0.260000	0.620	0.1177817
Temperature	14	63.6000000	65.2000000	65.9675401	65.6000000	66.400000	76.200	1.3827783
Usagecont	5	12.0800000	18.3600000	20.9929618	21.7900000	23.755000	25.900	2.9779364

Since `Brand code` is categorical, creating summary statistics was not possible. Instead, we implemented table summary of the distribution of each value. Note that we are missing 120 values.

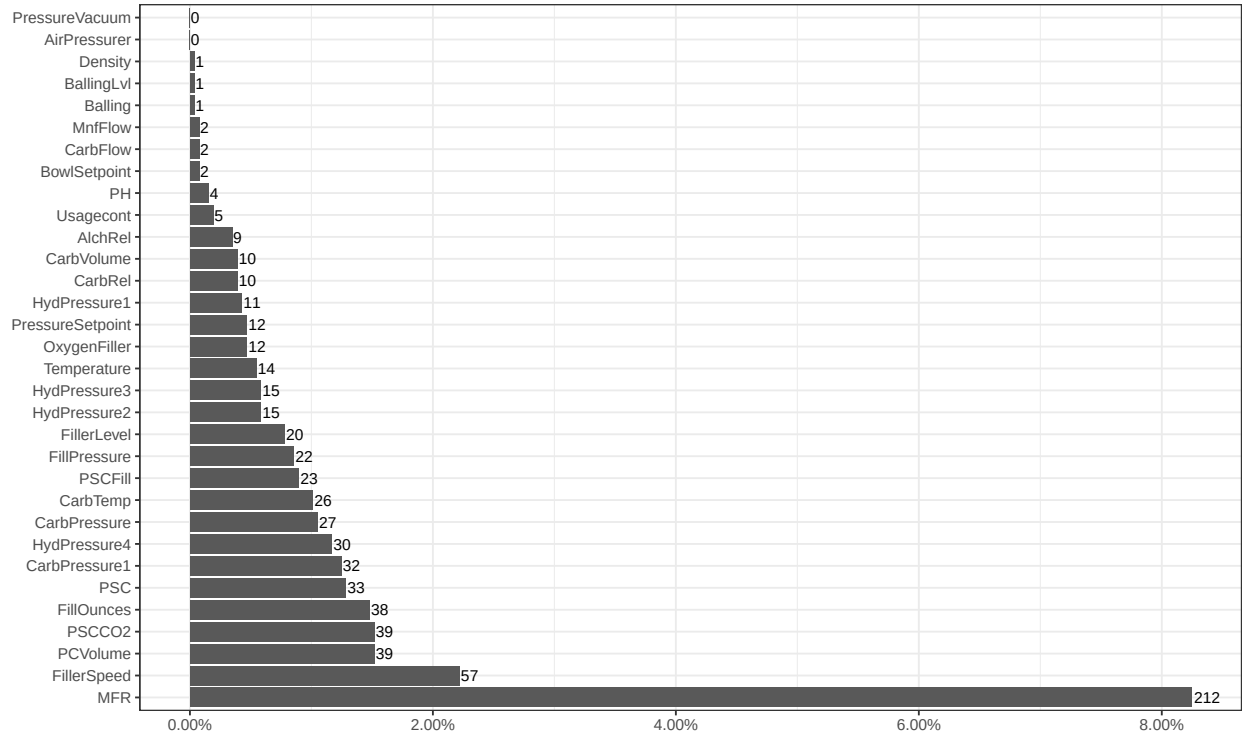
Table 1.3: Frequency distribution of BrandCode

Var1	Freq
A	293
B	1239
C	304
D	615
NA	120

Visualizations

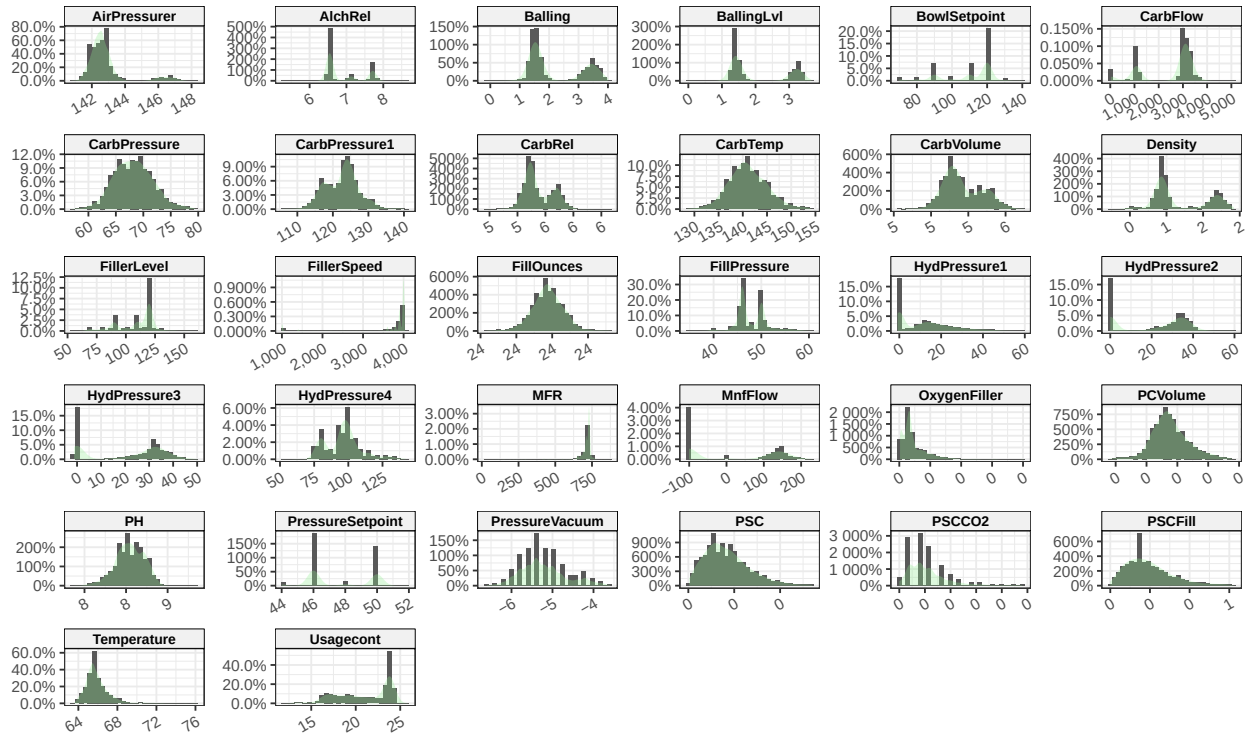
Missing data

With the help of visualization, it is easier to navigate how much each variable is missing. Note that `MFR` is the most missing variable.



Univariate distributions

We notice that there are some variables such as `Temperature` and `Oxygen Filler` that are highly positively skewed. There are potentials of presence of outliers for skewed variables.



As we expected, there are many outliers in `Temperature` and `Oxygen Filler`. Note that `MFR` greatly suffers from the presence of outliers.

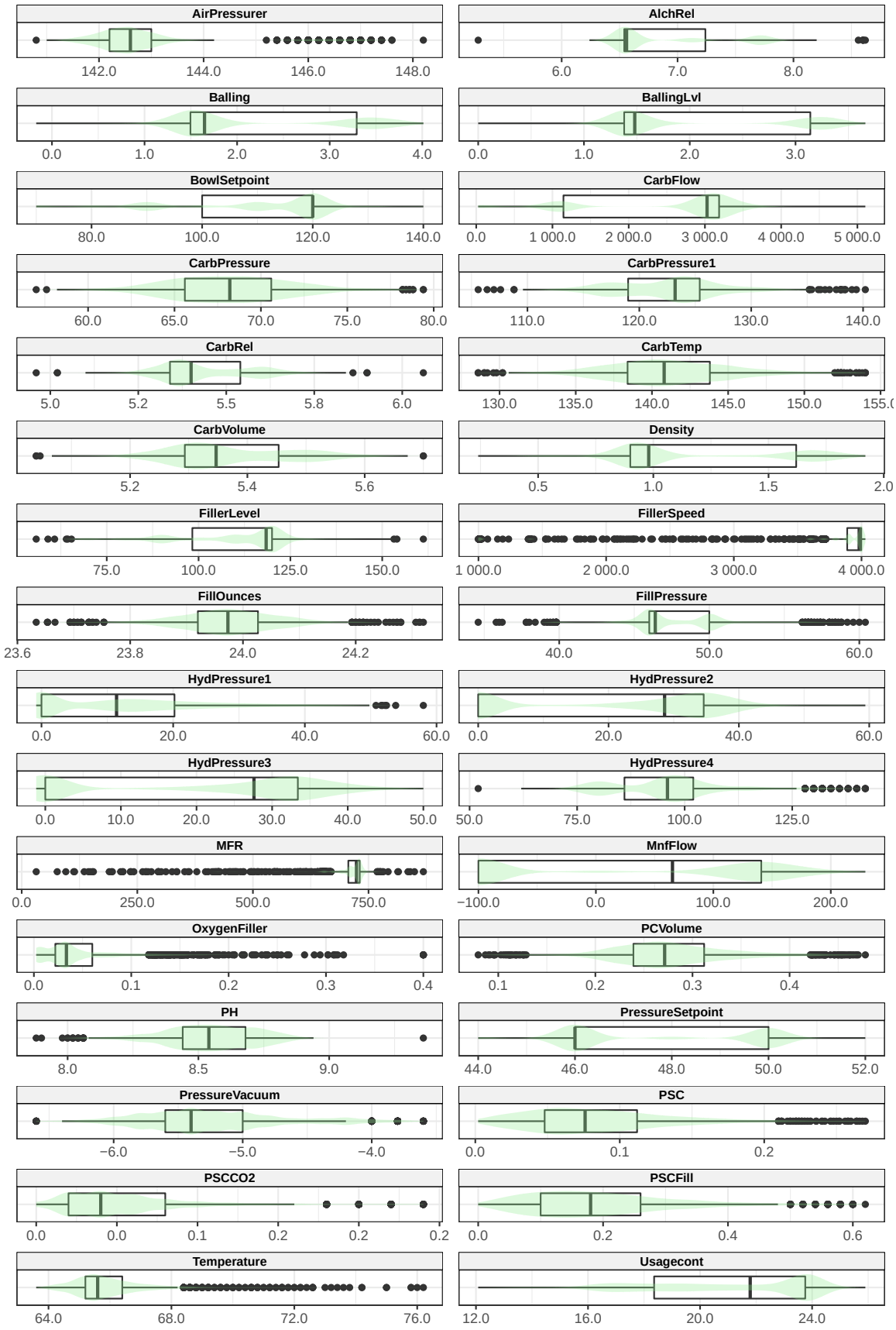
From the presence of multiple outliers, we then have to explain why/what/how we revise our initial model choice of `Bagged Tree` and apply necessary adjustment before applying `PLS`.

Ensembled model such as`Random Forest` is robust to outliers since only a subset of features are selected at random out of the total and the best split feature from the subset is used to split each node in a tree, unlike in bagging where all features are considered for splitting a node and this makes `Random Forest` a good altrenative for `Bagged Tree`.

Partial least squares regression (PLS regression) is used as an alternative for ordinary least squares regression in the presence of multicollinearity. This does not mean, however, `PLS` is robust to outliers. Thus, we will use Blocked Adaptive Computationally-Efficient Outlier Nominators (`BACON`) to eliminate outliers and then apply `PLS`.

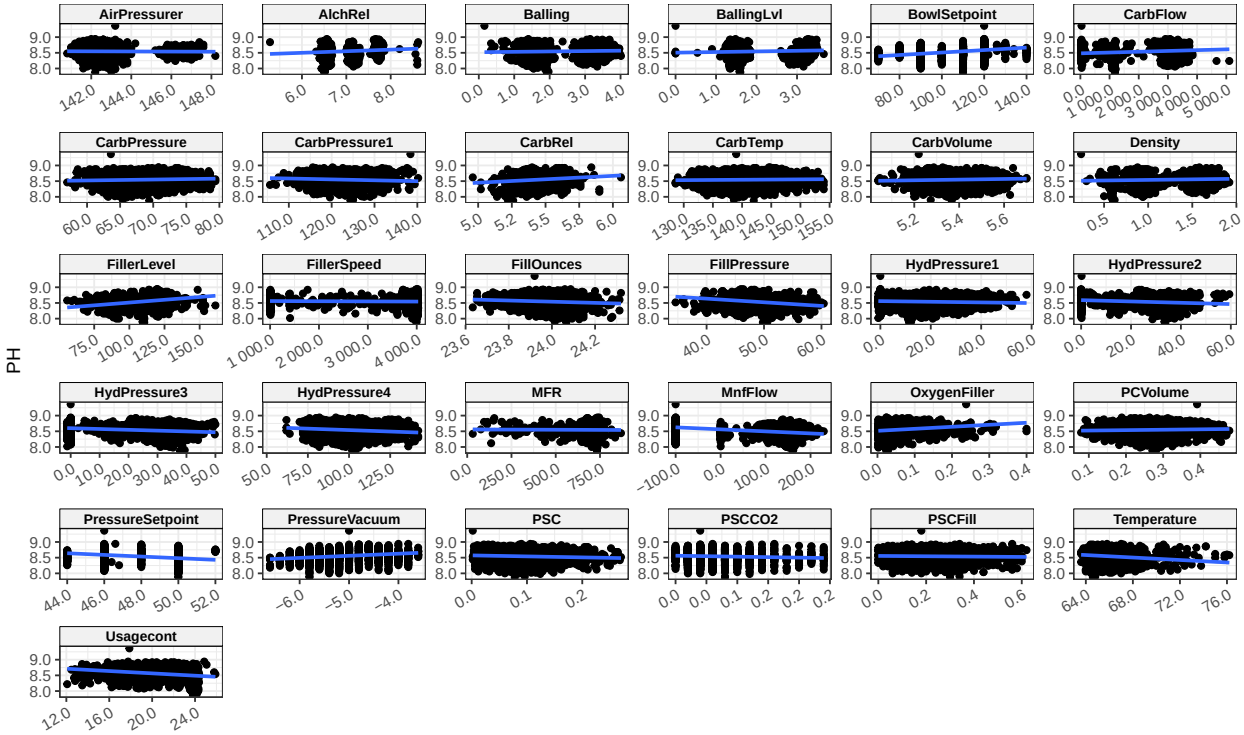
Reference:

(https://www.hindawi.com/journals/jam/2018/7696302/ - “PLS regression is sensitive to outliers and leverages. Thus several robust versions have been proposed in the literature, but only for linear PLS. Hubert [7] proposed two robust versions of the SIMPLS algorithm by using a robust estimation for the variance-covariance matrix. Kondylis and Hadi [8] used the BACON algorithm to eliminate outliers, resulting in a robust linear PLS.”)


AirPressurer
142.0 144.0 146.0 148.0
AlchRel
6.0 7.0 8.0
Balling
0.0 1.0 2.0 3.0 4.0
BallingLvl
0.0 1.0 2.0 3.0
BowlSetpoint
80.0 100.0 120.0 140.0
CarbFlow
0.0 1 000.0 2 000.0 3 000.0 4 000.0 5 000.0
CarbPressure
60.0 65.0 70.0 75.0 80.0
CarbPressure1
110.0 120.0 130.0 140.0
CarbRel
5.0 5.2 5.5 5.8 6.0
CarbTemp
130.0 135.0 140.0 145.0 150.0
CarbVolume
5.2 5.4 5.6
Density
0.5 1.0 1.5 2.0
FillerLevel
75.0 100.0 125.0 150.0
FillerSpeed
1 000.0 2 000.0 3 000.0 4 000.0
FillOunces
23.6 23.8 24.0 24.2
FillPressure
40.0 50.0 60.0
HydPressure1
0.0 20.0 40.0 60.0
HydPressure2
0.0 20.0 40.0 60.0
HydPressure3
0.0 10.0 20.0 30.0 40.0 50.0
HydPressure4
50.0 75.0 100.0 125.0
MFR
0.0 250.0 500.0 750.0
MnfFlow
−100.0 0.0 100.0 200.0
OxygenFiller
0.0 0.1 0.2 0.3 0.4
PCVolume
0.1 0.2 0.3 0.4
PH
8.0 8.5 9.0
PressureSetpoint
44.0 46.0 48.0 50.0 52.0
PressureVacuum
−6.0 −5.0 −4.0
PSC
0.0 0.1 0.2
PSCCO2
0.0 0.0 0.1 0.2 0.2 0.2
PSCFill
0.0 0.2 0.4 0.6
Temperature
64.0 68.0 72.0 76.0
Usagecont
12.0 16.0 20.0 24.0

Bivariate relationships

Note that when `Oxygen Filler` has fairly positive relationship with `PH`, `Temparature` has negative relationship. We confirmed the relationships between predictors and target variable.

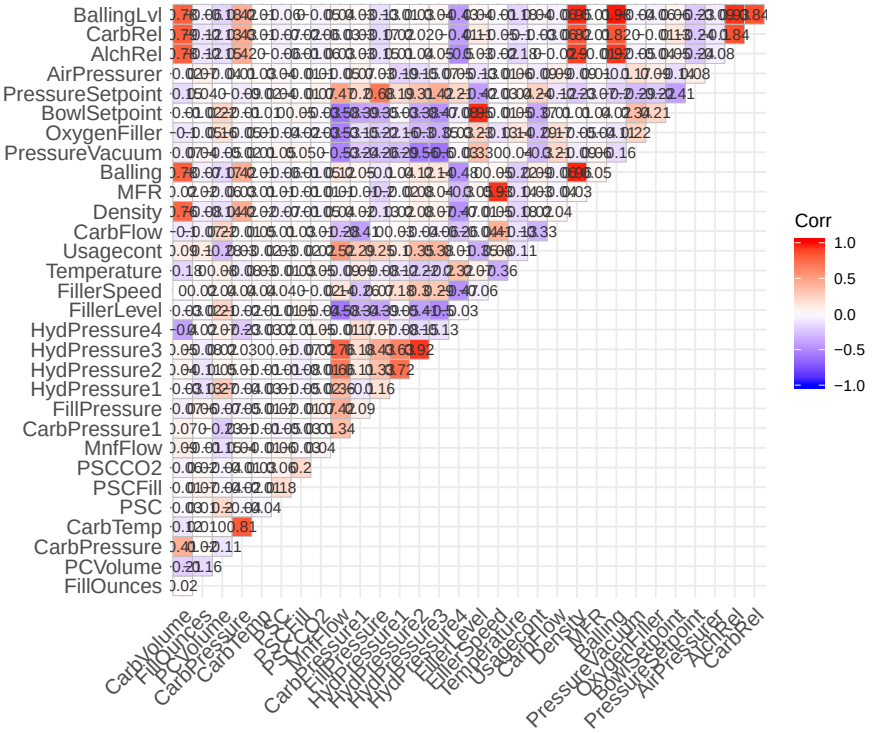


Correlation Matrix

We confirmed that there are several highly correlated predictors. This is another reason why `Random Forest` would be more preferred to `Bagged Tree` model. In fact, `Random Forest` is more robust to strong correlation than `Bagged Tree`. Although multicollinearity is not a big problem for for `PLS` and `Random Forest`, it can cause some problem when it comes to inferring the importance of certain predictors in tree model. Multicollinearity means that some predictors are shown as highly correlated with other combincations of predictors in variable importance. According to the reference, it may mislead business audience to think that some features are not more important than others vice-versa. However, we really do not know whether the reader of this paper would still treat highly correlated variable such as `HydPressure3` different from `HydPressure2` or not. Thus, we will just keep the variables as they are.

Reference:

(“For example, the two surface area predictors have an extremely high correlation (0.96) and each is used in the tree shown in Fig. 8.4. It is possible that the small difference between these predictors is strongly driving the choice between the two, but it is more likely to be due to small, random differences in the variables. Because of this, more predictors may be selected than actually needed. In addition, the variable importance values are affected. If the solubility data only contained one of the surface area predictors, then this predictor would have likely been used twice in the tree, therefore inflating its importance value. Instead, including both surface area predictors in the data causes their importance to have only moderate values.” Page 181 of Applied Predictive Modeling - Max KJ)



2 Data preparation

Before modeling can be done, the issues identified during the data exploration, namely, zero variance predictors, outliers and missing data need to be addressed. For `BACON` (outlier handling method) approach, categorical variable needs to be converted to numeric.

Imputation

After reviewing a few methods of multiple imputation techniques, Multiple Imputation Chained Equations (MICE) was selected for its strength in handling imputation for observations with more than one predictor missing. For categorical variable such as `Brand Code` will be imputed by mode. For `BACON` to work, we will also convert `Brand Code` to numeric.

We will also remove zero variance Predictors using `nearZeroVar`. It diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that are have both of the following characteristics: they have 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 is large.

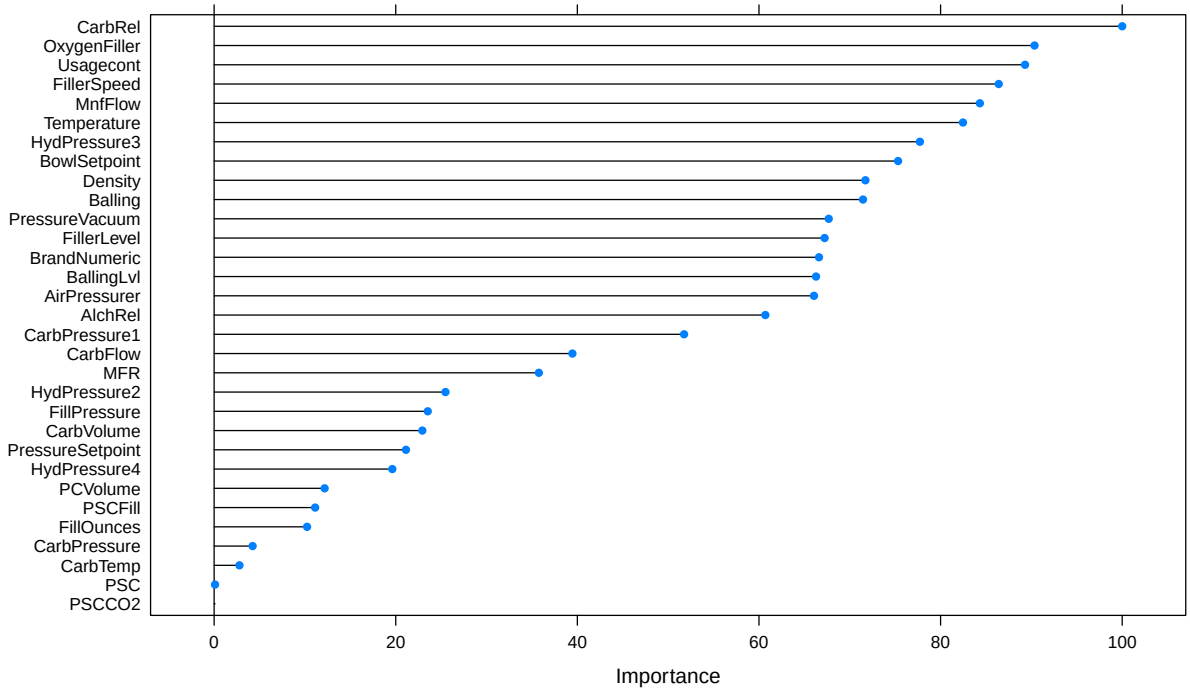
For outlier handling, `BACON` is used. `BACON`, short for 'Blocked Adaptive Computationally-Efficient Outlier Nominators', is a somewhat robust algorithm (set), with an implementation for regression or multivariate covariance estimation. The function produces index with `TRUE` or `FALSE` to identify rows with outliers. This index will be used as `subset` to train outlier free train set in the process of hyper-parameter tuning in `train` function and train the final model using full train set without removing outliers after being applied with best hyper-parameters.

3 Modeling

Model 1: Bagged Tree with BACON

Table 3.1: Model Summary - Bagged Tree with BACON

RMSE	Rsquared	MAPE
0.1198618	0.4922456	0.0106783

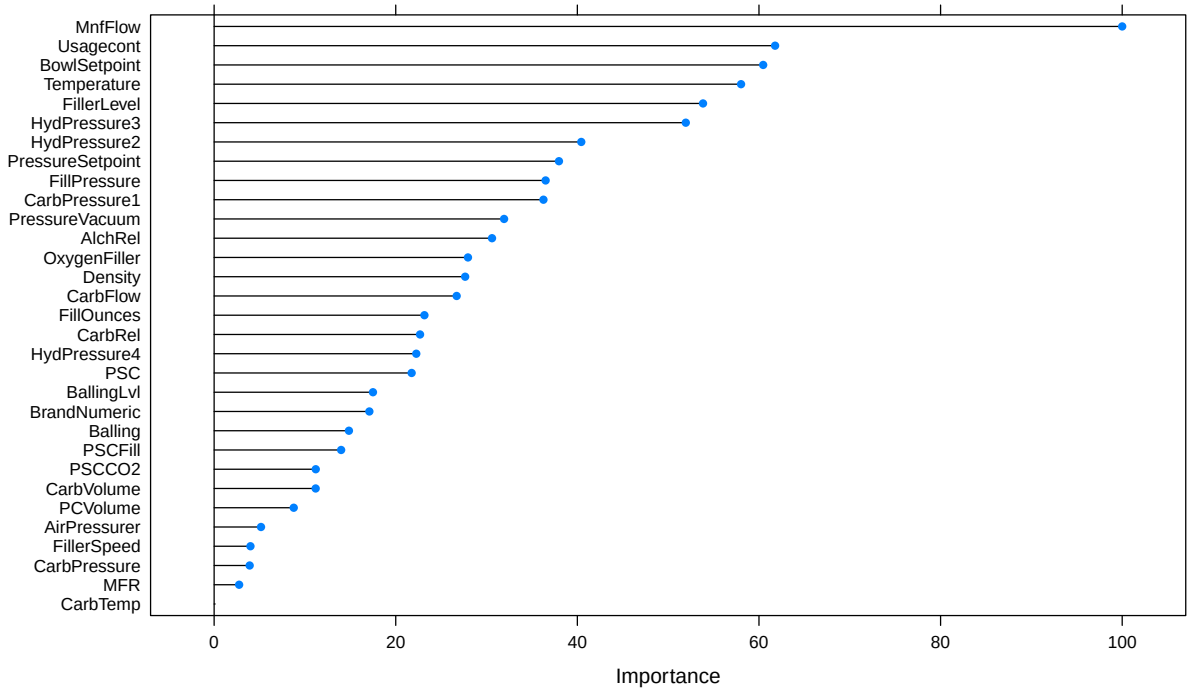


MAPE is 0.0106783 where as top 3 important predictors are `CarbRel`, `OxygenFiler` and `Usuagecont`. Since `Bagged Tree` uses all features for splitting a node Unlike `Random Forest` which uses only a subset of features at random out of the total for splitting each node in a tree, the order of feature importances between two models can be quite different.

Model 2: PLS with BACON

Table 3.2: Model Summary - PLS with BACON

	RMSE	Rsquared	MAPE
10	0.1327207	0.3740444	0.0123968



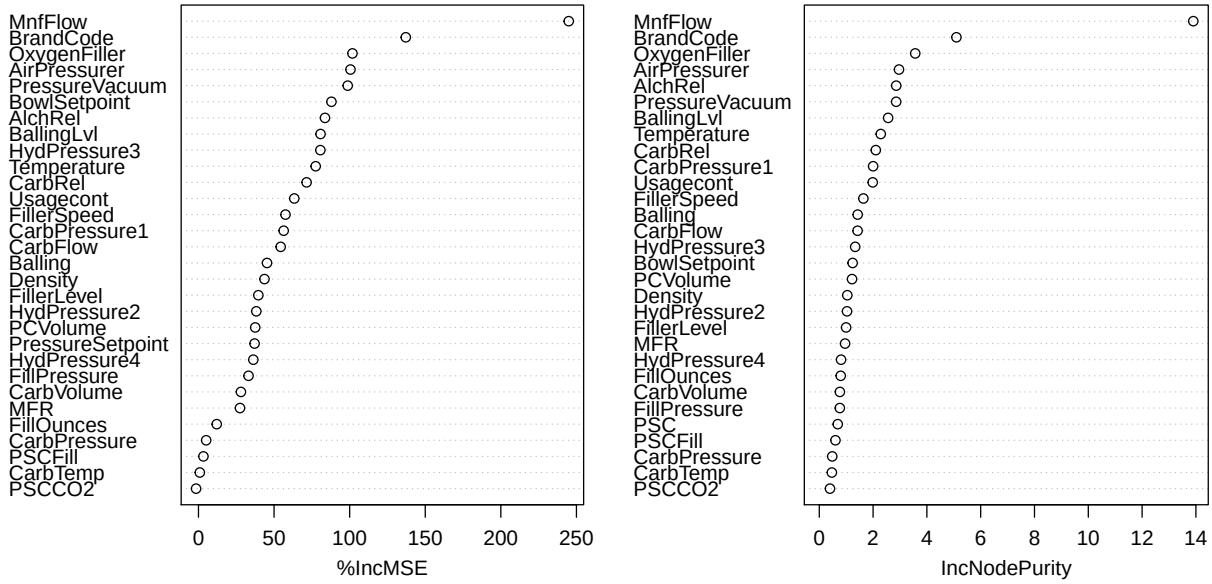
The final value used for the model was ncomp = 10. MAPE is 0.0123968, 0.0123968, 0.0123968, 0.0123968, 0.0123968, 0.0123968, 0.0123968, 0.0123968, 0.0123968, 0.0123968, 0.0123968 where as top 3 important predictors are `MnfFlow`, `Usuagecont` and `BowlSetpoint`. The order of variable importance, again, is quite different from `Bagged Tree` as PLS is not a tree-based model. Note that PLS is a dimension reduction technique with some similarity to principal component analysis. The predictor variables are mapped to a smaller set of variables and within that smaller space we perform a regression against the outcome variable. In contrast to principal component analysis where the dimension reduction ignores the outcome variable, the PLS procedure aims to choose new mapped variables that maximally explain the outcome variable.

Model 3: Random Forest with original data set

Table 3.3: Model Summary - RF with original data set

RMSE	Rsquared	MAPE
0.0969916	0.6909936	0.0081279

rf_model2



The final value used for the model was ncomp = 10. MAPE is 0.0081279 where as top 3 important predictors are `MnfFlow`, `BrandCode` and `PressureVacuum` for %incMSE and `MnfFlow`, `BrandCode` and `OxygenFiller` for IncNodePurity. Unlike `PLS`, `Random Forest` can produce 2 different variable importance plots.

The first graph shows that if a variable is assigned values by random permutation by how much will the MSE increase. The second plot is based on `node purity` which is measured by `Gini Index` and it is the the difference between RSS before and after the split on that variable. In short, each graph shows how much MSE or Impurity increases when each variable is randomly permuted.

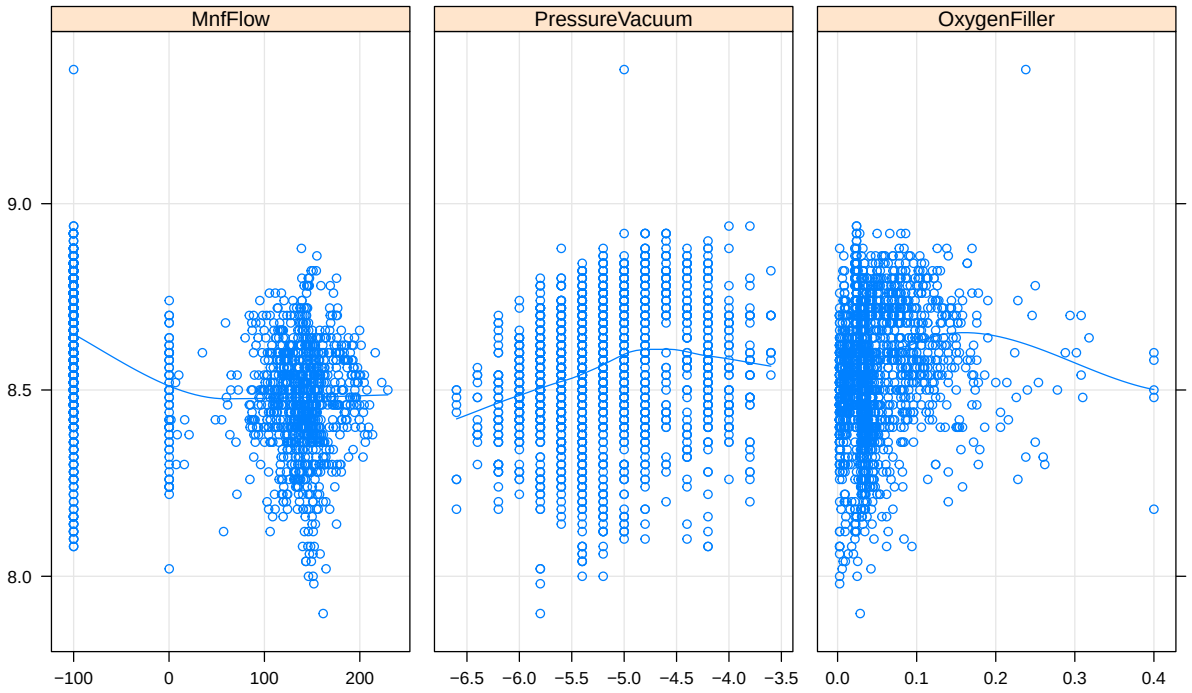
4 Evaluation

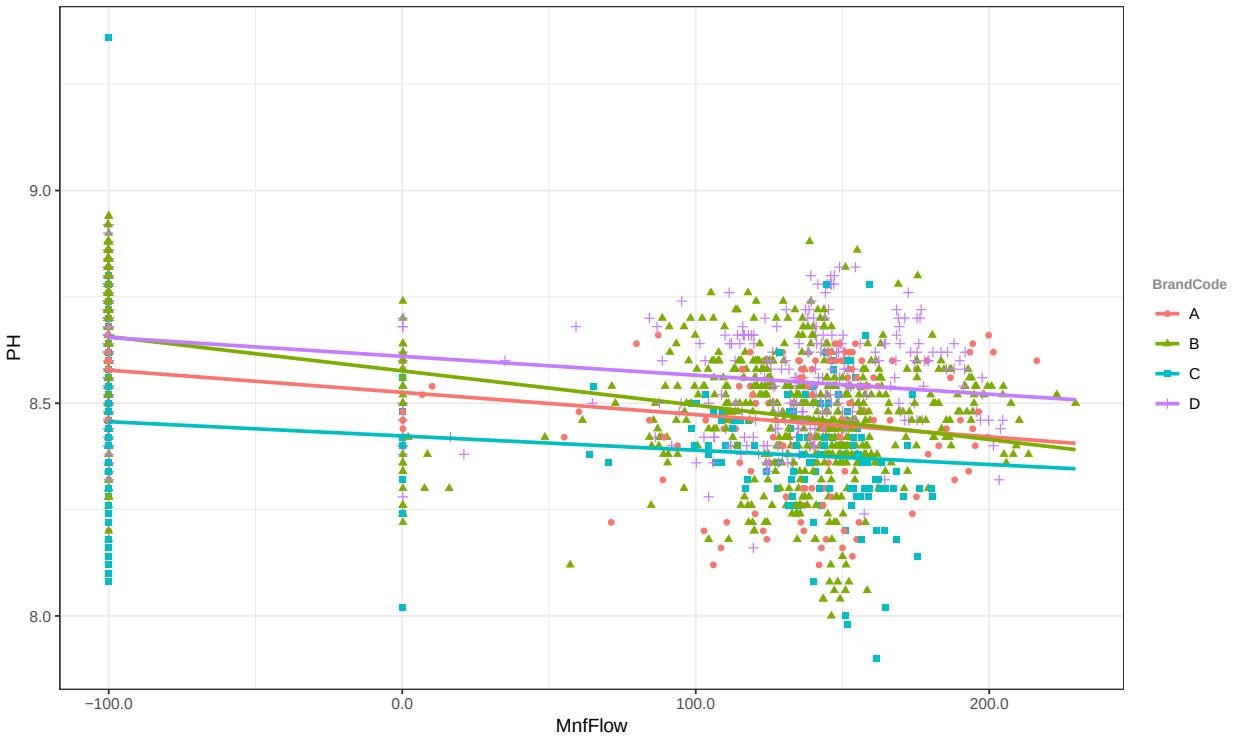
Table 4.1: Evaluation Summary on test set

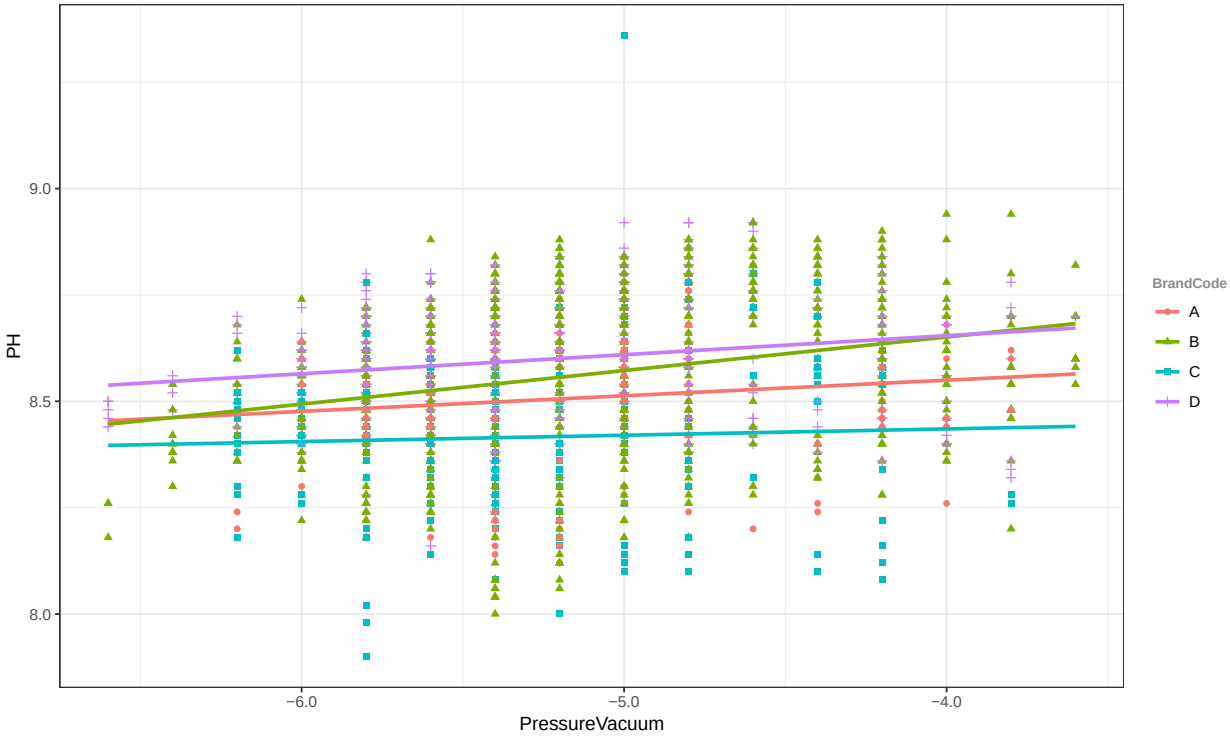
MODEL	RMSE	Rsquare	MAPE
Bagged Tree	0.1235554	0.5047507	0.0111408
PLS - BACON	0.1358493	0.3862047	0.0122273
RF	0.0883145	0.7449339	0.0075050

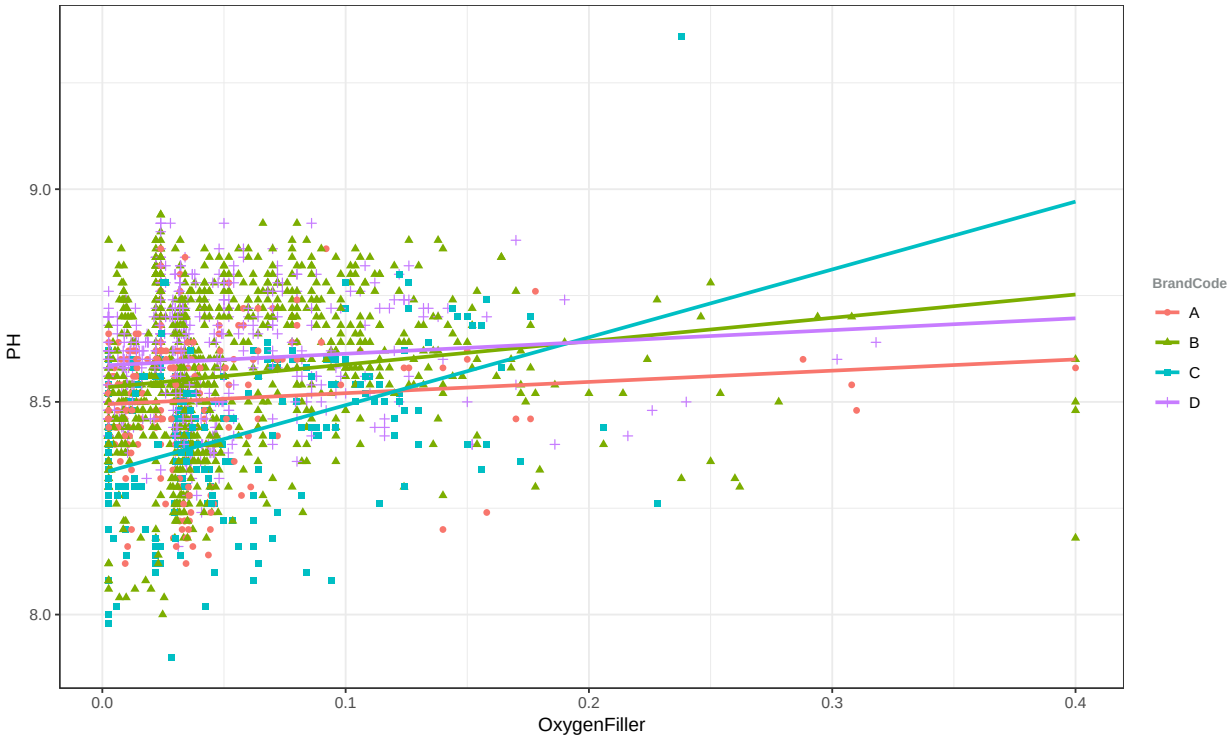
From the table, we confirmed that `Random Forest` is a clear winner with the lowest MAPE on test set.

Insight & Conclusion









We graphed top 4 most important variables from `varImp` of MSE and NodePurity. Given that `Brand Code` is categorical, we grouped 3 continous variables into `Brand Code`.

The first feature plot shows that 3 continous variables have very weak relationship with `PH`. `MnfFlow` has slight negative relation-

ship where as PressureVacuum and OxygenFiller have slight positive relationship.

From grouping by Brand Code, we see that for PH vs MnfFlow and PH vs PressureVacuum, it seems like predictors have the most negative relationship with PH with code = B. For PH vs OxygenFiller, code = C seems to have the most negative relationship with PH.

We can recommend business users to put more attention on increasing MnfFlow or decreasing PressureVacuum (especially code = B for both) and OxygenFiller (especially code = C) than any other predictors if their goal is to decrease PH.

Prediction

Let's export the prediction values of Random Forest (the best model) on StudentEvaluation as CSV file.

Appendix

```
library(tidyverse)
library(readxl)
library(psych)
library(ggplot2)
library(mice)
library(xtable)
library(GGally)
library(ggstance)
library(grid)
library(gridExtra)
library(ggpubr)
library(caret)
library(data.table)
library(recipes)
library(Metrics)
library(randomForest)
library(robustX)        # BACON
library(ggcorrplot)     # Vis corr matrix
library(e1071)          # Misc stats functions

df <- read_excel('C:/Users/ahwang/Desktop/Cuny/DATA624/project2/data/StudentData.xlsx')
df_eval <- read_excel('C:/Users/ahwang/Desktop/Cuny/DATA624/project2/data/StudentEvaluation.xlsx')
dict <- read_excel('C:/Users/ahwang/Desktop/Cuny/DATA624/project2/data/DataDictionary.xlsx')

# remove space in-between variable names
colnames(df) <- gsub(" ","",colnames(df))

# Introduction {-#intro}

# Data Exploration
## Data dictionary

kable(dict, caption="Data dictionary", booktabs=T)%>%kable_styling()%>%row_spec()

## Summary statistics

# Create a table summarizing the training data
```

```r
# create lists of desired summary stats for calculation
statFuns <-
  funs(missing = sum(is.na(.))
       , min = min(., na.rm = TRUE)
       , Q1 = quantile(., .25, na.rm = TRUE)
       , mean = mean(., na.rm = TRUE)
       , median = median(., na.rm = TRUE)
       , Q3 = quantile(., .75, na.rm = TRUE)
       , max = max(., na.rm = TRUE)
       , sd = sd(., na.rm = TRUE)
       , mad = mad(., na.rm = TRUE)
       , skewness = skewness(., na.rm = TRUE)
       , kurtosis = kurtosis(., na.rm = TRUE)
  )

# Create data frame of basic summary stats
dfSumTrain  <-
  df %>%
  # union(dfEval %>% mutate(TARGET_WINS = as.numeric(NA))) %>%
  dplyr::select(-`BrandCode`) %>%
  summarise_all(statFuns) %>%
  gather() %>%
  separate(key, c('metric', 'stat'), sep = '(_)(?!.*_)') %>%
  spread(stat, value) %>%
  dplyr::select(metric, names(statFuns))

dfSumTrain %>% kable(caption="Summary statistics", booktabs=T)%>%kable_styling()%>%row_spec()

table(df$`BrandCode`, useNA = "ifany")%>%kable(caption="Frequency distribution of BrandCode", booktabs="

## Visualizations

### Missing data

dfSumTrain %>%
  group_by(metric) %>%
  mutate(miss_perc = missing / !!nrow(df)) %>%
  dplyr::select(metric,missing, miss_perc) %>%
  ggplot(data = ., aes(x = reorder(metric, -miss_perc) , y = miss_perc)) +
  geom_bar(stat = 'identity') +
  coord_flip() +
  geom_text(aes(label = missing), hjust = -0.1, size = 3) +
  labs(x = NULL, y = NULL, Title = '% Missing') +
  theme_bw() +
  theme(legend.position = 'none') +
  scale_y_continuous(labels = scales::percent)

### Univariate distributions

df %>%
  # union(dfEval %>% mutate(TARGET_WINS = as.numeric(NA))) %>%
  dplyr::select(-BrandCode) %>%
  gather() %>%
```

```
  group_by(key) %>%
  ggplot(data = ., aes(value)) +
  geom_histogram(bins = 30, aes(y = ..density..)) +
  geom_density(alpha = 0.3, color = NA, fill = 'lightgreen') +
  scale_x_continuous(labels = scales::comma) +
  scale_y_continuous(labels = scales::percent) +
  facet_wrap(~key, scales = 'free') +
  labs(x = NULL, y = NULL) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))

df %>%
  # union(dfEval %>% mutate(TARGET_WINS = as.numeric(NA))) %>%
  dplyr::select(-BrandCode) %>%
  gather() %>%
  group_by(key) %>%
  ggplot(data = ., aes(x = '', y = value)) +
  geom_boxplot() +
  geom_violin(alpha = 0.3, color = NA, fill = 'lightgreen') +
  labs(x = NULL, y = NULL) +
  theme_bw() +
  theme(axis.ticks.y=element_blank()) +
  facet_wrap(~key, scales = 'free', ncol = 2) +
  coord_flip()

### Bivariate relationships

df %>%
  dplyr::select(-BrandCode) %>%
  gather(key, value, -PH) %>%
  group_by(key) %>%
  ggplot(data = ., aes(x = value, y = PH)) +
  geom_point() +
  geom_smooth(method = 'gam') +
  facet_wrap(~key, scales = 'free') +
  labs(x = NULL) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))

### Correlation Matrix

# Calculate pairwise pearson correlation and display as upper matrix plot
df %>%
  # union(dfEval %>% mutate(TARGET_WINS = as.numeric(NA))) %>%
  dplyr::select(-c('BrandCode','PH')) %>%
  cor(method = 'pearson', use = 'pairwise.complete.obs') %>%
  ggcorrplot(corr = ., method = 'square', type = 'upper'
             , lab = TRUE, lab_size = 3, lab_col = 'grey20')

# Data preparation
## Imputation

# set seed for split to allow for reproducibility
```

```r
set.seed(58677)

# use mice w/ default settings to impute missing data
miceImput <- mice(df, printFlag = FALSE)

# add imputed data to original data set
df_mice <- complete(miceImput)
df_mice$BrandCode[is.na(df_mice$BrandCode)] <- 'B'
#table(df_mice$BrandCode, useNA = "ifany")

# Look for any features with no variance:
zero_cols <- nearZeroVar( df_mice )
df_final <- df_mice[,-zero_cols] # drop these zero variance columns
df_final$BrandCode <- as.factor(df_final$BrandCode)

# convert categorical factor into numeric
M <- df_final
must_convert<-sapply(M,is.factor) # logical vector telling if a variable needs to be displayed as numer
BrandNumeric <- sapply(M[,must_convert],unclass)    # data.frame of all categorical variables now displ
df_final2<-cbind(M[,!must_convert],BrandNumeric)        # complete data.frame with all variables put to

# split data train/test
# df for random forest
training <- df_final$PH %>%createDataPartition(p = 0.8, list = FALSE)
df_train <- df_final[training, ]
df_test <- df_final[-training, ]

# df for PLS and Bagging
training2 <- df_final2$PH %>%createDataPartition(p = 0.8, list = FALSE)
df_train2 <- df_final2[training2, ]
df_test2 <- df_final2[-training2, ]

# X and y split for BACON fit
x <- subset(df_train2, select = -c(PH) )
x <- as.matrix(x)
y <- df_train2[, c('PH')]

bacon_fit <- BACON(x = x, y = y)

# Modeling

## Model 1: Bagged Tree with BACON

set.seed(58677)
bagged_model <- train( PH~., data = df_train2, method="treebag",
                       tuneLength=10,
                       subset = bacon_fit$subset,
                       trControl=trainControl(method="cv",number=5) )

# create MAPE table
train_bag_pred <- predict(bagged_model)
bagged_model$results$MAPE <- Metrics::mape(df_train2$PH, train_bag_pred)
```

```
bagged_model$results[,c(2,3,8)]%>%kable(caption="Model Summary - Bagged Tree with BACON", booktabs=T)%>%

# plot varImp
plot(varImp(bagged_model))

## Model 2: PLS with BACON

set.seed(58677)
df_final2$BrandNumeric <- as.factor(df_final2$BrandNumeric)

pls_model <- train( PH~., data = df_train2, method="pls",
                    tuneLength=10,
                    subset = bacon_fit$subset,
                    preProcess=c("center","scale"), trControl=trainControl(method="cv",number=5) )

# create MAPE table
train_pls_pred <- predict(pls_model)
pls_model$results$MAPE <- Metrics::mape(df_train2$PH, train_pls_pred)

pls_model$results[10,c(2,3,8)]%>%kable(caption="Model Summary - PLS with BACON", booktabs=T)%>%kable_sty

# plot varImp
plot(varImp(pls_model))

## Model 3-1: Random Forest with BACON

# set.seed(58677)
# rf_model <- train( PH~., data = df_train2, method="rf",
#                     tuneLength=10,
#                     subset = bacon_fit$subset,
#                     importance = TRUE,
#                     trControl=trainControl(method="cv",number=5) )
#
# # create MAPE table
# train_rf_pred <- predict(rf_model)
# rf_model$results$MAPE <- Metrics::mape(df_train2$PH, train_rf_pred)
#
# rf_model$results%>%kable(caption="Model Summary - RF with BACON", booktabs=T)%>%kable_styling()%>%row
#
# # plot varImp
# plot(varImp(rf_model))
#
# Although `BACON` is not needed for `Random Forest`, it would not bother using it anyway. We will expe

## Model 3: Random Forest with original data set

set.seed(58677)

# Algorithm Tune (tuneRF)
#bestmtry <- tuneRF(df_train[, -25], df_train[,25], stepFactor=1.5, improve=1e-5, ntree=2500)

##mtry <- ( (ncol(df_train) -1) / 3 ) or sqrt(ncol(df_train) - 1) # By default, # of predictors / 3 for
```

```r
# from above result, we got mtry= 27 and ntree=2500 as optimal parameters
rf_model2 <- randomForest(PH~., data=df_train, method="rf", mtry= 31, importance = TRUE, ntree = 2500)

# create MAPE table
train_rf_pred2 <- predict(rf_model2)

s <- data.frame(
  RMSE = Metrics::rmse(df_train$PH, train_rf_pred2),
  Rsquared = caret::R2(df_train$PH, train_rf_pred2),
  MAPE = Metrics::mape(df_train$PH, train_rf_pred2) )

s%>%kable(caption="Model Summary - RF with original data set", booktabs=T)%>%kable_styling()%>%row_spec

# plot varImp
varImpPlot(rf_model2)

# Evaluation

# Make predictions
p1 <- bagged_model %>% predict(df_test2)
p2 <- pls_model %>% predict(df_test2)
p3 <- rf_model2 %>% predict(df_test)

# Model performance metrics
sum_t <- data.frame(
  MODEL = c('Bagged Tree',
            'PLS - BACON',
            'RF'),
  RMSE = c(caret::RMSE(p1, df_test2$PH),
           caret::RMSE(p2, df_test2$PH),
           caret::RMSE(p3, df_test$PH) ),
  Rsquare = c(caret::R2(p1, df_test2$PH),
              caret::R2(p2, df_test2$PH),
              caret::R2(p3, df_test$PH)),
  MAPE = c(Metrics::mape(p1, df_test2$PH),
           Metrics::mape(p2, df_test2$PH),
           Metrics::mape(p3, df_test$PH))
)

sum_t%>%kable(caption="Evaluation Summary on test set", booktabs=T)%>%kable_styling()%>%row_spec()

## Insight & Conclusion

# code
top_var <- c('MnfFlow','PressureVacuum', 'OxygenFiller')

featurePlot(df_train[, top_var],
            df_train$PH,
            plot = "scatter",
            between = list(x = 1, y = 1),
            type = c("g", "p", "smooth"),
            layout = c(3,1),
            labels = rep("", 2))
```

```
# ggplot(df_train[df_train$BrandCode == 'B',], aes(x=MnfFlow, y=PH, shape=BrandCode, color=BrandCode)) +
#   geom_point()
#
# ggplot(df_train[df_train$BrandCode == 'A',], aes(x=MnfFlow, y=PH, shape=BrandCode, color=BrandCode)) +
#   geom_point()
#
# ggplot(df_train[df_train$BrandCode == 'C',], aes(x=MnfFlow, y=PH, shape=BrandCode, color=BrandCode)) +
#   geom_point()
#
# ggplot(df_train[df_train$BrandCode == 'D',], aes(x=MnfFlow, y=PH, shape=BrandCode, color=BrandCode)) +
#   geom_point()
ggplot(df_train, aes(x=MnfFlow, y=PH, color=BrandCode, shape=BrandCode)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE) +
  theme_bw()+theme()

ggplot(df_train, aes(x=PressureVacuum, y=PH, color=BrandCode, shape=BrandCode)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE) +
  theme_bw()+theme()

ggplot(df_train, aes(x=OxygenFiller, y=PH, color=BrandCode, shape=BrandCode)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE) +
  theme_bw()+theme()

## Prediction

# remove space in-between variable names
colnames(df_eval) <- gsub(" ","",colnames(df_eval))
# remove column with zero-variance
set.seed(58677)

# use mice w/ default settings to impute missing data
miceImput2 <- mice(df_eval, printFlag = FALSE)

# add imputed data to original data set
df_mice2 <- complete(miceImput2)
#table(df_eval$BrandCode, useNA = 'ifany')
df_mice2$BrandCode[is.na(df_mice2$BrandCode)] <- 'B'
#table(df_mice$BrandCode, useNA = "ifany")

# Look for any features with no variance:
#zero_cols <- nearZeroVar( df_mice2 )
df_final22 <- df_mice2[,-zero_cols] # drop these zero variance columns
df_final22$BrandCode <- as.factor(df_final22$BrandCode)

df_eval2 <- subset(df_eval, select = -PH)

pred_eval <- predict(rf_model2, subset(df_final22))
write.csv(pred_eval, 'prediction.csv')
```