

DATA 624 Project 2

Ron Balaban, Brandon Chung, Yanyi Li, Chi Hang(Philip) Cheung, Jiaxin Zheng

2025-03-13

```
library(caret)
library(dplyr)
library(e1071)
library(forecast)
library(feasts)
library(ggplot2)
library(ggfortify)
library(GGally)
library(knitr)
library(kernlab)
library(lubridate)
library(latex2exp)
library(missForest)
library(mice)
library(readxl)
library(readr)
library(tsibble)
library(tidyr)
library(VIM)
library(writexl)
library(tibble)
library(corrplot)
```

Instructions This is role playing. I am your new boss. I am in charge of production at ABC Beverage, and you are a team of data scientists reporting to me. My leadership has told me that new regulations require us to understand our manufacturing process, the predictive factors, and be able to report to them our predictive model of PH.

Please use the historical data set I am providing. Build and report the factors in BOTH a technical and non-technical report. I like to use Word and Excel. Please provide your non-technical report in a business-friendly, readable document and your predictions in an Excel-readable format. The technical report should show clearly the models you tested and how you selected your final approach.

Please submit both Rpubs links and .rmd files or other readable formats for technical and non-technical reports. Also submit the Excel file showing the prediction of your models for pH.

— Starts Here —

To load the training and test datasets. I have uploaded the two files and converted them to CSV files in my repo, and made the links available to load:

Loading data:

```
train_og<- read.csv('https://raw.githubusercontent.com/stormwhale/data-mines/refs/heads/main/StudentData/train_og.csv')
new_test_data<- read.csv('https://raw.githubusercontent.com/stormwhale/data-mines/refs/heads/main/StudentData/new_test_data.csv')
```

Data Exploration

```
# Summary statistics
summary(train_og)
```

```
##   Brand.Code      Carb.Volume     Fill.Ounces      PC.Volume
##   Length:2571      Min.    :5.040      Min.    :23.63      Min.    :0.07933
##   Class  :character 1st Qu.:5.293    1st Qu.:23.92    1st Qu.:0.23917
##   Mode   :character Median  :5.347    Median  :23.97    Median  :0.27133
##                           Mean    :5.370    Mean    :23.97    Mean    :0.27712
##                           3rd Qu.:5.453    3rd Qu.:24.03    3rd Qu.:0.31200
##                           Max.    :5.700    Max.    :24.32    Max.    :0.47800
##                           NA's    :10       NA's    :38       NA's    :39
##   Carb.Pressure     Carb.Temp        PSC          PSC.Fill
##   Min.    :57.00     Min.    :128.6     Min.    :0.00200    Min.    :0.00000
##   1st Qu.:65.60     1st Qu.:138.4     1st Qu.:0.04800    1st Qu.:0.1000
##   Median  :68.20     Median  :140.8     Median  :0.07600    Median  :0.1800
##   Mean    :68.19     Mean    :141.1     Mean    :0.08457    Mean    :0.1954
##   3rd Qu.:70.60     3rd Qu.:143.8     3rd Qu.:0.11200    3rd Qu.:0.2600
##   Max.    :79.40     Max.    :154.0     Max.    :0.27000    Max.    :0.6200
##   NA's    :27       NA's    :26       NA's    :33       NA's    :23
##   PSC.CO2          Mnf.Flow        Carb.Pressure1  Fill.Pressure
##   Min.    :0.00000    Min.    :-100.20    Min.    :105.6     Min.    :34.60
##   1st Qu.:0.02000    1st Qu.:-100.00    1st Qu.:119.0    1st Qu.:46.00
##   Median  :0.04000    Median  : 65.20     Median  :123.2     Median  :46.40
##   Mean    :0.05641    Mean    : 24.57     Mean    :122.6     Mean    :47.92
##   3rd Qu.:0.08000    3rd Qu.: 140.80    3rd Qu.:125.4    3rd Qu.:50.00
##   Max.    :0.24000    Max.    : 229.40    Max.    :140.2     Max.    :60.40
##   NA's    :39       NA's    : 2       NA's    :32       NA's    :22
##   Hyd.Pressure1    Hyd.Pressure2   Hyd.Pressure3   Hyd.Pressure4
##   Min.    :-0.80      Min.    : 0.00      Min.    :-1.20      Min.    : 52.00
##   1st Qu.: 0.00      1st Qu.: 0.00      1st Qu.: 0.00      1st Qu.: 86.00
##   Median  :11.40      Median : 28.60      Median : 27.60      Median : 96.00
##   Mean    :12.44      Mean   : 20.96      Mean   : 20.46      Mean   : 96.29
##   3rd Qu.:20.20      3rd Qu.: 34.60      3rd Qu.: 33.40      3rd Qu.:102.00
##   Max.    :58.00      Max.    : 59.40      Max.    : 50.00      Max.    :142.00
##   NA's    :11       NA's    : 15       NA's    :15       NA's    :30
##   Filler.Level    Filler.Speed    Temperature     Usage.cont     Carb.Flow
##   Min.    : 55.8     Min.    : 998      Min.    :63.60      Min.    :12.08     Min.    : 26
##   1st Qu.: 98.3     1st Qu.: 3888    1st Qu.:65.20      1st Qu.:18.36     1st Qu.:1144
##   Median  :118.4     Median : 3982      Median :65.60      Median :21.79     Median :3028
```

```

##  Mean    :109.3   Mean    :3687   Mean    :65.97   Mean    :20.99   Mean    :2468
##  3rd Qu.:120.0   3rd Qu.:3998   3rd Qu.:66.40   3rd Qu.:23.75   3rd Qu.:3186
##  Max.    :161.2   Max.    :4030   Max.    :76.20   Max.    :25.90   Max.    :5104
##  NA's    :20      NA's    :57     NA's    :14      NA's    :5       NA's    :2
##  Density          MFR          Balling        Pressure.Vacuum
##  Min.   :0.240   Min.   : 31.4   Min.   :-0.170  Min.   :-6.600
##  1st Qu.:0.900   1st Qu.:706.3  1st Qu.: 1.496  1st Qu.:-5.600
##  Median  :0.980   Median :724.0   Median : 1.648   Median :-5.400
##  Mean    :1.174   Mean    :704.0   Mean    : 2.198  Mean    :-5.216
##  3rd Qu.:1.620   3rd Qu.:731.0  3rd Qu.: 3.292  3rd Qu.:-5.000
##  Max.    :1.920   Max.    :868.6   Max.    : 4.012  Max.    :-3.600
##  NA's    :1      NA's    :212    NA's    : 1
##  PH            Oxygen.Filler  Bowl.Setpoint  Pressure.Setpoint
##  Min.   :7.880   Min.   :0.00240  Min.   : 70.0   Min.   :44.00
##  1st Qu.:8.440   1st Qu.:0.02200 1st Qu.:100.0  1st Qu.:46.00
##  Median  :8.540   Median :0.03340  Median :120.0  Median :46.00
##  Mean    :8.546   Mean    :0.04684  Mean    :109.3  Mean    :47.62
##  3rd Qu.:8.680   3rd Qu.:0.06000 3rd Qu.:120.0  3rd Qu.:50.00
##  Max.    :9.360   Max.    :0.40000  Max.    :140.0  Max.    :52.00
##  NA's    :4      NA's    :12     NA's    : 2      NA's    :12
##  Air.Pressurer  Alch.Rel     Carb.Rel     Balling.Lvl
##  Min.   :140.8   Min.   : 5.280  Min.   : 4.960  Min.   :0.00
##  1st Qu.:142.2   1st Qu.: 6.540  1st Qu.: 5.340  1st Qu.:1.38
##  Median  :142.6   Median : 6.560  Median : 5.400  Median :1.48
##  Mean    :142.8   Mean    : 6.897  Mean    : 5.437  Mean    :2.05
##  3rd Qu.:143.0   3rd Qu.: 7.240  3rd Qu.: 5.540  3rd Qu.:3.14
##  Max.    :148.2   Max.    : 8.620  Max.    : 6.060  Max.    :3.66
##  NA's    :9      NA's    :10     NA's    : 1

```

From our data summary we can note that there are null values in most variables, these will be handled in the data pre-processing portion.

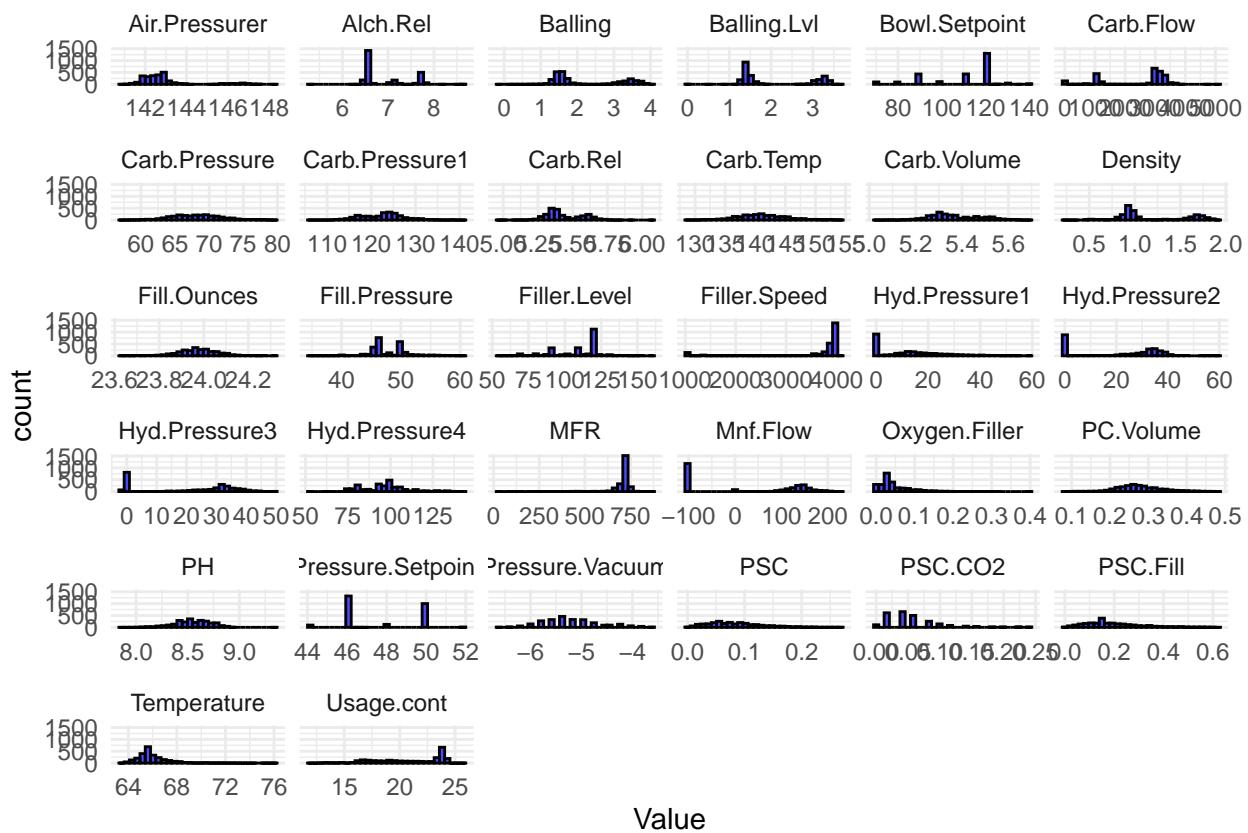
```

# Visualizing predictor and target PH distributions

# Convert the data to long format
train_og_long <- pivot_longer(train_og, cols = where(is.numeric), names_to = "Variable", values_to = "Value")

# Create histograms for all numeric variables
ggplot(train_og_long, aes(x = Value)) +
  geom_histogram(bins = 30, fill = "blue", color = "black", alpha = 0.7) +
  facet_wrap(~Variable, scales = "free_x") +
  theme_minimal()

```

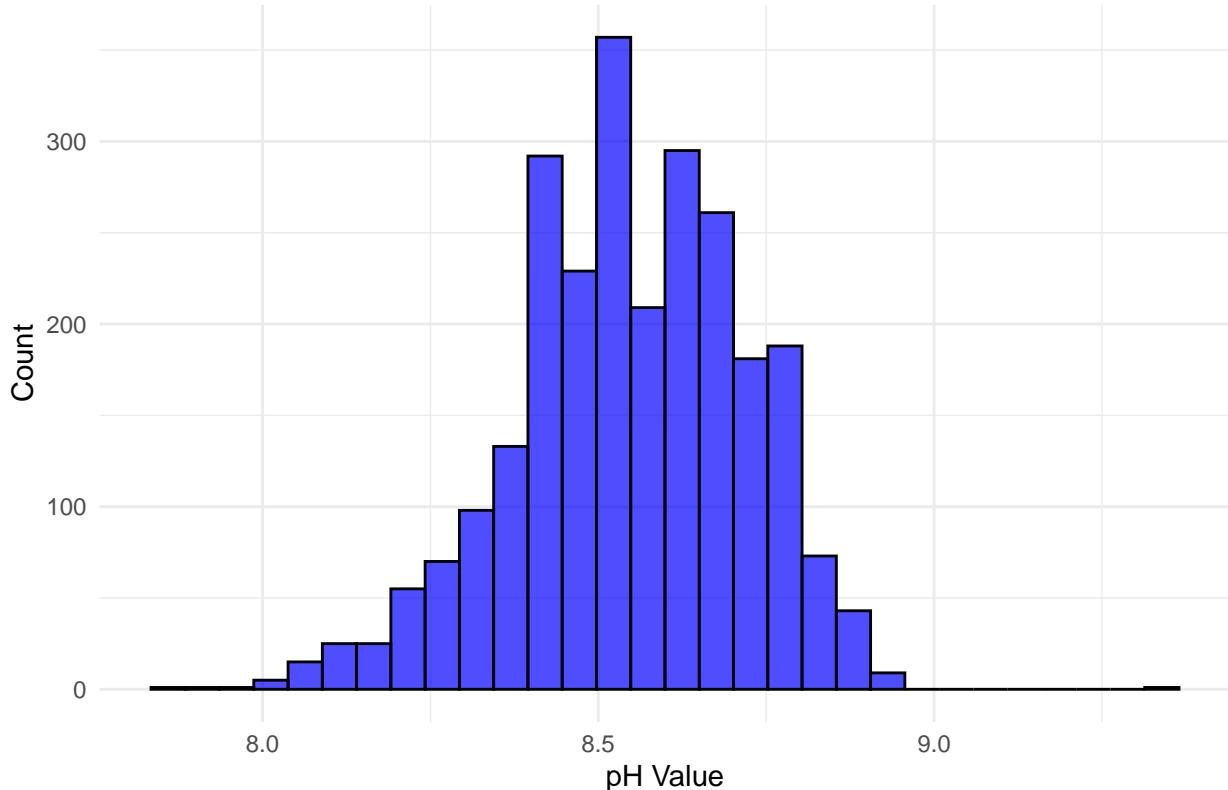


There are outliers seen in the distribution plots above, but none seem to be irregular or due to recording mistakes, so we will keep all values.

```
# Visualizing distribution of target PH
```

```
ggplot(train_og, aes(x = PH)) +
  geom_histogram(bins = 30, fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Histogram of pH Levels", x = "pH Value", y = "Count") +
  theme_minimal()
```

Histogram of pH Levels



There is a left sided tail in the distribution of PH, and so, how to maintain a sufficiently high pH level as a business question is of interest.

Data Pre-processing

Checking for and removing predictors with near-zero variance, as such predictors have very little variability and lend no predictive power.

```
# Viewing any predictors with near-zero variance
nzv <- nearZeroVar(train_og)
colnames(train_og)[nzv]

## [1] "Hyd.Pressure1"

# Removing near-zero variance column "Hyd.Pressure1" from train and new_test_data dataframes
train <- train_og[, -nearZeroVar(train_og)]
new_test_data <- new_test_data[, colnames(train)]

# Viewing remaining predictor names
colnames(train)

## [1] "Brand.Code"          "Carb.Volume"        "Fill.Ounces"
## [4] "PC.Volume"           "Carb.Pressure"       "Carb.Temp"
## [7] "PSC"                 "PSC.Fill"            "PSC.CO2"
```

```

## [10] "Mnf.Flow"           "Carb.Pressure1"      "Fill.Pressure"
## [13] "Hyd.Pressure2"       "Hyd.Pressure3"       "Hyd.Pressure4"
## [16] "Filler.Level"        "Filler.Speed"        "Temperature"
## [19] "Usage.cont"          "Carb.Flow"           "Density"
## [22] "MFR"                 "Balling"              "Pressure.Vacuum"
## [25] "PH"                  "Oxygen.Filler"       "Bowl.Setpoint"
## [28] "Pressure.Setpoint"   "Air.Pressurer"       "Alch.Rel"
## [31] "Carb.Rel"             "Balling.Lvl"

```

Checking for and removing highly correlated predictors

```

high_correlation <- findCorrelation(cor(train[,-c(1,25)]), cutoff = 0.75, exact = TRUE)
length(high_correlation)

```

```

## [1] 0

```

There are no highly correlated predictors from the training data, therefore we will not remove any predictors based on collinearity.

To visualize missing data:

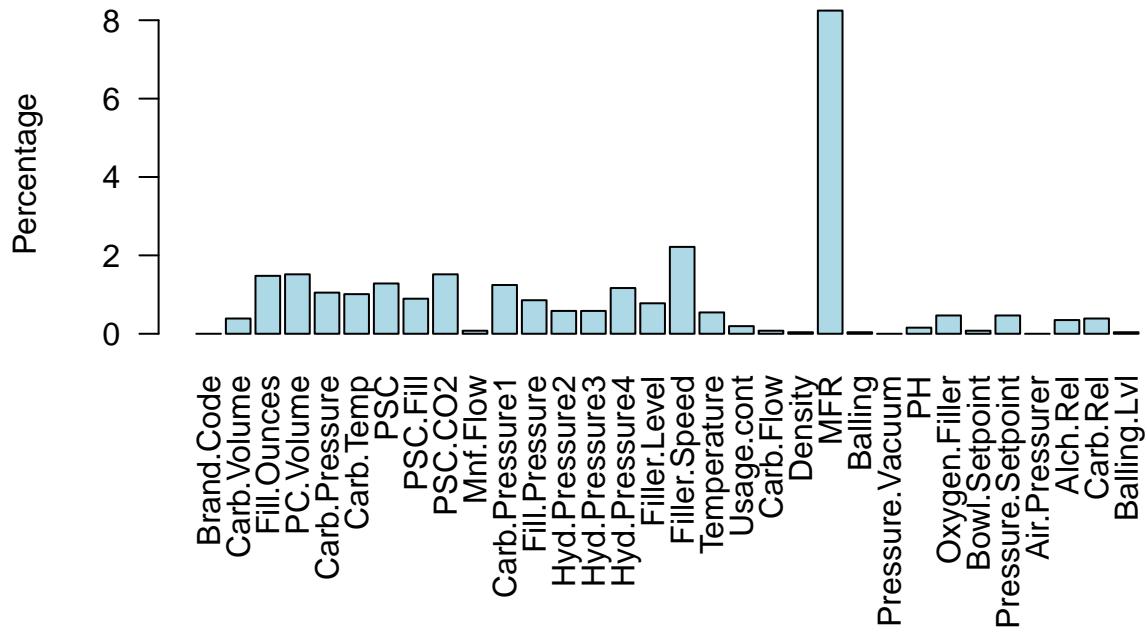
```

#Increase margin so the text can be read:
par(mar = c(10, 4, 4, 2))

train %>%
  sapply(function(x) sum(is.na(x)/length(x)*100)) %>%
  barplot(main = "Percentage of Missing Values",
          ylab = "Percentage",
          col = "lightblue",
          las = 2)

```

Percentage of Missing Values



To train split the data first:

```

set.seed(199)
#Dropping the 4 missing pH rows from the training dataset:
train<- train %>% drop_na(PH)

#We need to replace the empty spaces with NA in the brand code column:
train$Brand.Code<- replace(train$Brand.Code, train$Brand.Code=="", NA)

#We need to convert Brand.Code column to factor for imputation:
train$Brand.Code<- as.factor(train$Brand.Code)

#Train split 80/20:
train_idx<- createDataPartition(train$PH, p = 0.8, list = FALSE)
train_data<- train[train_idx, ]
test_data<- train[-train_idx, ]

#'train/test' are now ready for imputations:

```

We tested three imputation methods and compared them to see which one will be least likely to create deviation from the overall mean of the dataset.

Imputations

kNN imputation:

```
set.seed(199)
#impute train:
knn_imput_train<- kNN(train_data)
#impute test:
knn_imput_test<- kNN(test_data)
```

randomForest imputation

```
set.seed(199)
#Running the imputation for the training set:
rf_imput_train<- missForest(train_data)
#Running the imputation for the test set:
rf_imput_test<- missForest(test_data)

#Need to convert the imputed train object to dataframe:
rf_imput_train<- as.data.frame(rf_imput_train$ximp)
#Need to convert the imputed test object to dataframe:
rf_imput_test<- as.data.frame(rf_imput_test$ximp)
```

MICE imputation:

```
set.seed(199)
#Running the imputation for the training set:
mice_imput_train<- mice(train_data, m=5, maxit=50, meth='pmm', seed=100, printFlag = FALSE)
#Running the imputation for the test set:
mice_imput_test<- mice(test_data, m=5, maxit=50, meth='pmm', seed=100, printFlag = FALSE)

mice_1_train<- complete(mice_imput_train, 1) #We choose the first imputed data
mice_1_test<- complete(mice_imput_test, 1) #We choose the first imputed data
```

To check if everything is imputed:

```
is.na(c('knn_imput_train', 'rf_imput_train', 'mice_1_train', 'mice_1_test', 'rf_imput_test', 'knn_imput
## [1] FALSE
```

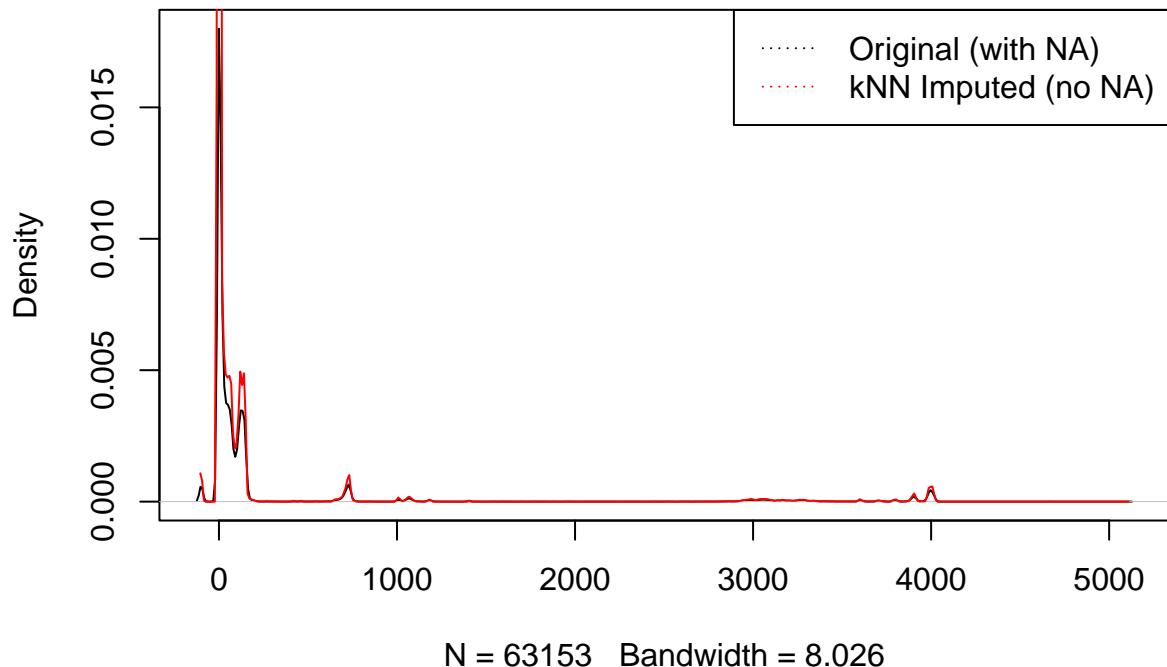
To visualize the variations of the dataset after imputation through the above three methods:

Imputed density plots

KNN imputed density plots

```
plot(density(unlist(train_data[, -1]), na.rm=TRUE),
      main='KNN imputed vs original data w/ NA removed')
lines(density(unlist(knn_imput_train[, -1]), na.rm=TRUE),
      col = 'red')
legend('topright', legend = c("Original (with NA)", "kNN Imputed (no NA)"),
       col = c("black", "red"), lty = 3)
```

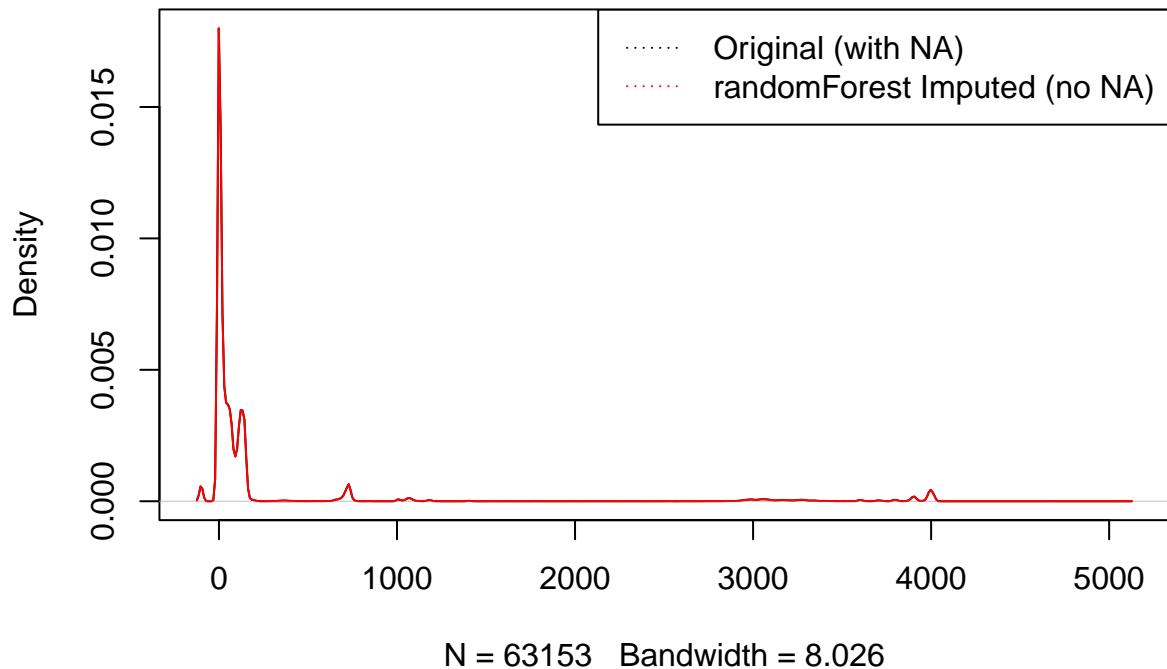
KNN imputed vs original data w/ NA removed



RandomForest imputed density plots

```
plot(density(unlist(train_data[, -1]), na.rm=TRUE),
      main='RF imputed vs original data w/ NA removed')
lines(density(unlist(rf_imput_train[, -1]), na.rm=TRUE),
      col = 'red')
legend('topright', legend = c("Original (with NA)", "randomForest Imputed (no NA)"),
       col = c("black", "red"), lty = 3)
```

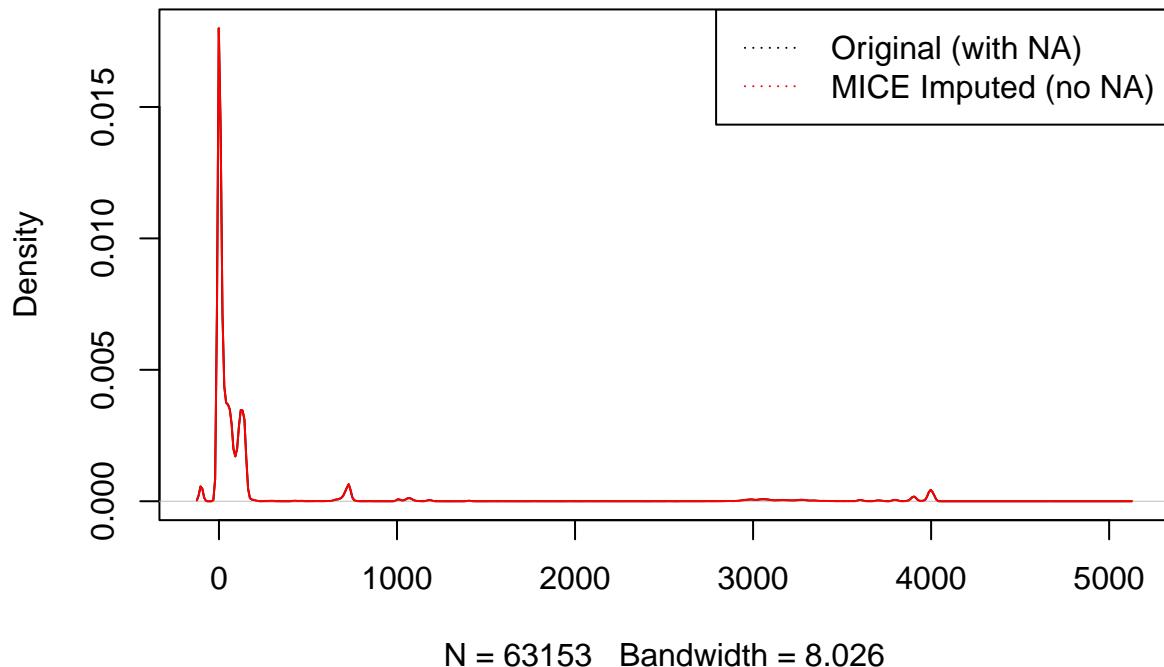
RF imputed vs original data w/ NA removed



MICE imputed density plots

```
plot(density(unlist(train_data[, -1]), na.rm=TRUE),
      main='MICE imputed vs original data w/ NA removed')
lines(density(unlist(mice_1_train[, -1]), na.rm=TRUE),
      col = 'red')
legend('topright', legend = c("Original (with NA)", "MICE Imputed (no NA)"),
       col = c("black", "red"), lty = 3)
```

MICE imputed vs original data w/ NA removed



Running a t-test to compare all column values except for the first column for both the original and the imputed dataset. P-value > 0.05, indicating the imputation did not change the overall data structure.

```
t.test(train[, -1], mice_1_train[, -1]) #No difference
```

```
##  
## Welch Two Sample t-test  
##  
## data: train[, -1] and mice_1_train[, -1]  
## t = 0.14608, df = 136603, p-value = 0.8839  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -7.723202 8.967115  
## sample estimates:  
## mean of x mean of y  
## 254.1059 253.4839
```

```
t.test(train[, -1], knn_imput_train[, -1]) #Significant difference
```

```
##  
## Welch Two Sample t-test  
##  
## data: train[, -1] and knn_imput_train[, -1]  
## t = 39.523, df = 128213, p-value < 2.2e-16
```

```

## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## 122.7717 135.5838
## sample estimates:
## mean of x mean of y
## 254.1059 124.9281

t.test(train[, -1], rf_imput_train[, -1])#No difference

## Welch Two Sample t-test
## data: train[, -1] and rf_imput_train[, -1]
## t = 0.16746, df = 136638, p-value = 0.867
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -7.629477 9.054945
## sample estimates:
## mean of x mean of y
## 254.1059 253.3932

```

Conclusion: KNN changes the dataset structure significantly, while randomForest and MICE did not. We will exclude KNN imputation and select MICE as our imputation method.

To separate the predictors from the target variables:

```

#Separate the PH column from the mice_1_test:
mice_1_test_predictors<- mice_1_test %>% select(-PH)
mice_1_test_target<- mice_1_test$PH

```

Data Modeling

Models:

Cubist

```

# Cubist Model
library(Cubist)

# Train Cubist model on MICE-imputed data
set.seed(199)
cubist_model <- train(
  PH ~ .,
  data = mice_1_train,
  method = "cubist",
  trControl = trainControl(method = "cv", number = 5),
  tuneGrid = expand.grid(committees = c(1, 5, 10), neighbors = c(0, 3, 5, 7))
)

```

```

# Make predictions on test set
cubist_pred <- predict(cubist_model, newdata = mice_1_test_predictors)

# Evaluate performance using caret functions
cubist_rmse <- caret::RMSE(cubist_pred, mice_1_test_target)
cubist_r2 <- caret::R2(cubist_pred, mice_1_test_target)

# Print results
cat("Cubist Model Performance:\n")

## Cubist Model Performance:

cat("RMSE:", cubist_rmse, "\n")

## RMSE: 0.103696

cat("R-squared:", cubist_r2, "\n")

## R-squared: 0.6491117

# Variable importance
varImp(cubist_model)

## cubist variable importance
##
##      only 20 most important variables shown (out of 33)
##
##          Overall
## Mnf.Flow           100.00
## Balling.Lvl        63.46
## Alch.Rel           60.90
## Balling            59.62
## Pressure.Vacuum    55.77
## Air.Pressurer       55.13
## Density             53.85
## Bowl.Setpoint        46.79
## Oxygen.Filler        44.87
## Carb.Rel            40.38
## Filler.Speed         40.38
## Temperature          40.38
## Brand.CodeC          39.74
## Hyd.Pressure3        39.74
## Carb.Pressure1        36.54
## Carb.Flow             33.97
## Carb.Volume            30.77
## Hyd.Pressure2          29.49
## Carb.Pressure           29.49
## Usage.cont              26.92

```

XGBoosting

```
# XGBoost Model
library(xgboost)

# Prepare data for XGBoost
xgb_train <- mice_1_train %>% select(-PH)
xgb_test <- mice_1_test %>% select(-PH)

# Convert factors to numeric (XGBoost requires numeric matrix)
xgb_train <- model.matrix(~ . -1, data = xgb_train)
xgb_test <- model.matrix(~ . -1, data = xgb_test)

# Create DMatrix objects
dtrain <- xgb.DMatrix(data = xgb_train, label = mice_1_train$PH)
dtest <- xgb.DMatrix(data = xgb_test, label = mice_1_test$PH)

# Set parameters
params <- list(
  objective = "reg:squarederror",
  eta = 0.01,
  max_depth = 6,
  min_child_weight = 5,
  subsample = 0.8,
  colsample_bytree = 0.8
)

# Train XGBoost model
set.seed(199)
xgb_model <- xgb.cv(
  params = params,
  data = dtrain,
  nrounds = 500,
  nfold = 5,
  early_stopping_rounds = 20,
  print_every_n = 10,
  verbose = 1
)

## [1] train-rmse:7.967691+0.003861    test-rmse:7.967678+0.015597
## Multiple eval metrics are present. Will use test_rmse for early stopping.
## Will train until test_rmse hasn't improved in 20 rounds.
##
## [11] train-rmse:7.206753+0.003525    test-rmse:7.206739+0.015919
## [21] train-rmse:6.518614+0.003254    test-rmse:6.518598+0.016173
## [31] train-rmse:5.896253+0.002913    test-rmse:5.896235+0.016496
## [41] train-rmse:5.333404+0.002642    test-rmse:5.333385+0.016747
## [51] train-rmse:4.824404+0.002384    test-rmse:4.824382+0.016983
## [61] train-rmse:4.364023+0.002157    test-rmse:4.363999+0.017184
## [71] train-rmse:3.947754+0.001915    test-rmse:3.947727+0.017401
## [81] train-rmse:3.571335+0.001773    test-rmse:3.571306+0.017509
## [91] train-rmse:3.230944+0.001588    test-rmse:3.230911+0.017661
## [101]    train-rmse:2.923124+0.001414   test-rmse:2.923084+0.017797
```

```

## [111] train-rmse:2.6444859+0.001260 test-rmse:2.6444818+0.017701
## [121] train-rmse:2.393193+0.001142 test-rmse:2.393149+0.017553
## [131] train-rmse:2.165675+0.001074 test-rmse:2.165604+0.017281
## [141] train-rmse:1.960003+0.000969 test-rmse:1.959906+0.017091
## [151] train-rmse:1.774027+0.000875 test-rmse:1.773927+0.016866
## [161] train-rmse:1.605936+0.000812 test-rmse:1.605856+0.016596
## [171] train-rmse:1.453992+0.000738 test-rmse:1.453903+0.016362
## [181] train-rmse:1.316696+0.000667 test-rmse:1.316612+0.016119
## [191] train-rmse:1.192629+0.000574 test-rmse:1.192581+0.015899
## [201] train-rmse:1.080548+0.000541 test-rmse:1.080580+0.015579
## [211] train-rmse:0.979240+0.000485 test-rmse:0.979296+0.015278
## [221] train-rmse:0.887747+0.000417 test-rmse:0.887803+0.014951
## [231] train-rmse:0.805179+0.000417 test-rmse:0.805289+0.014555
## [241] train-rmse:0.730635+0.000365 test-rmse:0.730834+0.014208
## [251] train-rmse:0.663345+0.000367 test-rmse:0.663653+0.013840
## [261] train-rmse:0.602622+0.000331 test-rmse:0.603092+0.013420
## [271] train-rmse:0.547837+0.000295 test-rmse:0.548522+0.013031
## [281] train-rmse:0.498369+0.000251 test-rmse:0.499372+0.012720
## [291] train-rmse:0.453775+0.000266 test-rmse:0.455190+0.012450
## [301] train-rmse:0.413471+0.000231 test-rmse:0.415370+0.012120
## [311] train-rmse:0.377200+0.000280 test-rmse:0.379576+0.011815
## [321] train-rmse:0.344483+0.000321 test-rmse:0.347506+0.011534
## [331] train-rmse:0.314974+0.000297 test-rmse:0.318712+0.011253
## [341] train-rmse:0.288428+0.000341 test-rmse:0.292948+0.010889
## [351] train-rmse:0.264511+0.000353 test-rmse:0.269860+0.010602
## [361] train-rmse:0.243042+0.000372 test-rmse:0.249388+0.010318
## [371] train-rmse:0.223730+0.000385 test-rmse:0.231011+0.010017
## [381] train-rmse:0.206413+0.000392 test-rmse:0.214781+0.009762
## [391] train-rmse:0.190911+0.000451 test-rmse:0.200397+0.009434
## [401] train-rmse:0.177083+0.000504 test-rmse:0.187712+0.009172
## [411] train-rmse:0.164660+0.000550 test-rmse:0.176478+0.008859
## [421] train-rmse:0.153616+0.000579 test-rmse:0.166678+0.008556
## [431] train-rmse:0.143845+0.000638 test-rmse:0.158185+0.008220
## [441] train-rmse:0.135162+0.000631 test-rmse:0.150836+0.007961
## [451] train-rmse:0.127497+0.000680 test-rmse:0.144444+0.007608
## [461] train-rmse:0.120694+0.000739 test-rmse:0.138940+0.007311
## [471] train-rmse:0.114694+0.000801 test-rmse:0.134182+0.006947
## [481] train-rmse:0.109379+0.000817 test-rmse:0.130162+0.006668
## [491] train-rmse:0.104715+0.000826 test-rmse:0.126616+0.006335
## [500] train-rmse:0.101016+0.000872 test-rmse:0.123850+0.006061

```

```

# Final model with optimal rounds
final_xgb <- xgboost(
  params = params,
  data = dtrain,
  nrounds = xgb_model$best_iteration,
  verbose = 0
)

# Make predictions
xgb_pred <- predict(final_xgb, dtest)

# Evaluate performance using caret functions
xgb_rmse <- caret::RMSE(xgb_pred, mice_1_test_target)

```

```

xgb_r2 <- caret::R2(xgb_pred, mice_1_test_target)

# Print results
cat("\nXGBoost Model Performance:\n")

## 
## XGBoost Model Performance:

cat("RMSE:", xgb_rmse, "\n")

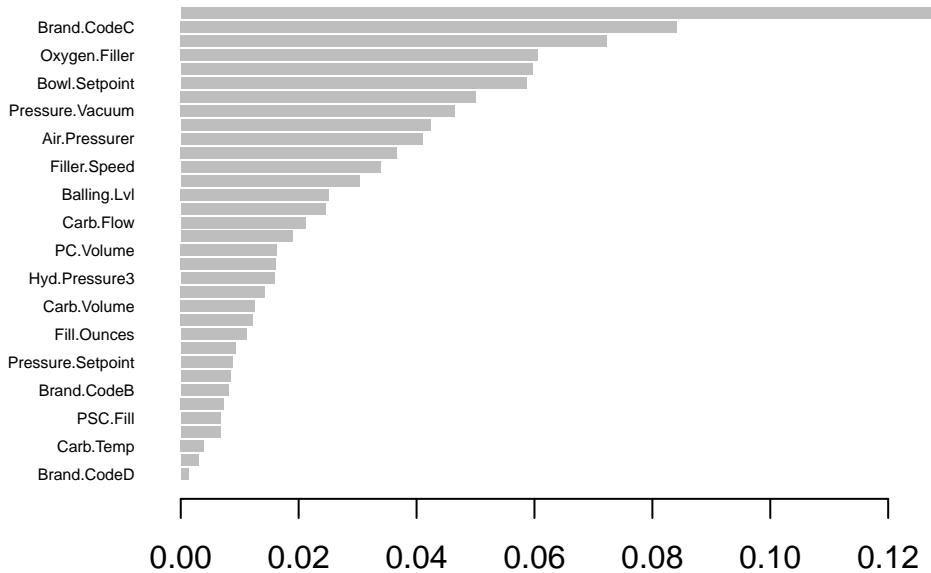
## RMSE: 0.1244964

cat("R-squared:", xgb_r2, "\n")

## R-squared: 0.5951791

# Variable importance
importance_matrix <- xgb.importance(model = final_xgb)
xgb.plot.importance(importance_matrix)

```



RandomForest

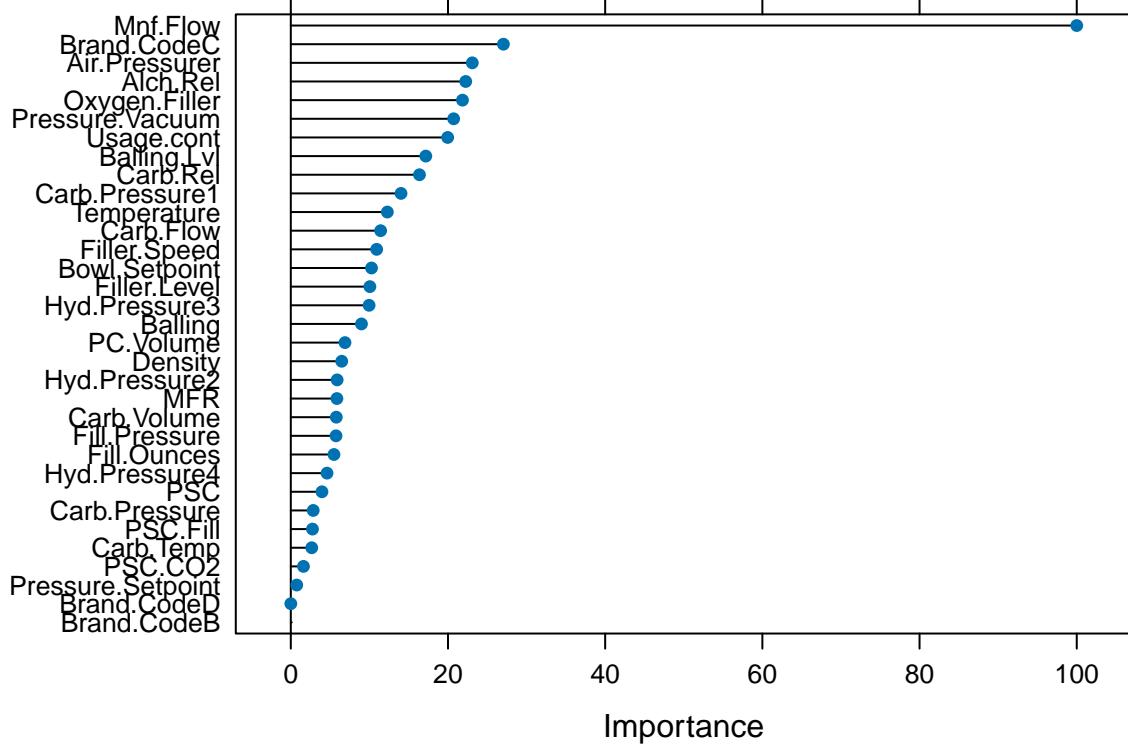
```

#The original code was tuned by the code ' rf_tune<- expand.grid(mtry = seq(20, (ncol(mice_1_train))-1),
#The best model was mtry = 29.
#In here we applied mtry = 29 directly to reduce computation time when rendering the html file

#fit model with MICE imputed data:
rf_model_mice<-train(PH~.,
                       data = mice_1_train,
                       method = 'rf',
                       tuneGrid = data.frame(mtry=29),
                       trControl = trainControl(method='cv', number=10),
                       ntree = 1000)

#Checking importance chart from the fitted model:
plot(varImp(rf_model_mice))

```



```

#Making prediction based on the training test data:
rf_pred_mice<- predict(rf_model_mice, mice_1_test_predictors)

#Get the best tuned hyperparameters:
rf_model_mice$bestTune #Best model mtry = 29

```

```

## mtry
## 1 29

```

```

#Getting the residual metrics:
postResample(rf_pred_mice, mice_1_test_target)

##          RMSE      Rsquared        MAE
## 0.10314889 0.65809554 0.07203081

#RMSE = 0.10267; R^2 = 0.66

```

MARS Model

```

library(earth)

set.seed(199)

# Using the MICE test

# Training data
x_train <- mice_1_train %>% select(-PH)
y_train <- mice_1_train$PH

# Testing data
x_test <- mice_1_test %>% select(-PH)
y_test <- mice_1_test$PH

# Create grid for tuning
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

# Set the cross-validation control
mars_ctrl <- trainControl(
  method = "cv",
  number = 10 # use 10 fold
)

# Train with MARS model with MICE imputed
marsTuned <- train(x = x_train, y = y_train,
                     method = "earth",
                     tuneGrid = marsGrid,
                     trControl = mars_ctrl)
print(marsTuned)

## Multivariate Adaptive Regression Spline
##
## 2055 samples
##    31 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1850, 1849, 1850, 1850, 1849, 1850, ...
## Resampling results across tuning parameters:
##

```

##	degree	nprune	RMSE	Rsquared	MAE
##	1	2	0.1514953	0.2237390	0.11899621
##	1	3	0.1437887	0.3006491	0.11220950
##	1	4	0.1415303	0.3221462	0.10990329
##	1	5	0.1417003	0.3212013	0.11018435
##	1	6	0.1400126	0.3382204	0.10855800
##	1	7	0.1387636	0.3499414	0.10713851
##	1	8	0.1368651	0.3673042	0.10588062
##	1	9	0.1339871	0.3933100	0.10370016
##	1	10	0.1324346	0.4075206	0.10237560
##	1	11	0.1314111	0.4170143	0.10143639
##	1	12	0.1309035	0.4215777	0.10050973
##	1	13	0.1307132	0.4231422	0.10012509
##	1	14	0.1305281	0.4247589	0.09988682
##	1	15	0.1306661	0.4238054	0.09981952
##	1	16	0.1305776	0.4245179	0.09977317
##	1	17	0.1304474	0.4256239	0.09961406
##	1	18	0.1304421	0.4256903	0.09955534
##	1	19	0.1304816	0.4254118	0.09960085
##	1	20	0.1306250	0.4242923	0.09971317
##	1	21	0.1306250	0.4242923	0.09971317
##	1	22	0.1306250	0.4242923	0.09971317
##	1	23	0.1306250	0.4242923	0.09971317
##	1	24	0.1306250	0.4242923	0.09971317
##	1	25	0.1306250	0.4242923	0.09971317
##	1	26	0.1306250	0.4242923	0.09971317
##	1	27	0.1306250	0.4242923	0.09971317
##	1	28	0.1306250	0.4242923	0.09971317
##	1	29	0.1306250	0.4242923	0.09971317
##	1	30	0.1306250	0.4242923	0.09971317
##	1	31	0.1306250	0.4242923	0.09971317
##	1	32	0.1306250	0.4242923	0.09971317
##	1	33	0.1306250	0.4242923	0.09971317
##	1	34	0.1306250	0.4242923	0.09971317
##	1	35	0.1306250	0.4242923	0.09971317
##	1	36	0.1306250	0.4242923	0.09971317
##	1	37	0.1306250	0.4242923	0.09971317
##	1	38	0.1306250	0.4242923	0.09971317
##	2	2	0.1514953	0.2237390	0.11899621
##	2	3	0.1436695	0.3018568	0.11240008
##	2	4	0.1403635	0.3337447	0.10887636
##	2	5	0.1394706	0.3425308	0.10755214
##	2	6	0.1374276	0.3615108	0.10592524
##	2	7	0.1372795	0.3647319	0.10636284
##	2	8	0.1344145	0.3892523	0.10326492
##	2	9	0.1331279	0.4022120	0.10277393
##	2	10	0.1316346	0.4152980	0.10177973
##	2	11	0.1301831	0.4282434	0.10041772
##	2	12	0.1287907	0.4407272	0.09916174
##	2	13	0.1273739	0.4527883	0.09854193
##	2	14	0.1266665	0.4587140	0.09795019
##	2	15	0.1262038	0.4630748	0.09726924
##	2	16	0.1262381	0.4631001	0.09719034
##	2	17	0.1254892	0.4691968	0.09682219

```

##   2      18    0.1254228  0.4701284  0.09671140
##   2      19    0.1239732  0.4817974  0.09542425
##   2      20    0.1239965  0.4816945  0.09519956
##   2      21    0.1240984  0.4810424  0.09529162
##   2      22    0.1239582  0.4824537  0.09519960
##   2      23    0.1240201  0.4821097  0.09517055
##   2      24    0.1233084  0.4879308  0.09486434
##   2      25    0.1231455  0.4893915  0.09475091
##   2      26    0.1231362  0.4895272  0.09479505
##   2      27    0.1228389  0.4919890  0.09449132
##   2      28    0.1229403  0.4911703  0.09462910
##   2      29    0.1228874  0.4916210  0.09463973
##   2      30    0.1228859  0.4916017  0.09473338
##   2      31    0.1228332  0.4921195  0.09465830
##   2      32    0.1226902  0.4934920  0.09458925
##   2      33    0.1226276  0.4939149  0.09473067
##   2      34    0.1226387  0.4939551  0.09477679
##   2      35    0.1225993  0.4942070  0.09472582
##   2      36    0.1226262  0.4939930  0.09476846
##   2      37    0.1226059  0.4941548  0.09476536
##   2      38    0.1226059  0.4941548  0.09476536
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 35 and degree = 2.

```

```

# Make predictions
mars_pred <- predict(marsTuned, newdata = x_test)
MARS <- postResample(pred = mars_pred, obs = y_test)

print(MARS)

```

```

##          RMSE      Rsquared        MAE
## 0.12586934 0.48272308 0.09391658

```

KNN

```

KNNctrl <- trainControl(method = "cv", number = 10)

knnModel <- train(PH ~ .,
                    data = mice_1_train,
                    preProcess = c("center", "scale"),
                    method = "knn",
                    tuneLength = 25,
                    trControl = KNNctrl)

print(knnModel)

```

```

## k-Nearest Neighbors
##
## 2055 samples
##   31 predictor

```

```

## 
## Pre-processing: centered (33), scaled (33)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1850, 1849, 1850, 1850, 1852, 1848, ...
## Resampling results across tuning parameters:
##
##     k    RMSE    Rsquared    MAE
##     5   0.1224141  0.4997742  0.09120416
##     7   0.1204170  0.5150157  0.09037700
##     9   0.1211325  0.5096751  0.09139358
##    11   0.1217984  0.5051183  0.09239463
##    13   0.1229834  0.4963326  0.09361851
##    15   0.1236280  0.4920899  0.09396729
##    17   0.1242091  0.4875551  0.09479626
##    19   0.1252448  0.4787293  0.09582487
##    21   0.1262521  0.4699312  0.09673347
##    23   0.1268199  0.4650547  0.09716382
##    25   0.1273934  0.4604880  0.09777410
##    27   0.1276426  0.4586306  0.09815730
##    29   0.1280989  0.4549434  0.09854262
##    31   0.1285869  0.4501513  0.09895968
##    33   0.1290416  0.4461825  0.09923231
##    35   0.1295093  0.4420588  0.09963684
##    37   0.1300335  0.4372704  0.10019341
##    39   0.1305118  0.4329878  0.10055708
##    41   0.1309480  0.4290319  0.10094846
##    43   0.1311955  0.4268270  0.10130004
##    45   0.1315905  0.4234101  0.10160068
##    47   0.1319103  0.4202826  0.10180123
##    49   0.1323381  0.4165378  0.10205485
##    51   0.1325308  0.4149477  0.10224778
##    53   0.1328017  0.4127152  0.10243384
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.

```

```

#Predict
knnPred <- predict(knnModel, newdata = mice_1_test_predictors)

KNN <- postResample(pred = knnPred, obs = mice_1_test_target)

print(KNN)

```

```

##      RMSE    Rsquared    MAE
## 0.12447437 0.49816629 0.09450614

```

Neural Network

```

nnetGrid <- expand.grid(.decay = c(0, .01, 1),
                        .size = c(1:10),
                        .bag = FALSE)

```

```

ctrl <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 3,
  verboseIter = FALSE
)

set.seed(199)
nnetTune <- train(PH ~ .,
  data = mice_1_train,
  method = "avNNet",
  tuneGrid = nnetGrid,
  trControl = ctrl,
  linout = TRUE, trace = FALSE,
  MaxNWts = 5* (ncol(x_train) + 1) + 5 + 1,
  maxit = 250)

print(nnetTune)

## Model Averaged Neural Network
##
## 2055 samples
##    31 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1850, 1849, 1850, 1850, 1849, 1850, ...
## Resampling results across tuning parameters:
##
##     decay  size   RMSE      Rsquared      MAE
##     0.00    1    0.1754327  0.00804171  0.13903648
##     0.00    2    0.1715665  0.02335966  0.13825397
##     0.00    3    0.1744395  0.02599566  0.13855446
##     0.00    4    0.1707978  0.04207170  0.13761640
##     0.00    5        NaN      NaN        NaN
##     0.00    6        NaN      NaN        NaN
##     0.00    7        NaN      NaN        NaN
##     0.00    8        NaN      NaN        NaN
##     0.00    9        NaN      NaN        NaN
##     0.00   10        NaN      NaN        NaN
##     0.01    1    0.1390160  0.39379378  0.10870226
##     0.01    2    0.1318028  0.42229737  0.10102288
##     0.01    3    0.1286561  0.44554970  0.09854000
##     0.01    4    0.1291078  0.44273707  0.09871829
##     0.01    5        NaN      NaN        NaN
##     0.01    6        NaN      NaN        NaN
##     0.01    7        NaN      NaN        NaN
##     0.01    8        NaN      NaN        NaN
##     0.01    9        NaN      NaN        NaN
##     0.01   10        NaN      NaN        NaN
##     1.00    1    0.1359557  0.38328540  0.10648380
##     1.00    2    0.1325795  0.41725867  0.10273293
##     1.00    3    0.1304019  0.42765076  0.10079883

```

```

##   1.00    4    0.1291404  0.43906592  0.09949838
##   1.00    5        NaN        NaN        NaN
##   1.00    6        NaN        NaN        NaN
##   1.00    7        NaN        NaN        NaN
##   1.00    8        NaN        NaN        NaN
##   1.00    9        NaN        NaN        NaN
##   1.00   10        NaN        NaN        NaN
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 3, decay = 0.01 and bag = FALSE.

```

```

#Predict
nnet_pred <- predict(nnetTune, newdata = mice_1_test_predictors)

nnet <- postResample(pred = nnet_pred, obs = mice_1_test_target)

print(nnet)

```

```

##      RMSE  Rsquared      MAE
## 0.1320920 0.4425870 0.1025863

```

OLS

```

set.seed(199)
# Fit OLS model
ols_model <- lm(PH ~ ., data = mice_1_train)

# Make predictions on test set
ols_pred <- predict(ols_model, newdata = mice_1_test_predictors)

# Evaluate performance
postResample(pred = ols_pred, obs = mice_1_test_target)

```

```

##      RMSE  Rsquared      MAE
## 0.1349770 0.4034361 0.1042772

```

```

# Variable importance
varImp(ols_model)

```

```

##          Overall
## Brand.CodeB    3.4771114
## Brand.CodeC    1.9864550
## Brand.CodeD    2.2989544
## Carb.Volume    1.3242265
## Fill.Ounces    2.2554517
## PC.Volume     2.0391625
## Carb.Pressure  1.4257649
## Carb.Temp      0.9403136
## PSC            1.5502937

```

```

## PSC.Fill          0.8816379
## PSC.CO2          1.9429217
## Mnf.Flow         14.2569386
## Carb.Pressure1   8.9946575
## Fill.Pressure    2.3585439
## Hyd.Pressure2    2.0171494
## Hyd.Pressure3    5.1875066
## Hyd.Pressure4    0.0326008
## Filler.Level     1.2530198
## Filler.Speed     0.3772844
## Temperature      6.0218163
## Usage.cont        5.6213878
## Carb.Flow         2.5768755
## Density           4.6385287
## MFR               0.3822669
## Balling           2.5496042
## Pressure.Vacuum  2.4596148
## Oxygen.Filler     3.4475357
## Bowl.Setpoint     4.6898761
## Pressure.Setpoint 4.9529129
## Air.Pressurer     1.3730754
## Alch.Rel          2.8595215
## Carb.Rel          0.5819047
## Balling.Lvl       4.2212776

```

PLS

```

set.seed(199)
library(pls)

# Fit PLS model
pls_model <- plsr(PH ~ ., data = mice_1_train, scale = TRUE, validation = "CV")

# Make predictions on test set
pls_pred <- predict(pls_model, newdata = mice_1_test_predictors)

pls_pred <- pls_pred[, , 1] # Extract first component
pls_pred <- as.numeric(pls_pred) # Convert to vector

```

```

# Evaluate Performance
postResample(pred = pls_pred, obs = mice_1_test_target)

```

```

##      RMSE  Rsquared      MAE
## 0.1493953 0.2690851 0.1182675

```

```

# Variable importance
varImp(pls_model)

```

```

##                               Overall
## Brand.CodeB            0.0064842779

```

```

## Brand.CodeC      0.0107228184
## Brand.CodeD      0.0049055201
## Carb.Volume      0.0024089791
## Fill.Ounces      0.0035263737
## PC.Volume        0.0017910647
## Carb.Pressure    0.0018100310
## Carb.Temp         0.0010736417
## PSC              0.0035543716
## PSC.Fill         0.0014938553
## PSC.CO2          0.0027305961
## Mnf.Flow         0.0142025087
## Carb.Pressure1   0.0051411708
## Fill.Pressure    0.0062427053
## Hyd.Pressure2    0.0062186293
## Hyd.Pressure3    0.0077786102
## Hyd.Pressure4    0.0036307354
## Filler.Level     0.0086793343
## Filler.Speed     0.0009793104
## Temperature      0.0068354996
## Usage.cont       0.0094245196
## Carb.Flow        0.0046640937
## Density           0.0036497488
## MFR              0.0007596970
## Balling          0.0025687712
## Pressure.Vacuum  0.0060048775
## Oxygen.Filler    0.0049491123
## Bowl.Setpoint    0.0097685926
## Pressure.Setpoint 0.0077500173
## Air.Pressurer    0.0010303194
## Alch.Rel         0.0042041906
## Carb.Rel         0.0042283147
## Balling.Lvl      0.0034004872

```

Ridge

```

set.seed(199)

# Ridge regression model
library(glmnet)

# converting predictor variables to matrices
x_train <- as.matrix(mice_1_train %>% select(-PH))
y_train <- mice_1_train$PH # Target variable

x_test <- as.matrix(mice_1_test_predictors)
y_test <- mice_1_test_target # Test target

ridge_model <- glmnet(x_train, y_train, alpha = 0)

# Using cross validation to find best lambda for regularization
cv_ridge <- cv.glmnet(x_train, y_train, alpha = 0)
best_lambda <- cv_ridge$lambda.min # Optimal lambda

```

```

best_lambda

## [1] 0.007716322

# Training ridge model
ridge_best <- glmnet(x_train, y_train, alpha = 0, lambda = best_lambda)

# Making predictions on test set
ridge_pred <- predict(ridge_best, newx = x_test)
ridge_pred <- as.numeric(ridge_pred) # Ensure correct format

# Evaluate performance
postResample(pred = ridge_pred, obs = y_test)

##      RMSE    Rsquared      MAE
## 0.1440366 0.3205567 0.1098951

# Variable importance
varImp(ridge_best, lambda = best_lambda)

##                               Overall
## Brand.Code          0.000000e+00
## Carb.Volume        3.120112e-02
## Fill.Ounces        1.029650e-01
## PC.Volume          1.451594e-01
## Carb.Pressure      2.566385e-03
## Carb.Temp          1.965630e-04
## PSC                1.103616e-01
## PSC.Fill           2.278810e-02
## PSC.CO2            1.499224e-01
## Mnf.Flow           6.118405e-04
## Carb.Pressure1     6.255669e-03
## Fill.Pressure      2.432110e-03
## Hyd.Pressure2      1.219939e-04
## Hyd.Pressure3      2.031186e-03
## Hyd.Pressure4      5.332656e-04
## Filler.Level       1.070469e-04
## Filler.Speed       5.775458e-06
## Temperature        2.201417e-02
## Usage.cont          7.511107e-03
## Carb.Flow           1.199855e-05
## Density             9.559674e-02
## MFR                 3.351758e-05
## Balling             1.809523e-02
## Pressure.Vacuum    1.902953e-04
## Oxygen.Filler       2.195670e-01
## Bowl.Setpoint       1.964084e-03
## Pressure.Setpoint   9.926179e-03
## Air.Pressurer       1.049428e-03
## Alch.Rel            6.466887e-02
## Carb.Rel            1.275639e-01
## Balling.Lvl         4.790359e-03

```

SVM

```
# Set up cross-validation and preprocessing
svm_ctrl <- trainControl(method = "cv", number = 10)

# Train SVM with Radial Basis Function kernel
set.seed(199)
svm_model <- train(PH ~ .,
                     data = mice_1_train,
                     method = "svmRadial",
                     trControl = svm_ctrl,
                     preProcess = c("center", "scale"), # SVMs are sensitive to scale of input features,
                     tuneLength = 10)

# View model performance across hyperparameters
print(svm_model)

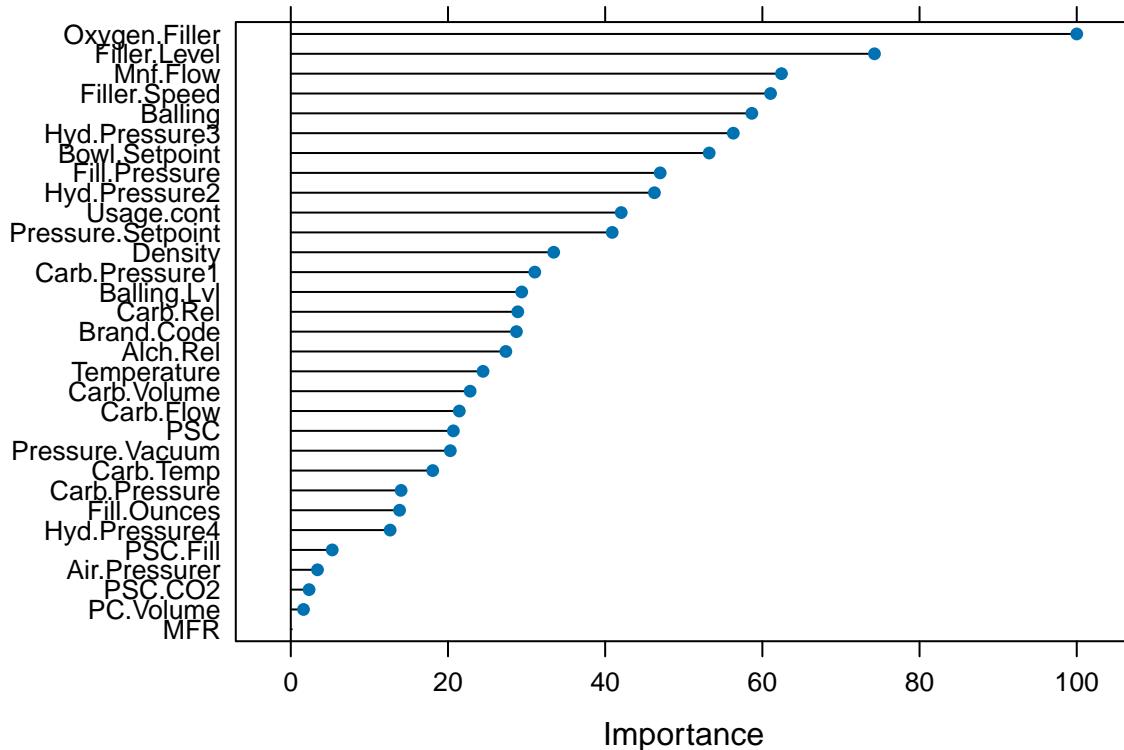
## Support Vector Machines with Radial Basis Function Kernel
##
## 2055 samples
##    31 predictor
##
## Pre-processing: centered (33), scaled (33)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1850, 1849, 1850, 1850, 1849, 1850, ...
## Resampling results across tuning parameters:
##
##     C      RMSE      Rsquared      MAE
##     0.25   0.1249236  0.4809823  0.09319563
##     0.50   0.1220407  0.5017754  0.09005954
##     1.00   0.1198158  0.5178346  0.08790545
##     2.00   0.1177696  0.5331031  0.08660305
##     4.00   0.1167118  0.5420557  0.08607914
##     8.00   0.1163228  0.5470061  0.08629362
##    16.00   0.1165203  0.5504831  0.08664444
##    32.00   0.1193825  0.5385023  0.08887790
##    64.00   0.1237570  0.5192078  0.09188947
##   128.00   0.1287455  0.4982280  0.09573496
##
## Tuning parameter 'sigma' was held constant at a value of 0.0210289
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.0210289 and C = 8.

# Make predictions
svm_pred <- predict(svm_model, newdata = mice_1_test_predictors)

# Evaluate performance
svm_metrics <- postResample(pred = svm_pred, obs = mice_1_test_target)
print(svm_metrics)

##          RMSE      Rsquared      MAE
## 0.12255388  0.51765506  0.08977719
```

```
# Variable importance
svm_varimp <- varImp(svm_model)
plot(svm_varimp)
```



Support Vector Machines with Radial Basis Function Kernel 2055 samples 32 predictor

Pre-processing: centered (34), scaled (34) Resampling: Cross-Validated (10 fold) Summary of sample sizes: 1850, 1849, 1850, 1850, 1849, 1850, ... Resampling results across tuning parameters: C RMSE Rsquared MAE

0.25 0.1254060 0.4781279 0.09351635 0.50 0.1219222 0.5036008 0.08993672 1.00 0.1198646 0.5178693
 0.08787178 2.00 0.1176100 0.5337863 0.08620710 4.00 0.1157883 0.5474234 0.08530468 8.00 0.1154913
 0.5519441 0.08560608 16.00 0.1161971 0.5514808 0.08666007 32.00 0.1183374 0.5443769 0.08822439 64.00
 0.1225489 0.5261688 0.09184539 128.00 0.1269883 0.5076909 0.09508938

Tuning parameter ‘sigma’ was held constant at a value of 0.02010561 RMSE was used to select the optimal model using the smallest value. The final values used for the model were sigma = 0.02010561 and C = 8. RMSE Rsquared MAE 0.12160863 0.52309895 0.08841903

Plotting the model metrics

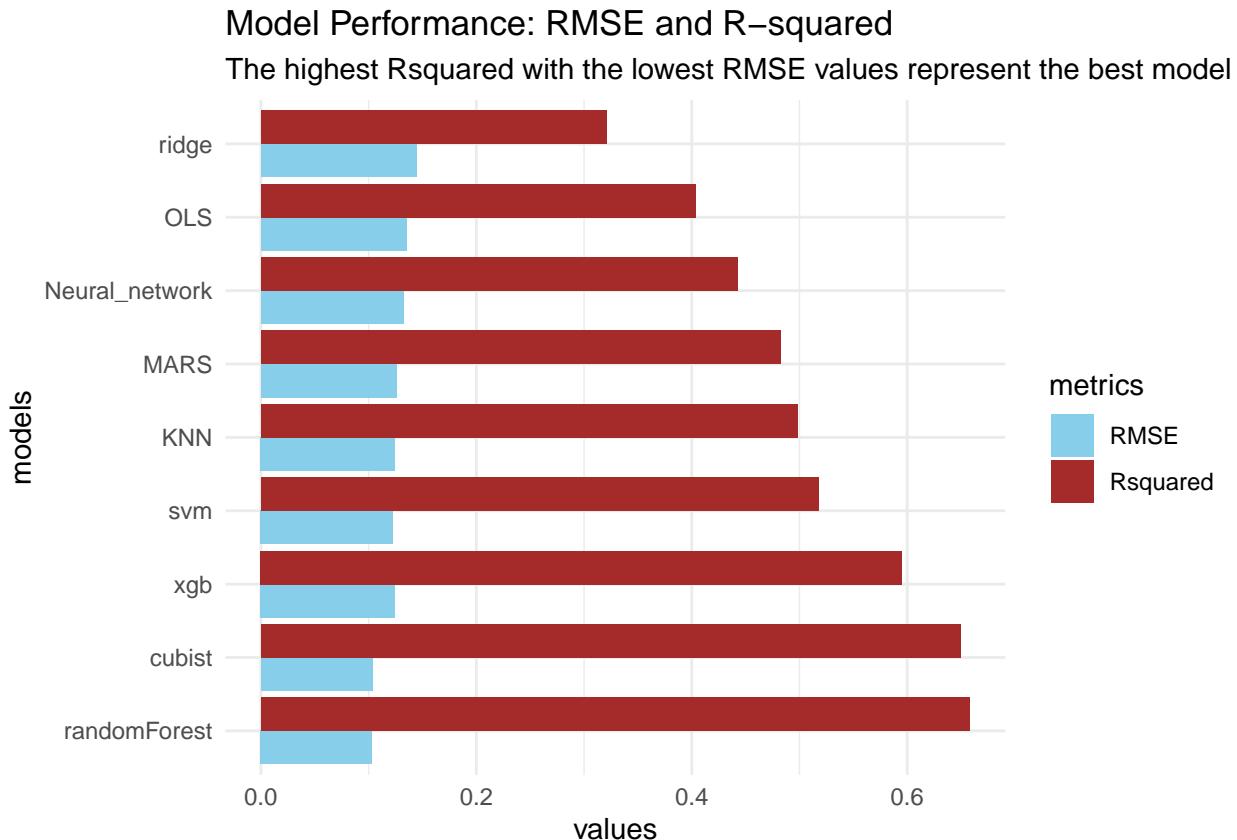
```
#Compile all the metrics into a list:
metrics<- list(cubist = postResample(cubist_pred, mice_1_test_target),
```

```

xgb = postResample(xgb_pred, mice_1_test_target),
randomForest = postResample(rf_pred_mice, mice_1_test_target),
MARS = postResample(mars_pred, mice_1_test_target),
KNN = postResample(knnPred, mice_1_test_target),
Neural_network = postResample(nnet_pred, mice_1_test_target),
OLS = postResample(ols_pred, mice_1_test_target),
ridge = postResample(ridge_pred, mice_1_test_target),
svm = postResample(svm_pred, mice_1_test_target))

#Convert the list into a DataFrame and then in a tidy form:
metrics_df<- as.data.frame(metrics)
metrics_t<- t(metrics_df) #Transpose the rows to column and vice versa
metrics_t<- as_tibble(metrics_t, rownames='models')
metrics_long<- metrics_t %>% pivot_longer(col=c(2:4),
                                              names_to = 'metrics',
                                              values_to = 'values')
# We will only look at RSquare and RMSE
metrics_long<- metrics_long %>%
  filter(metrics != 'MAE')
# Re-arranging the metrics values
metrics_long$models <- factor(metrics_long$models,
                               levels = metrics_long$models[metrics_long$metrics == 'Rsquared'][order(-metrics_long$values)])
#Plotting all metrics in a barplot:
ggplot(metrics_long, aes(x=models, y=values, fill=metrics))+
  geom_bar(stat = "identity", position='dodge')+
  coord_flip()+
  labs(title = "Model Performance: RMSE and R-squared", subtitle = 'The highest Rsquared with the lowest RMSE',
       scale_fill_manual(values = c("RMSE" = "skyblue", "Rsquared" = "brown")) +
  theme_minimal()

```



RandomForest has the highest RSquared and RMSE

Exploring the top ten predictor relationships with the target variable:

```
#select top ten predictor names from the RF model:
rf_ten<- varImp(rf_model_mice)
top_ten_predictor<- rf_ten$importance %>%
  as.data.frame() %>%
  rownames_to_column(var = 'predictor') %>%
  arrange(desc(Overall)) %>%
  slice_max(Overall, n=11) %>%
  pull(predictor)

#Since the brand.code was converted as a factor, we will exclude a single factor 'C' from the correlation calculation
existing_vars <- top_ten_predictor %in% colnames(train_og)

#Impute the original data as a whole for correlation calculation:
imputed_train_og<- mice(train_og, m=5, maxit=50, meth='pmm', seed=100, printFlag = FALSE)
imputed_train_og<- complete(imputed_train_og, 1)

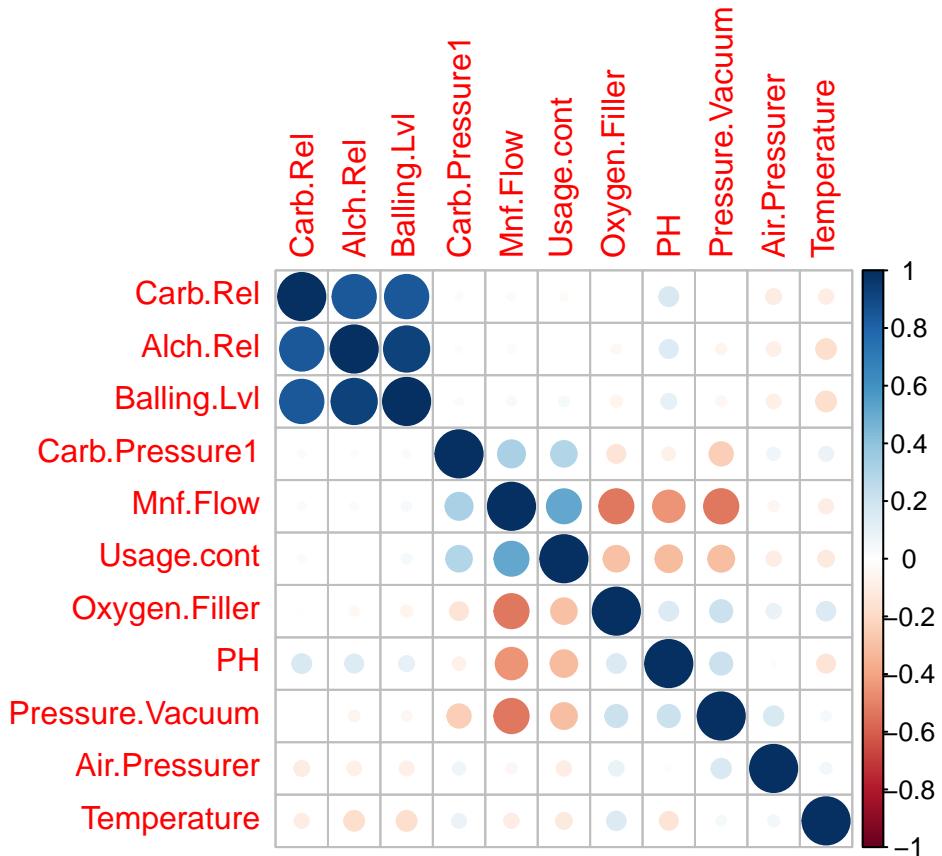
#Calculate the correlation of these predictors with the target variable pH:
corr_top_ten<- imputed_train_og %>%
```

```

select(all_of(c("PH", existing_vars)))

#Correlation plot:
corr_top_ten %>% cor() %>% corrplot::corrplot(order = 'hclust')

```



Data Prediction and exportation to excel

Imputing new test data

Before Imputation

```

#Remove pH column from the dataset:
new_test_data<- new_test_data %>% select(-PH)

#Converting empty values to NA in 'Brand.Code' Column:
new_test_data$Brand.Code<- replace(new_test_data$Brand.Code, new_test_data$Brand.Code=="", NA)

#Converting 'Brand.Code' column as factors:
new_test_data$Brand.Code<- as.factor(new_test_data$Brand.Code)

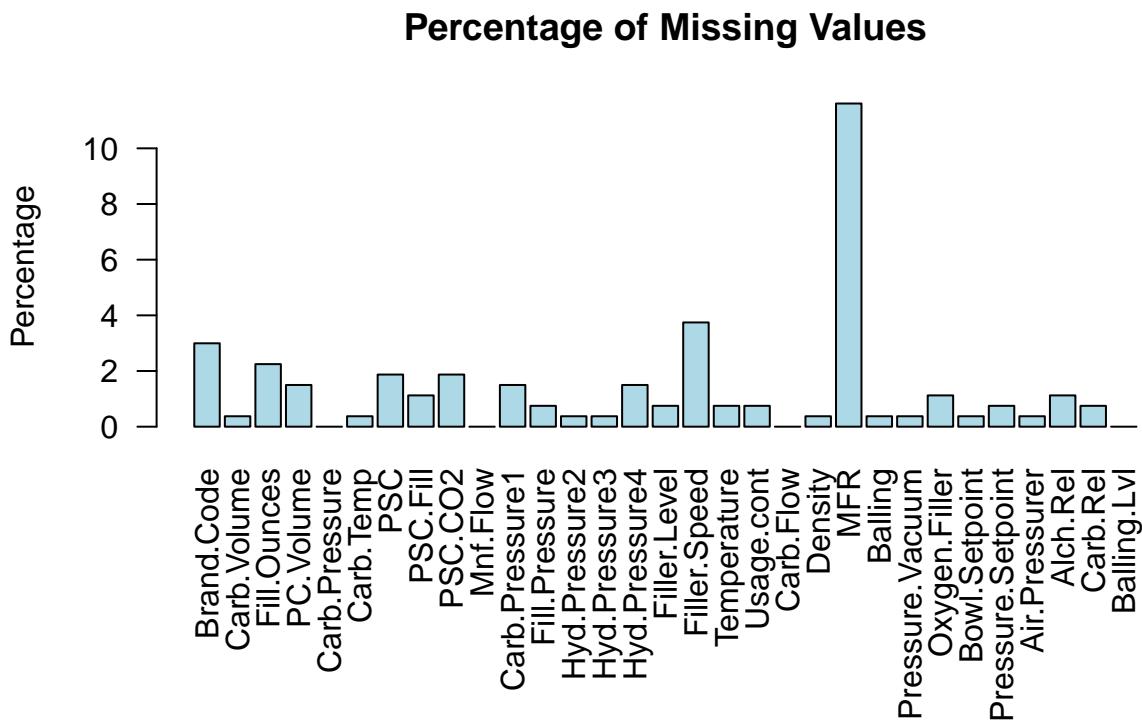
#Checking missing data in the new_test_data set:
par(mar = c(10, 4, 4, 2))

```

```

new_test_data %>%
  sapply(function(x) sum(is.na(x)/length(x)*100)) %>%
  barplot(main = "Percentage of Missing Values",
          ylab = "Percentage",
          col = "lightblue",
          las = 2)

```



After MICE Imputation and visualization:

```

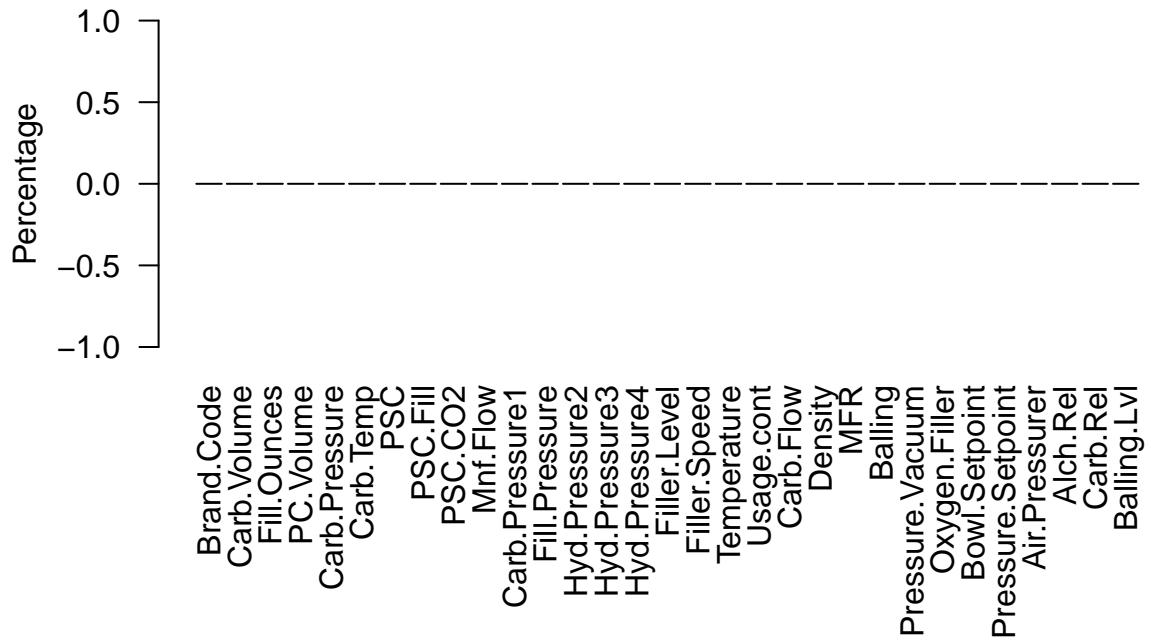
#Impute with MICE:
imputed_new_test_data<- mice(new_test_data, m=5, maxit=50, meth='pmm', seed=100, printFlag = FALSE)
#Select the first iteration:
imputed_new_test_data<- complete(imputed_new_test_data, 1)

#Visualize the data again for missing values:
par(mar = c(10, 4, 4, 2))

imputed_new_test_data %>%
  sapply(function(x) sum(is.na(x)/length(x)*100)) %>%
  barplot(main = "Percentage of Missing Values",
          ylab = "Percentage",
          col = "lightblue",
          las = 2)

```

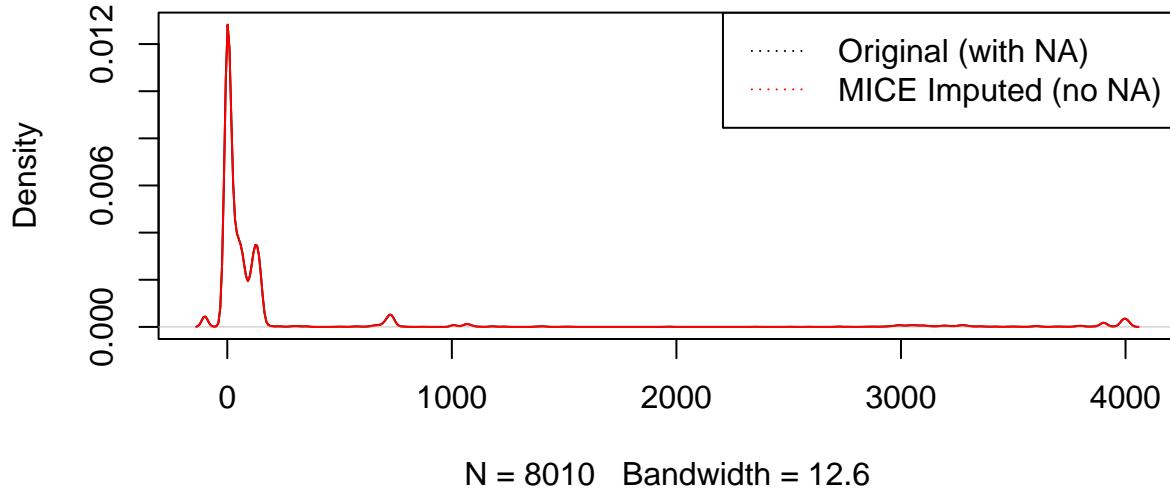
Percentage of Missing Values



```
#All missing data are imputed

#Checking if the imputation creates bias:
plot(density(unlist(imputed_new_test_data[, -1]), na.rm=TRUE),
      main='MICE imputed test data vs original data w/ NA removed')
lines(density(unlist(imputed_new_test_data[, -1]), na.rm=TRUE),
      col = 'red')
legend('topright', legend = c("Original (with NA)", "MICE Imputed (no NA)"),
       col = c("black", "red"), lty = 3)
```

MICE imputed test data vs original data w/ NA removed



```
#Also use T-test to check for mean bias:  
t.test(new_test_data[, -1], imputed_new_test_data[, -1])
```

```
##  
## Welch Two Sample t-test  
##  
## data: new_test_data[, -1] and imputed_new_test_data[, -1]  
## t = -0.093452, df = 15916, p-value = 0.9255  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -25.99571 23.62973  
## sample estimates:  
## mean of x mean of y  
## 254.6245 255.8075
```

#P-value >> 0.05, indicating no statistical difference in mean between the before and after imputed data

Data prediction and exportation:

```
rf_ph_result<- predict(rf_model_mice, imputed_new_test_data)  
  
#Combine the pH values with the new_test_data:  
imputed_new_test_data$pH <- rf_ph_result
```

```
#Exporting results:  
write_xlsx(imputed_new_test_data, 'predicted_pH.xlsx')
```