Beverage PH Machine Learning

Angel Claudio

August 2020

Proposal and Goal

The goal of this project is to use a data set from a beverage manufacturing company consisting of 2,571 observations and 33 variables to train a model. The model will be used to predict pH in sample data provided separately.

Import Data and EDA (Exploratory Data Analysis)

We will begin by exploring the original data and investigating the type and distribution of all the features available. Due to the nature of this project, scaling and centering is an inappropriate method for model training as the model must be used to predict values in a separate set of provided data. Conversions of the final predictions would be necessary (and impossible to make accurately) with a model trained on centered and scaled data.

Import Data

Approach: In order to make this a reproducible example, we use the *download.file* function from *utils* to download a public hosted file and read it into memory. To ensure we write the file in binary for use, we set the argument *mode* with value **wb** (writing in binary mode). Since this is an excel file, we use the *read_excel* function from the *readxl* library to read the original data into memory.

```
url_data <- paste0('https://github.com/AngelClaudio/data-sources/blob/master/',
            'excel/StudentData%20-%20TO%20MODEL.xls?raw=true')

download.file(url_data, "temp.xls", mode = "wb")

original_data <- read_excel("temp.xls", )
```

Interpretation: Running the program will create a local copy of the data on the working directory. There is no dependency on a physical file to run this R Markdown file.

Initial Look of Data Structure and Content

Approach: We use the base function *str* to explore the structure of the data, data types, data size, and preview some data samples per variable.

```
str(original_data)
```

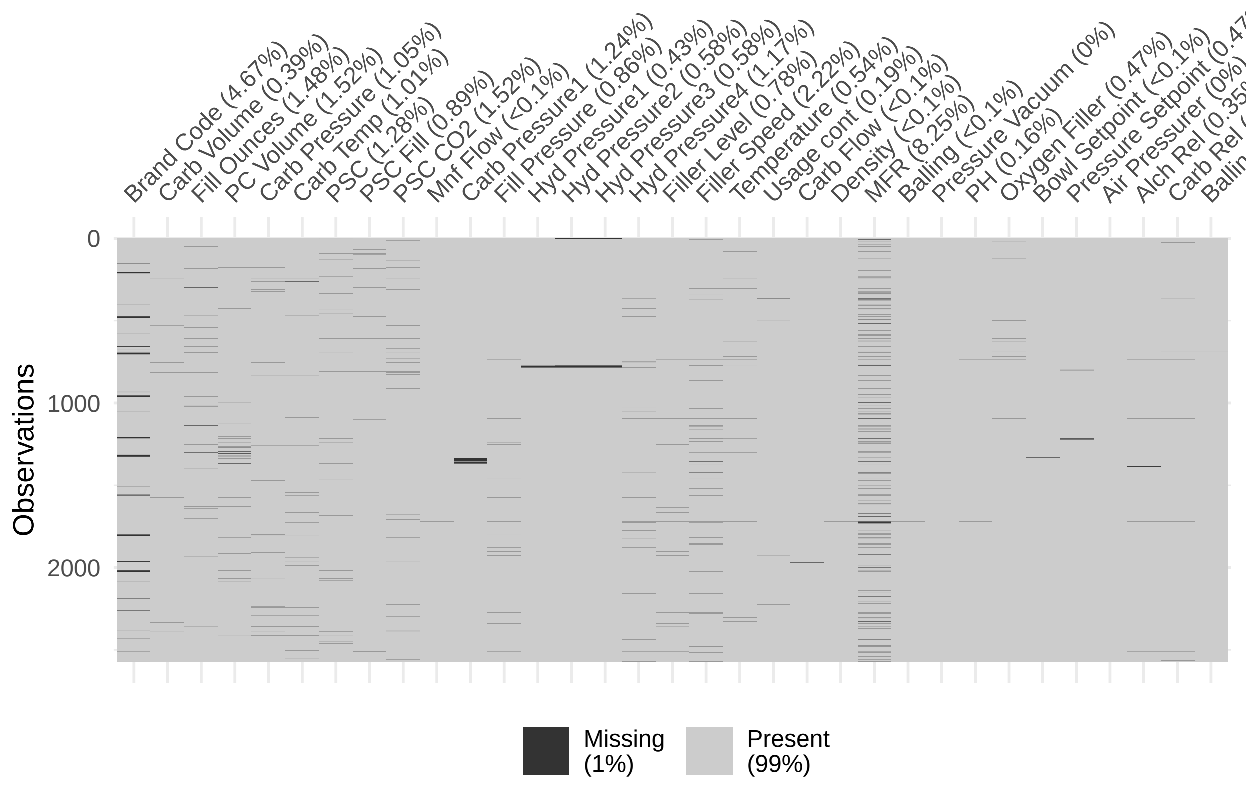
```
## tibble [2,571 x 33] (S3: tbl_df/tbl/data.frame)
##  $ Brand Code       : chr [1:2571] "B" "A" "B" "A" ...
##  $ Carb Volume      : num [1:2571] 5.34 5.43 5.29 5.44 5.49 ...
##  $ Fill Ounces      : num [1:2571] 24 24 24.1 24 24.3 ...
##  $ PC Volume        : num [1:2571] 0.263 0.239 0.263 0.293 0.111 ...
##  $ Carb Pressure    : num [1:2571] 68.2 68.4 70.8 63 67.2 66.6 64.2 67.6 64.2 72 ...
##  $ Carb Temp        : num [1:2571] 141 140 145 133 137 ...
##  $ PSC              : num [1:2571] 0.104 0.124 0.09 NA 0.026 0.09 0.128 0.154 0.132 0.014 ...
##  $ PSC Fill         : num [1:2571] 0.26 0.22 0.34 0.42 0.16 ...
##  $ PSC CO2          : num [1:2571] 0.04 0.04 0.16 0.04 0.12 ...
##  $ Mnf Flow         : num [1:2571] -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...
##  $ Carb Pressure1   : num [1:2571] 119 122 120 115 118 ...
##  $ Fill Pressure    : num [1:2571] 46 46 46 46.4 45.8 45.6 51.8 46.8 46 45.2 ...
##  $ Hyd Pressure1    : num [1:2571] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd Pressure2    : num [1:2571] NA NA NA 0 0 0 0 0 0 0 ...
##  $ Hyd Pressure3    : num [1:2571] NA NA NA 0 0 0 0 0 0 0 ...
##  $ Hyd Pressure4    : num [1:2571] 118 106 82 92 92 116 124 132 90 108 ...
##  $ Filler Level     : num [1:2571] 121 119 120 118 119 ...
##  $ Filler Speed     : num [1:2571] 4002 3986 4020 4012 4010 ...
##  $ Temperature      : num [1:2571] 66 67.6 67 65.6 65.6 66.2 65.8 65.2 65.4 66.6 ...
##  $ Usage cont       : num [1:2571] 16.2 19.9 17.8 17.4 17.7 ...
##  $ Carb Flow        : num [1:2571] 2932 3144 2914 3062 3054 ...
##  $ Density          : num [1:2571] 0.88 0.92 1.58 1.54 1.54 1.52 0.84 0.84 0.9 0.9 ...
##  $ MFR              : num [1:2571] 725 727 735 731 723 ...
##  $ Balling          : num [1:2571] 1.4 1.5 3.14 3.04 3.04 ...
##  $ Pressure Vacuum  : num [1:2571] -4 -4 -3.8 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 ...
##  $ PH               : num [1:2571] 8.36 8.26 8.94 8.24 8.26 8.32 8.4 8.38 8.38 8.5 ...
##  $ Oxygen Filler    : num [1:2571] 0.022 0.026 0.024 0.03 0.03 0.024 0.066 0.046 0.064 0.022 ...
##  $ Bowl Setpoint    : num [1:2571] 120 120 120 120 120 120 120 120 120 120 ...
##  $ Pressure Setpoint: num [1:2571] 46.4 46.8 46.6 46 46 46 46 46 46 46 ...
##  $ Air Pressurer    : num [1:2571] 143 143 142 146 146 ...
##  $ Alch Rel         : num [1:2571] 6.58 6.56 7.66 7.14 7.14 7.16 6.54 6.52 6.52 6.54 ...
##  $ Carb Rel         : num [1:2571] 5.32 5.3 5.84 5.42 5.44 5.44 5.38 5.34 5.34 5.34 ...
##  $ Balling Lvl      : num [1:2571] 1.48 1.56 3.28 3.04 3.04 3.02 1.44 1.44 1.44 1.38 ...
```

Interpretation: We can see that all the variables are numeric except `Brand Code`. We'll convert the variable `Brand Code` to *dummy variables* so that it can be quantitative and used for analysis and fitting a model.

Analysis of Missing Data

Approach: The following code chunk creates a missingness map and bar chart of missing value percentages to investigate whether there are any patterns in the missing values.

```r
vis_miss(original_data)
```

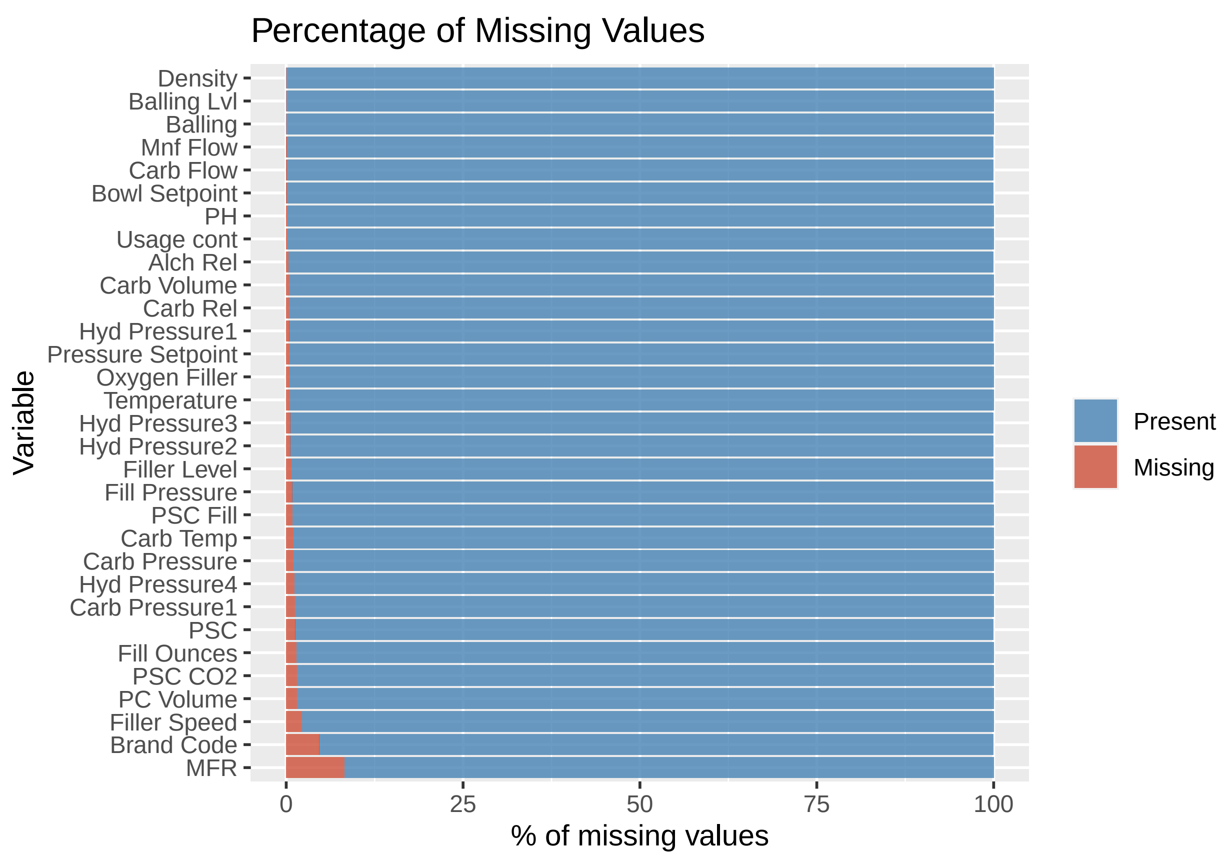


```
missing.values <- original_data %>%
  gather(key = "key", value = "val") %>%
  mutate(isna = is.na(val)) %>%
  group_by(key) %>%
  mutate(total = n()) %>%
  group_by(key, total, isna) %>%
  summarise(num.isna = n()) %>%
  mutate(pct = num.isna / total * 100)


levels <-
    (missing.values  %>% filter(isna == T) %>% arrange(desc(pct)))$key

percentage.plot <- missing.values %>%
      ggplot() +
        geom_bar(aes(x = reorder(key, desc(pct)),
                     y = pct, fill=isna),
                 stat = ’identity’, alpha=0.8) +
      scale_x_discrete(limits = levels) +
      scale_fill_manual(name = "",
                        values = c(’steelblue’, ’tomato3’), labels = c("Present", "Missing")) +
      coord_flip() +
      labs(title = "Percentage of Missing Values", x =
             ’Variable’, y = "% of missing values")

percentage.plot
```



Interpretation: MFR has the largest percentage of missing values (about 8%). Since this is a small amount relative to the total size of the dataset, we will use an imputation method to fill all of the missing values.

Check for Zero Variance

Approach: We want to check variables for near zero variance. We can do this by using the *nearZeroVar* function of the *caret* package.

```
nzv <- nearZeroVar(original_data, saveMetrics= TRUE)
nzv[nzv$nzv,]
```

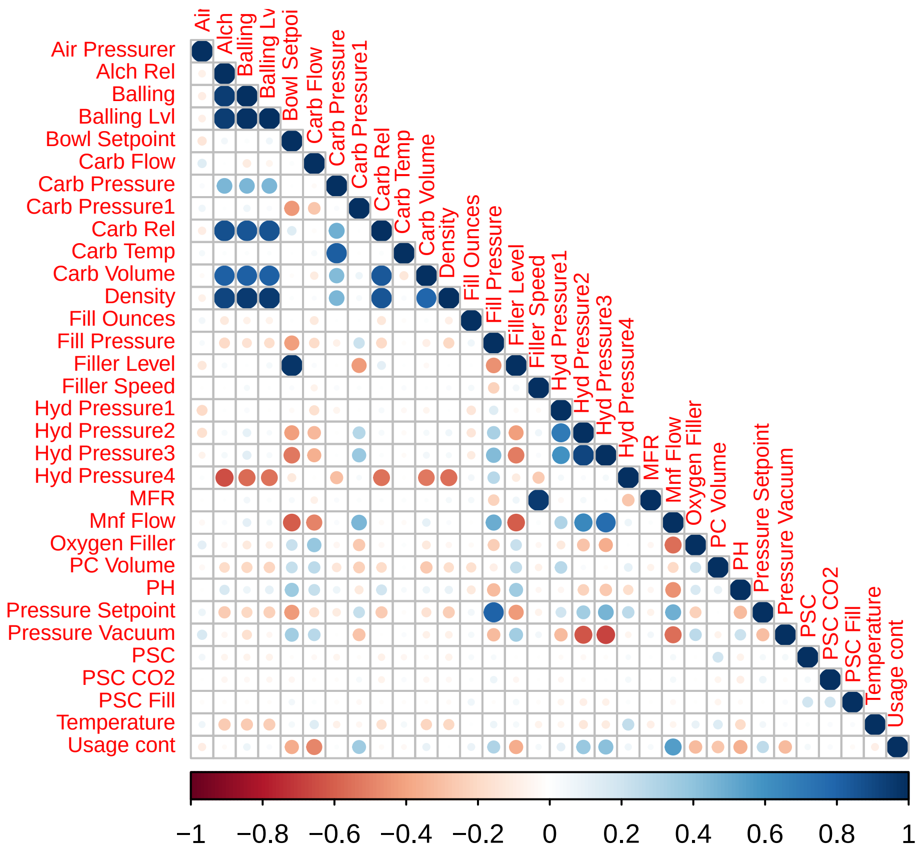
```
##               freqRatio percentUnique zeroVar  nzv
## Hyd Pressure1  31.11111      9.529366   FALSE TRUE
```

Interpretation: We can see that we found *Hyd Pressur1* to be labeled *TRUE* for pre-processing of near-zero variance.

Checking Correlation

Approach: We use the *corrplot* function to display a graphical correlation matrix.

```
corrplot(cor(original_data[,-1], use = "na.or.complete"), type="lower",
         order="alphabet", tl.cex=.7)
```

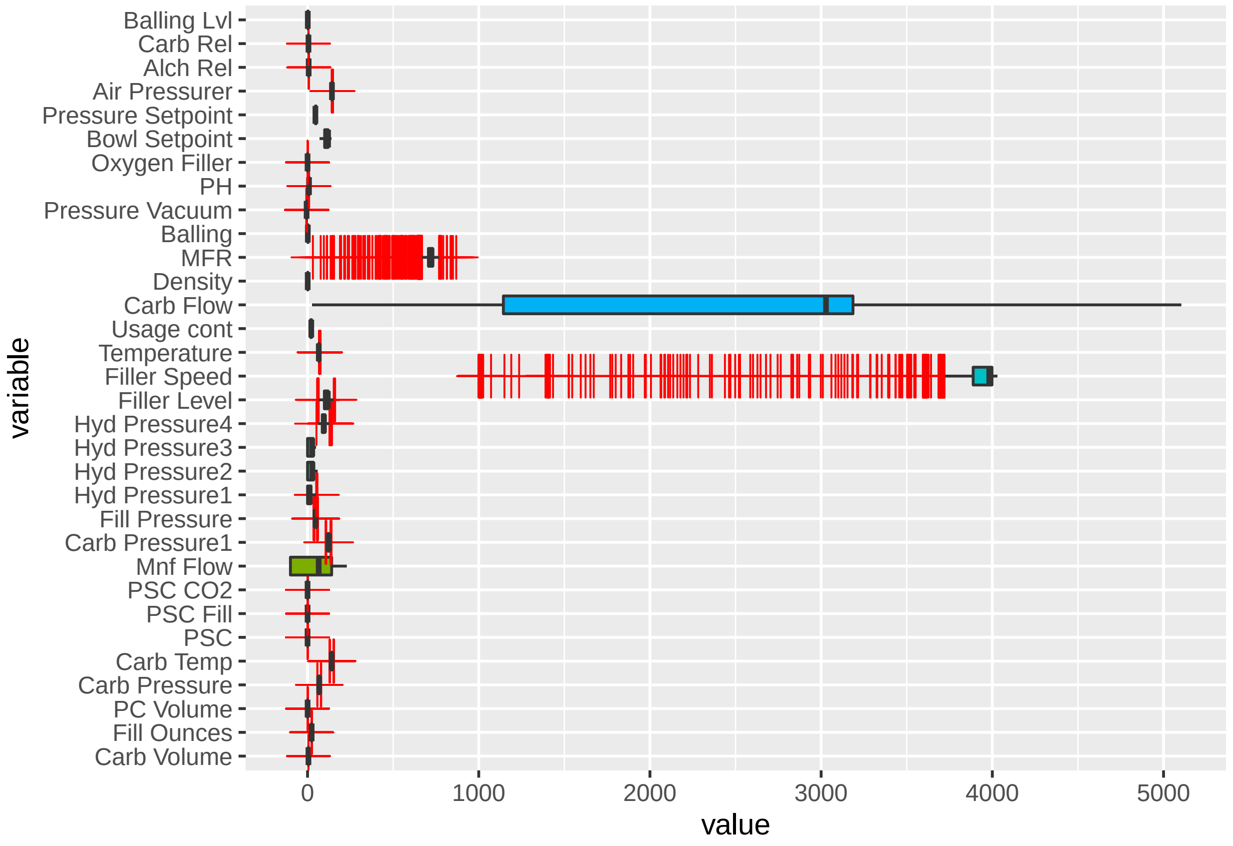


Interpretation: Overall there seems to be low correlation, but there are some areas of concern. For example, we do see problematic areas with Balling Level and Balling across the Carb Rel, Carb Volume, and Density features.

Checking for Outliers and Magnitude

Approach: We use the functions *ggplot* and *geom_boxplot* to identify outliers and the scale of the data. We use *melt* function on the data to pass the data in a long format.

```r
ggplot(data = reshape2::melt(original_data) , aes(x=variable, y=value)) +
  geom_boxplot(outlier.colour="red", outlier.shape=3, outlier.size=5,
               aes(fill=variable)) +
  coord_flip() +
  theme(legend.position="none")
```



Interpretation: We can see that the variables Carb Flow, Filler Speed, and MFR have values far exceeding the values of the other variables.

Data Transformation

Now that we have an idea about the data we are working with, we will begin preliminary transformation of the data to prepare for model creation and training.

Column Transformations

Approach: We first alter the column names by removing spaces using the base function *str_replace* to facilitate general data management.

```r
# Remove spaces from all column names
transformed_data <- original_data
colnames(transformed_data) <- str_replace(colnames(original_data), '\\s', '')
colnames(transformed_data)
```

```
##  [1] "BrandCode"       "CarbVolume"      "FillOunces"      "PCVolume"
##  [5] "CarbPressure"    "CarbTemp"        "PSC"             "PSCFill"
##  [9] "PSCCO2"          "MnfFlow"         "CarbPressure1"   "FillPressure"
## [13] "HydPressure1"    "HydPressure2"    "HydPressure3"    "HydPressure4"
## [17] "FillerLevel"     "FillerSpeed"     "Temperature"     "Usagecont"
## [21] "CarbFlow"        "Density"         "MFR"             "Balling"
## [25] "PressureVacuum"  "PH"              "OxygenFiller"    "BowlSetpoint"
## [29] "PressureSetpoint" "AirPressurer"     "AlchRel"          "CarbRel"
## [33] "BallingLvl"
```

Interpretation: We can see from the output that all the column names have been cleansed of spaces!

Handling NAs

Approach: We will drop NAs from the response variable using *drop_na* function.

```
transformed_data <- drop_na(transformed_data, PH)
```

Interpretation: All of the variables have missing values except for PressureVacuum and AirPressure. The response variable has 4 missing values. These observations are removed from the dataset as they constitute an less than 1% of the sample.

Handling NZV

Approach: We have identified during our EDA that *Hyd Pressur1* has near-zero variance. However, we will include the near-zero variance variables for the purposes of `gbm` as omission will yield no benefit.

```
data_for_modeling <- transformed_data#[,-which(colnames(transformed_data)=='PH')]
```

Dummy Variables

To ensure accuracy, we computed dummy variables manually and removed the categorical variable *BrandCode*:

```
data_for_modeling$BrandA <- 0
data_for_modeling$BrandA[which(data_for_modeling$BrandCode=='A')] <- 1
data_for_modeling$BrandB <- 0
data_for_modeling$BrandB[which(data_for_modeling$BrandCode=='B')] <- 1
data_for_modeling$BrandC <- 0
data_for_modeling$BrandC[which(data_for_modeling$BrandCode=='C')] <- 1
data_for_modeling$BrandD <- 0
data_for_modeling$BrandD[which(data_for_modeling$BrandCode=='D')] <- 1
data_for_modeling$BrandNA <- 0
data_for_modeling$BrandNA[which(is.na(data_for_modeling$BrandCode)==TRUE)] <- 1

data_for_modeling$BrandCode <- NULL
```

Imputation: `mice` and `parlmice`

There are missing data in both provided datasets; in order to minimize error in the predictions and replicate imputation as closely as possible, `PH` is omitted in the the training data before imputation. We imputed missing data using `mice`, which creates 5 sets of imputed data by default. The *seed* option within the command ensures reproducibility. Density plots for a few imputed variables allows us to assess whether the imputation method is appropriate. We will not remove columns of near-zero variance as they have no detrimental effect on the function of our model.

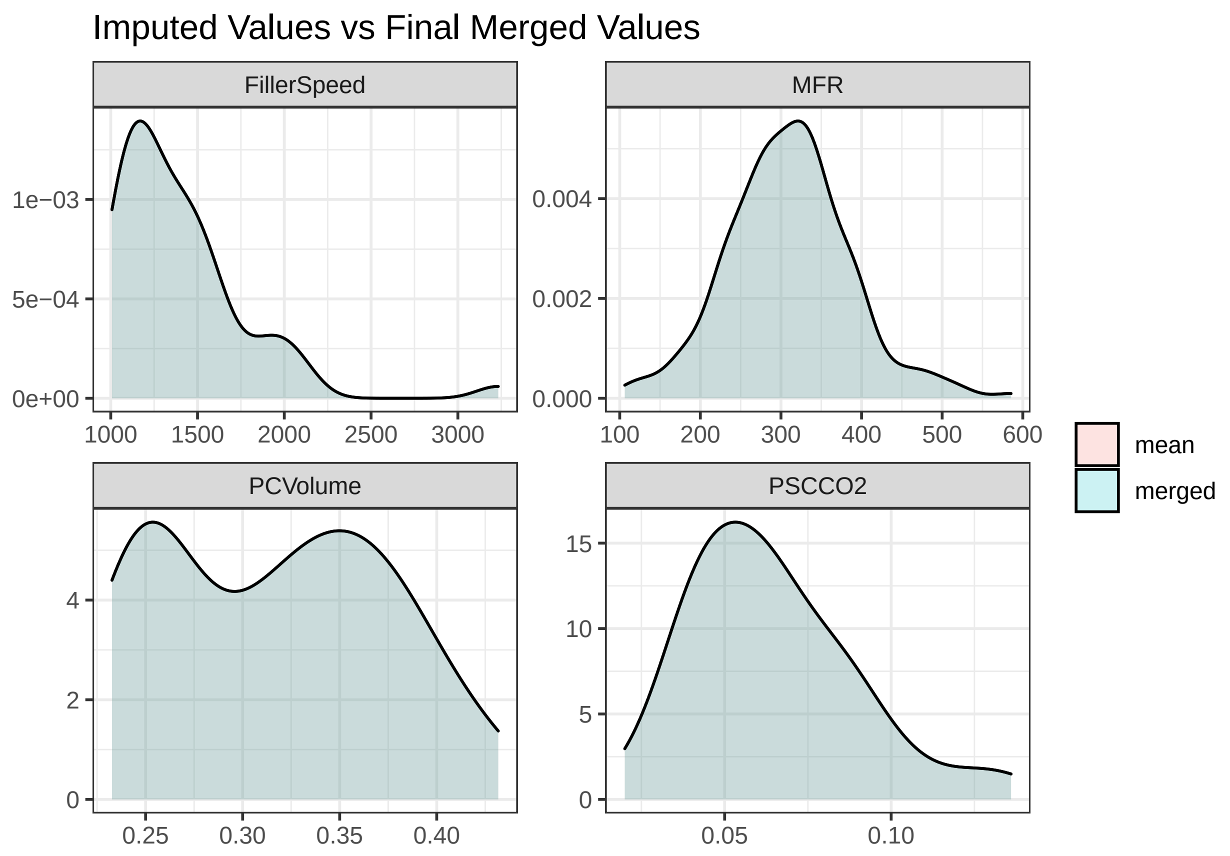
Due to the amount of processing time involved in the `mice` command, `parlmice` is used to speed processing with the addition of the *n.core* parameter.

```
td_noPH <- data_for_modeling[,-which(colnames(data_for_modeling)=='PH')]

# -2 CORES FOR SAFETY
number_of_cores <- detectCores() - 2
```

```r
# USING PARLMICE INSTEAD TO LEVERAGE PARALLEL PROCESSING
df2_imp <- parlmice(data = td_noPH, m = 5, method = "pmm",
                    n.core = number_of_cores, maxit = 50, seed = 500, print = FALSE)

# plot density of imputed values
plot_merge <- merge_imputations(
              td_noPH,
              df2_imp,
              summary = c("dens"),
              filter = c("PSCCO2","PCVolume","FillerSpeed","MFR")
              )
plot_merge$plot[9]$labels$title = 'Imputed Values vs Final Merged Values'
plot_merge
```



Interpretation: The plot "shows the distribution of the mean of the imputed values for each variable at each observation. The larger the areas overlap, the better is the fit of the merged value compared to the imputed value." (Source: https://www.rdocumentation.org/packages/sjmisc/versions/2.8.4/topics/merge_imputations)

According to our EDA, the variables *PSCCO2*, *PCVolume*, *FillerSpeed*, and *MFR* had the highest proportion of missing values and provide the best indication of whether the imputation approach was appropriate. In the plots above, the regions defined by the mean and the merged values overlap exactly, indicating consistency among the imputed values for each observation across all 5 resulting dataframes yielded by imputation.

This strengthens our confidence in the imputed values. In the code chunk below, the mean of the imputed values across the 5 imputed datasets are appended to the data. The columns with missing data are dropped. This concludes the imputation process. We will replicate this process to impute missing data in the dataset provided for final predictions.

```r
NAnames <- names(td_noPH)[sapply(td_noPH, anyNA)]
td_noPH2 <- td_noPH[,-which(names(td_noPH) %in% NAnames)]

data_for_modeling2 <- merge_imputations(td_noPH,df2_imp,td_noPH2)

data_for_modeling2$PH <- transformed_data$PH
```

Training and Tuning the Model: Partitioned, Imputed Data

First, data were partitioned into training and testing dataframes, using 70% of the data to train and 30% of the data to test the model. Testing the model is necessary to assess predictive accuracy.

```r
set.seed(500) #to get repeatable data

train2 <- sample_frac(data_for_modeling2, 0.7)
train2_index <- as.numeric(rownames(train2))
test2 <- data_for_modeling2[-train2_index, ]

y2 <- test2$PH
```

Benchmark Models

We built two Benchmark models (1) Linear Regression - OLS and (2) Decision Tree - Regression. These models will enable us to compare model performance to select the most appropriate for our prediction.

Approach: The first Benchmark model is an ordinary least squares model - the Linear Regression.

The training data has 1797 observations and 37 variables while the test data has 770 observations with 37 variables.

Linear Regression - OLS

```r
RegModel <-lm(PH ~ ., data = train2)
summary(RegModel)
```

```
##
## Call:
## lm(formula = PH ~ ., data = train2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.54607 -0.07752  0.01088  0.08641  0.84893
##
## Coefficients: (1 not defined because of singularities)
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             1.069e+01  1.317e+00   8.116 8.91e-16 ***
## MnfFlow                -7.421e-04  5.559e-05 -13.349  < 2e-16 ***
## Density                -1.139e-01  3.401e-02  -3.350 0.000824 ***
## Balling                -7.394e-02  2.810e-02  -2.632 0.008569 **
## PressureVacuum         -1.972e-02  9.366e-03  -2.105 0.035427 *
## AirPressurer           -1.515e-03  2.825e-03  -0.536 0.591797
## BrandA                2.110e-02  2.930e-02   0.720 0.471647
## BrandB                7.892e-02  1.650e-02   4.783 1.87e-06 ***
## BrandC               -7.480e-02  1.804e-02  -4.147 3.53e-05 ***
## BrandD                6.719e-02  3.311e-02   2.029 0.042578 *
## BrandNA                      NA         NA      NA       NA
## CarbVolume_imp       -1.155e-01  1.099e-01  -1.051 0.293424
## FillOunces_imp       -7.622e-02  3.907e-02  -1.951 0.051217 .
## PCVolume_imp         -1.026e-01  6.649e-02  -1.543 0.122956
## CarbPressure_imp      2.622e-03  5.300e-03   0.495 0.620865
## CarbTemp_imp         -1.288e-03  4.152e-03  -0.310 0.756441
## PSC_imp              -7.084e-02  6.901e-02  -1.027 0.304761
## PSCFill_imp          -3.988e-02  2.845e-02  -1.402 0.161089
## PSCCO2_imp           -8.199e-02  7.698e-02  -1.065 0.286959
## CarbPressure1_imp     7.660e-03  8.356e-04   9.167  < 2e-16 ***
## FillPressure_imp      1.463e-03  1.497e-03   0.978 0.328308
## HydPressure1_imp      3.033e-04  4.394e-04   0.690 0.490042
## HydPressure2_imp     -1.570e-03  6.314e-04  -2.487 0.012968 *
## HydPressure3_imp      3.759e-03  7.127e-04   5.275 1.49e-07 ***
## HydPressure4_imp     -1.269e-04  4.047e-04  -0.314 0.753862
## FillerLevel_imp      -8.253e-04  6.944e-04  -1.188 0.234801
## FillerSpeed_imp       1.707e-05  1.600e-05   1.066 0.286390
## Temperature_imp      -1.310e-02  2.832e-03  -4.627 3.99e-06 ***
## Usagecont_imp        -7.625e-03  1.389e-03  -5.491 4.58e-08 ***
## CarbFlow_imp          1.252e-05  4.544e-06   2.755 0.005923 **
## MFR_imp              -1.183e-04  9.664e-05  -1.224 0.221187
## OxygenFiller_imp     -3.909e-01  8.563e-02  -4.564 5.36e-06 ***
## BowlSetpoint_imp      3.059e-03  7.191e-04   4.253 2.22e-05 ***
## PressureSetpoint_imp -9.944e-03  2.375e-03  -4.186 2.98e-05 ***
## AlchRel_imp           5.969e-02  2.403e-02   2.484 0.013076 *
## CarbRel_imp           6.081e-02  5.654e-02   1.075 0.282326
## BallingLvl_imp        9.858e-02  2.547e-02   3.870 0.000113 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1337 on 1761 degrees of freedom
## Multiple R-squared:  0.4352, Adjusted R-squared:  0.424
## F-statistic: 38.77 on 35 and 1761 DF,  p-value: < 2.2e-16
```

Interpretation:

Although we have a significant p-value, the R-squared shows that the model is only accounting for 42% of the variation in the data.

Variable Significance

```
set.seed (124)
variableimp <- as.data.frame(varImp(RegModel))
variableimp <- data.frame(overall = variableimp$Overall,
           names   = rownames(variableimp))
variableimp[order(variableimp$overall,decreasing = T),]
```

```
##       overall                names
## 1  13.3489911              MnfFlow
## 18  9.1667800    CarbPressure1_imp
## 27  5.4909896        Usagecont_imp
## 22  5.2752185     HydPressure3_imp
## 7   4.7831768               BrandB
## 26  4.6266164      Temperature_imp
## 30  4.5642709     OxygenFiller_imp
## 31  4.2531807     BowlSetpoint_imp
## 32  4.1861496 PressureSetpoint_imp
## 8   4.1466016               BrandC
## 35  3.8701759       BallingLvl_imp
## 2   3.3503691              Density
## 28  2.7553799         CarbFlow_imp
## 3   2.6317241              Balling
## 21  2.4871844     HydPressure2_imp
## 33  2.4842161          AlchRel_imp
## 4   2.1050699       PressureVacuum
## 9   2.0292995               BrandD
## 11  1.9509834       FillOunces_imp
## 12  1.5432248         PCVolume_imp
## 16  1.4020076          PSCFill_imp
## 29  1.2238121              MFR_imp
## 24  1.1884889      FillerLevel_imp
## 34  1.0754371          CarbRel_imp
## 25  1.0663971      FillerSpeed_imp
## 17  1.0651381           PSCCO2_imp
## 10  1.0509527       CarbVolume_imp
## 15  1.0265758              PSC_imp
## 19  0.9777997     FillPressure_imp
## 6   0.7199566               BrandA
## 20  0.6903868     HydPressure1_imp
## 5   0.5363319         AirPressurer
## 13  0.4947128     CarbPressure_imp
## 23  0.3135994     HydPressure4_imp
## 14  0.3102064         CarbTemp_imp
```

Interpretation:

The variable significance in descending order shows that *MnfFlow*, *CarbPressure1_imp*, *Usagecont_imp*, *HydPressure3_imp*, *BrandB*, *Temperature_imp*, *OxygenFiller_imp*, *BowlSetpoint_imp*, *PressureSetpoint_imp*, *BrandC*, *BallingLvl_imp*, and *Density* are the top 12 important variables in predicting *PH*. This information will guide us in our subsequent approaches in modeling.

Please Note: The '_imp' are suffixes created from the transformation process. These suffixes will be removed prior to calculating the model predictions.

Improved Linear Regression - OLS

Approach:

We will improve the initial OLS using the 12 significant variables identified by the first model to see if the R-squared may improve.

```
RegModel2 <-lm(PH ~ MnfFlow + CarbPressure1_imp + Usagecont_imp + HydPressure3_imp + BrandB + Temperatu
summary(RegModel2)
```

```
##
## Call:
## lm(formula = PH ~ MnfFlow + CarbPressure1_imp + Usagecont_imp +
##     HydPressure3_imp + BrandB + Temperature_imp + OxygenFiller_imp +
##     BowlSetpoint_imp + PressureSetpoint_imp + BrandC + BallingLvl_imp +
##     Density, data = train2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.56259 -0.07794  0.01210  0.08757  0.90024
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           8.907e+00  2.187e-01  40.725  < 2e-16 ***
## MnfFlow              -7.903e-04  5.366e-05 -14.727  < 2e-16 ***
## CarbPressure1_imp     7.016e-03  7.644e-04   9.179  < 2e-16 ***
## Usagecont_imp        -9.980e-03  1.311e-03  -7.615 4.26e-14 ***
## HydPressure3_imp      2.880e-03  3.207e-04   8.979  < 2e-16 ***
## BrandB                6.966e-02  1.433e-02   4.863 1.26e-06 ***
## Temperature_imp      -1.265e-02  2.607e-03  -4.852 1.33e-06 ***
## OxygenFiller_imp     -3.733e-01  8.587e-02  -4.347 1.46e-05 ***
## BowlSetpoint_imp      1.835e-03  2.809e-04   6.533 8.41e-11 ***
## PressureSetpoint_imp -9.661e-03  1.908e-03  -5.063 4.55e-07 ***
## BrandC               -8.846e-02  1.570e-02  -5.635 2.03e-08 ***
## BallingLvl_imp        7.766e-02  1.305e-02   5.952 3.18e-09 ***
## Density              -1.047e-01  2.805e-02  -3.731 0.000197 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1363 on 1784 degrees of freedom
## Multiple R-squared:  0.4054, Adjusted R-squared:  0.4014
## F-statistic: 101.4 on 12 and 1784 DF,  p-value: < 2.2e-16
```

Performance Metrics - OLS

```r
OLSMetrics <- data.frame(
  R2 = rsquare(RegModel2, data = train2),
  RMSE = rmse(RegModel2, data = train2),
  MAE = mae(RegModel2, data = train2)
)
print(OLSMetrics)
```

```
##          R2      RMSE       MAE
## 1 0.4053746 0.1358043 0.1049277
```

Interpretation:

Although the RMSE and the MAE are low, the improved model using the 12 significant variables did not improve the R-squared as can been seen from the performance metrics above.

Decision Tree - Regression

The next Benchmark model is the Decision Tree - Regression Tree using the same variables.

Approach:

We will use the `rpart` library for the decision tree and try to identify any significant variables in building the tree.

```
Decisiontree <- rpart(PH ~., method = "anova", data = train2)

printcp(Decisiontree) # display the results
```

```
##
## Regression tree:
## rpart(formula = PH ~ ., data = train2, method = "anova")
##
## Variables actually used in tree construction:
##  [1] AirPressurer      AlchRel_imp       BowlSetpoint_imp  BrandC
##  [5] CarbPressure1_imp CarbRel_imp       HydPressure3_imp  MnfFlow
##  [9] OxygenFiller_imp  PressureVacuum    Temperature_imp   Usagecont_imp
##
## Root node error: 55.735/1797 = 0.031016
##
## n= 1797
##
##          CP nsplit rel error  xerror     xstd
## 1  0.227829      0   1.00000 1.00118 0.034489
## 2  0.065956      1   0.77217 0.77391 0.031313
## 3  0.036455      2   0.70621 0.70974 0.029406
## 4  0.030934      3   0.66976 0.68501 0.028438
## 5  0.025231      4   0.63883 0.64795 0.028214
## 6  0.018045      5   0.61359 0.62586 0.027168
## 7  0.016150      6   0.59555 0.62735 0.027792
## 8  0.015959      8   0.56325 0.61414 0.026274
## 9  0.013408      9   0.54729 0.59741 0.025634
## 10 0.012404     10   0.53388 0.58329 0.025639
## 11 0.012022     11   0.52148 0.57557 0.025548
## 12 0.011940     12   0.50946 0.57412 0.025529
## 13 0.011744     14   0.48558 0.57264 0.025648
## 14 0.011556     15   0.47383 0.56630 0.025753
## 15 0.010488     16   0.46228 0.54848 0.025260
## 16 0.010000     17   0.45179 0.53619 0.025003
```

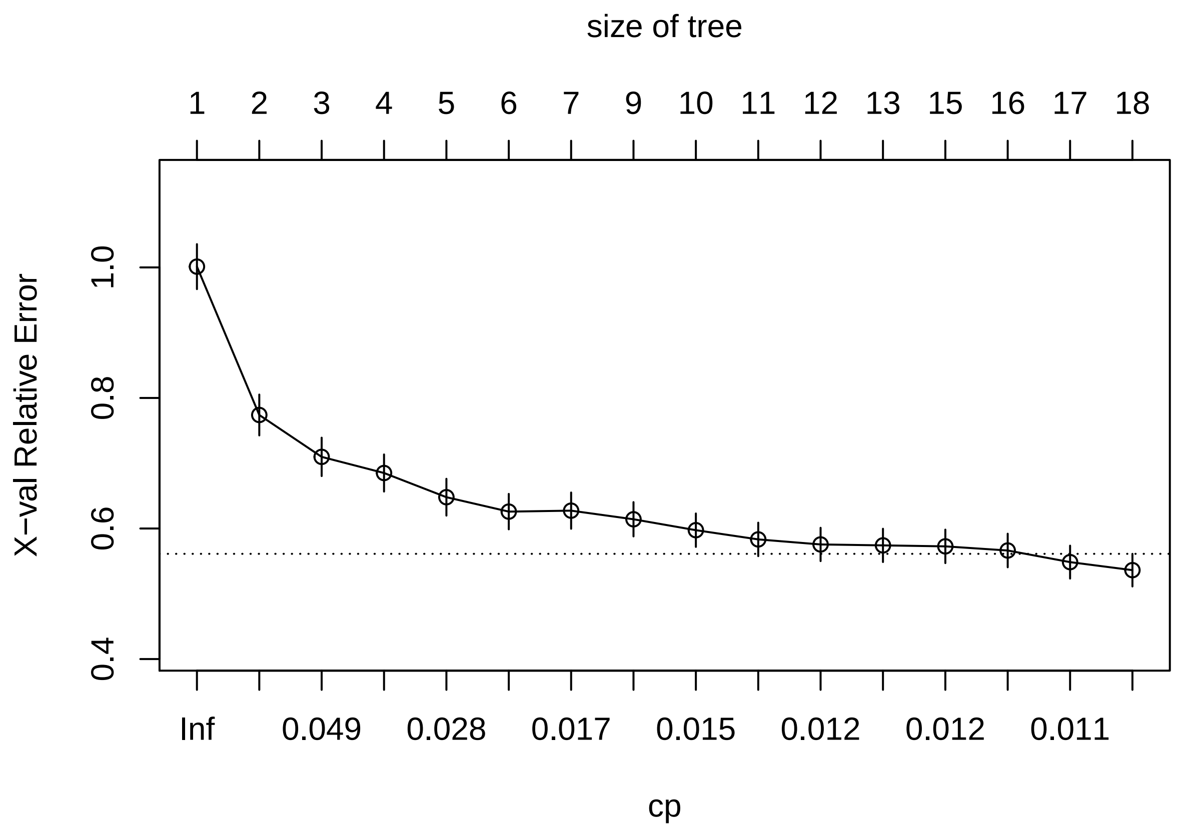
Interpretation:

The decison tree identified the following variables as significant for building the tree: *AirPressurer*, *AlchRel_imp*, *BowlSetpoint_imp*, *BrandC*, *CarbPressure1_imp*, *CarbRel_imp*, *HydPressure3_imp*, *MnfFlow*, *OxygenFiller_imp*, *PressureVacuum*, *Temperature_imp* and *Usagecont_imp*. The "_imp" suffix denotes variables that have imputed values.

The root node error is about 3% which means that the splitting at the root node is 97% accurate.

Visualize cross-validation results

```
plotcp(Decisiontree)
```

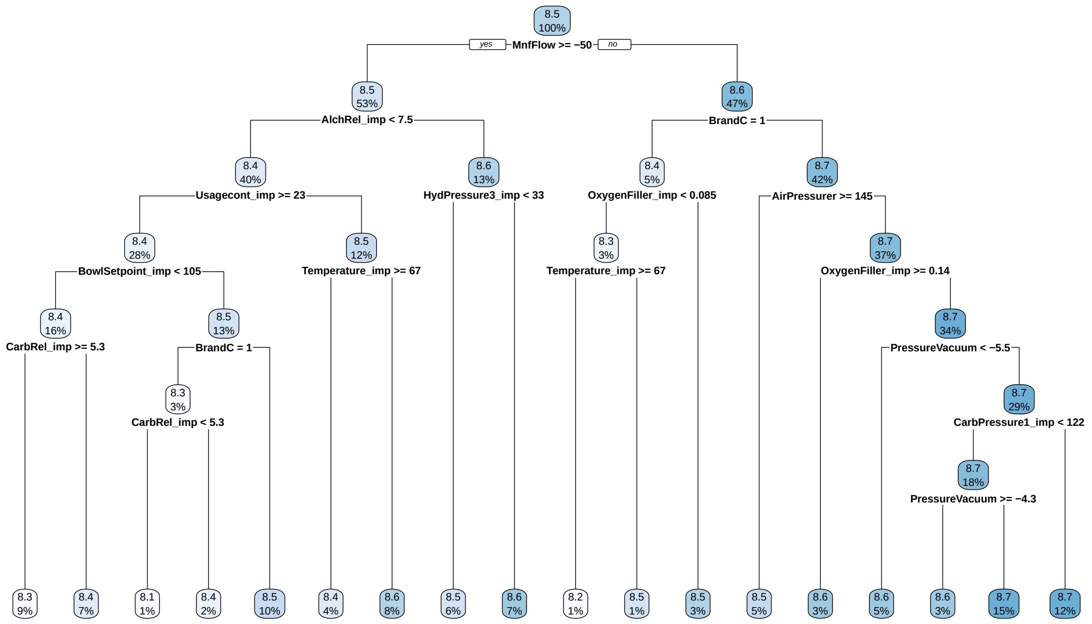


Interpretation:

As the complexity parameter (CP) decreases, we can see that the relative errors equally decrease.

Plot the Tree

```
rpart.plot(Decisiontree, extra = "auto",fallen.leaves = TRUE, box.palette = "auto")
```



Interpretation:

From the plot of the Regression Tree, we can see the the splitting started from the Root Node, *MnfFlow*, as the most significant variable for predicting *PH* followed by *AlchRel* and *BrandC*.

Although the Decision Tree is easy to understand and interpret, they are prone to overfitting. That's why other tree based models such as Random Forest and Gradient Boosting are preferred.

Gradient Boost Modeling: `gbm`

One of the fundamental reasons for Gradient Boost Modeling was to solve the overfitting issue of decision tree. GBM avoids overfitting by attempting to automatically select the inflection point where performance on the test dataset starts to decrease while performance on the training dataset continues to improve (Singh, 2018).

In the context of GBM, early stopping can be based either on an out of bag sample set ("OOB") or cross-validation ("CV"). To avoid overfitting, the GBM stops training the model when the validation error has decreased and stabilized right before the error starts to increase.

The biggest motivations of using Gradient Boosting is that it allows us to optimize a specified cost function, instead of a loss function that usually offers less control and does not essentially correspond with real world applications.

Strategy

Approach: We used 70% of the training data provided to train several different models; accuracy of each model was tested against the remaining 30% of the training data. Ordinary linear regression using the `lm` command was least accurate. Among `caretEnsemble` methods, only `rpart` (recursive partitioning) and `svmRadial` (support vector machines) provided satisfactory predictions, though tuning of those models did not outperform the tuned 'glm' model.

We include the code and results for the tuned 'glm' model below. Once the model is tuned to optimize accuracy in the imputed and partitioned test data, the final model is trained on the entire imputed dataset. We then impute missing data in the data provided for submission and run the final model on the imputed submission data to make our final predictions.

The following code chunk trains the model. Tuning is accomplished by employing `caret` to train the model using a range of values for each parameter. The *method* parameter is set to compare values using the repeated cross-validation method (*"repeatedcv"*) and repeat each run 5 times with 5 iterations. Values that yielded the least Root Mean Squared Error (RMSE) were selected.

This process is exceedingly lengthy and the individual steps are omitted with the final values included in the code below. First, interaction depth (*interaction.depth*), which signifies the overall number of divisions within the data, is tested with other values set as default.

Each division represents a layer of clustered data for which the model fit did not meet a threshold of accuracy in the node, or cluster, before it. Tuning proceeds with another parameter, *n.minobsinnode*, which sets the minimum number of observations per node. Once this value is tested over a range of values and the value yielding the best RMSE is selected, tuning continues in a rhythm.

The number of trees (*n.trees*), or the number of versions of the modeling trees, is selected after testing various values. Tuning is concluded with the *shrinkage* parameter, which signifies the learning rate. In order to speed the tuning process, the *bag.fraction* parameter maintains value at 0.8, which indicates the random sample size of the data used to train the model on each iteration.

The final values of the parameters are shown in the code below. The code below can take up to 173.92 seconds to run, or 2.8986667 minutes. Processing speed is increased using the `registerDoParallel` command.

```r
cl <- makeCluster(number_of_cores)
registerDoParallel(cl)

fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 5,
                           ## repeated ten times
                           repeats = 5)

gbmGrid <-  expand.grid(interaction.depth = 32,#seq(26,36,by=2),#3,
                        n.trees = 500,#seq(990,1040,by=5),
                        shrinkage = 0.1,#seq(0.01,0.1,by=0.01),
                        n.minobsinnode = 10)#seq(5,15,by=2))
set.seed(500)
system.time(gbmFit2 <- train(PH ~ ., data = train2,
                 method = "gbm",
                 trControl = fitControl,
                 bag.fraction = 0.8,
                 ## This last option is actually one
                 ## for gbm() that passes through
                 verbose = FALSE,
                 tuneGrid = gbmGrid)
)
```

```
##    user  system elapsed
##   10.11    0.01   59.01
```

```
gbmFit2
```

```
## Stochastic Gradient Boosting
##
## 1797 samples
##   36 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 1437, 1437, 1437, 1438, 1439, 1437, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1033126  0.6559972  0.07483122
##
## Tuning parameter 'n.trees' was held constant at a value of 500
## Tuning
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
```

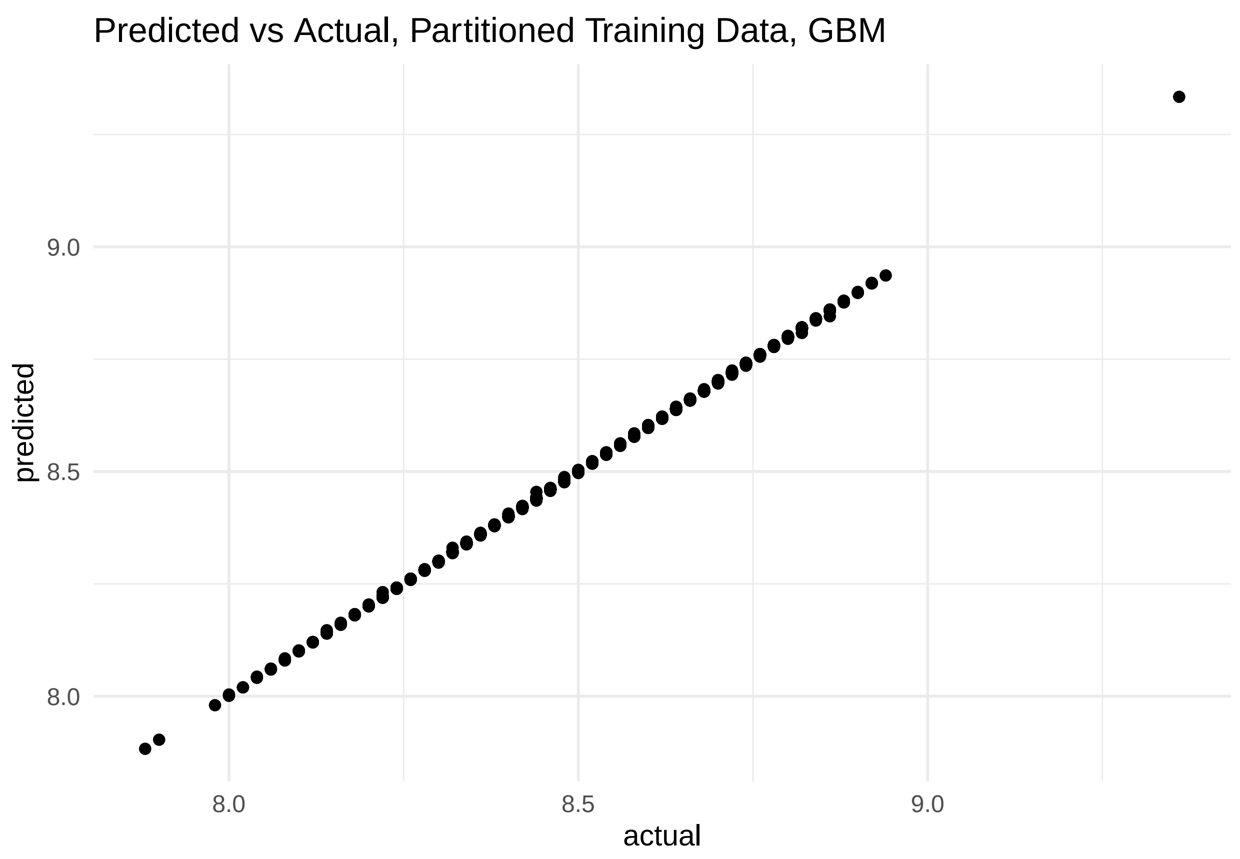
Interpretation: The values of RMSE and R-squared are optimized with the tuning process described above. The parameters listed in the output repesent the final values we will use to make our predictions. With the model tuned, predictive accuracy must be assessed in the partitioned test data.

The code chunk below plots predicted values vs actual values in the partitioned training data. Axes are limited to maximum and minimum values.

```
yhat_train2 <- predict(gbmFit2, newdata = train2[,-which(colnames(train2)=='PH')], type = 'raw')

plot_df_train2 <- as.data.frame(cbind(predicted = yhat_train2,actual = train2$PH))

ggplot(plot_df_train2, aes(actual, predicted)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Predicted vs Actual, Partitioned Training Data, GBM") +
  ylim(min(plot_df_train2$predicted),max(plot_df_train2$predicted)) +
  xlim(min(plot_df_train2$actual),max(plot_df_train2$actual))
```



Interpretation: The values align beautifully, but we will check the residuals in the testing data below for potential overfitting.

Model Assessment

Approach: Using the data provided, how accurate are our predictions? The code chunk below assesses various metrics of the predicted values the model yields. When combined with model tuning, parameters can be adjusted to optimize the result. Commands included in the `Metrics` package are used in the code chunk below. We compare predicted values with observed values in the partitioned test data with calculations of general mean difference, the proportion of exact matches after rounding, SMAPE, and RMSE.

```
# test 2
yhat_t2 <- predict(gbmFit2, newdata = test2, type = 'raw')
fit_t2 <- data.frame(cbind(yhat_t2,y2))
fit_t2$rnd_yhat <- round(fit_t2$yhat_t2,2)
fit_t2$error <- abs(fit_t2$yhat_t2-fit_t2$y2)
cat("mean error: ",mean(fit_t2$error),"\n")
```

```
## mean error:  0.01984321
```

```
cat("exact matches, accuracy: ",accuracy(y2,fit_t2$rnd_yhat),"\n")
```

```
## exact matches, accuracy:  0.7051948
```

```
cat("SMAPE: ",smape(y2,yhat_t2),"\n")
```

```
## SMAPE:  0.002344813
```

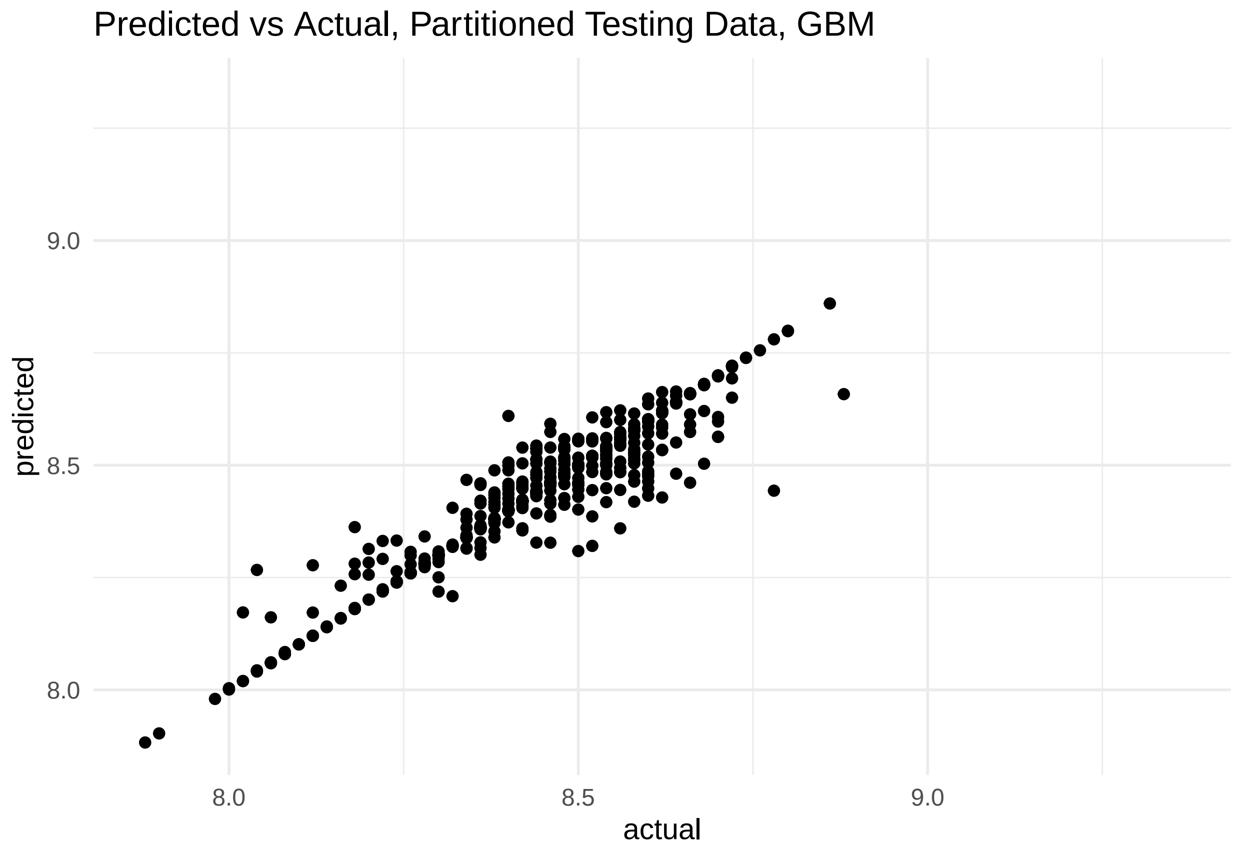
```
cat("RMSE: ",Metrics::rmse(y2,yhat_t2))
```

```
## RMSE:  0.04468366
```

Interpretation: The mean difference is quite low, with nearly 71% of *PH* values correctly predicted. RMSE is also quite low. Such a result is the product of careful model tuning. The code chunk below plots predicted values vs actual values. The x and y axes are limited to the maximum and minimum of the actual and predicted values of teh training data for appropriate comparison.

```
fit_test2 <- data.frame(cbind(predicted=yhat_t2,actual=y2))

ggplot(fit_test2, aes(actual, predicted)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Predicted vs Actual, Partitioned Testing Data, GBM") +
  ylim(min(plot_df_train2$predicted),max(plot_df_train2$predicted)) +
  xlim(min(plot_df_train2$actual),max(plot_df_train2$actual))
```



Interpretation: Predicted values are evenly distributed about the implied line of regression, indicating symmetry in the residuals and adequate model fit in the testing data. We can proceed with training a new model with the same parameters on the entire set of training data.

Training the Model: All Provided Imputed Data

Approach: With the model optimized we'll train our model with the optimal parameter values selected above on the entire dataset provided and prepare to make our final predictions.

First, we'll need to ensure that the columns are appropriately named and remove the *'_imp'* suffix from the imputed columns. This suffix is added by default to imputed variables by the `mice` function. The column names must match the names in the submission data in order for the model to run and make predictions.

```r
data_for_modeling3 <- data_for_modeling2
colnames(data_for_modeling3) <- str_replace(colnames(data_for_modeling2), '_imp', '')

fitControl <- trainControl(## 10-fold CV
                           method = "repeatedcv",
                           number = 5,
                           ## repeated ten times
                           repeats = 5)

gbmGrid <-  expand.grid(interaction.depth = 32,#seq(26,36,by=2),#3,
                        n.trees = 500,#seq(990,1040,by=5),
                        shrinkage = 0.1,#seq(0.01,0.1,by=0.01),
                        n.minobsinnode = 10)#seq(5,15,by=2))
set.seed(500)
system.time(gbmFit_Final <- train(PH ~ ., data = data_for_modeling3,
                 method = "gbm",
                 trControl = fitControl,
                 bag.fraction = 0.8,
                 ## This last option is actually one
                 ## for gbm() that passes through
                 verbose = FALSE,
                 tuneGrid = gbmGrid)
)
```

```
##    user  system elapsed
##   13.79    0.00   77.05
```

```r
gbmFit_Final
```

```
## Stochastic Gradient Boosting
##
## 2567 samples
##   36 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 5 times)
## Summary of sample sizes: 2055, 2052, 2054, 2053, 2054, 2054, ...
## Resampling results:
##
##   RMSE        Rsquared   MAE
##   0.09560299  0.6932342  0.06928541
##
## Tuning parameter 'n.trees' was held constant at a value of 500
## Tuning
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
```

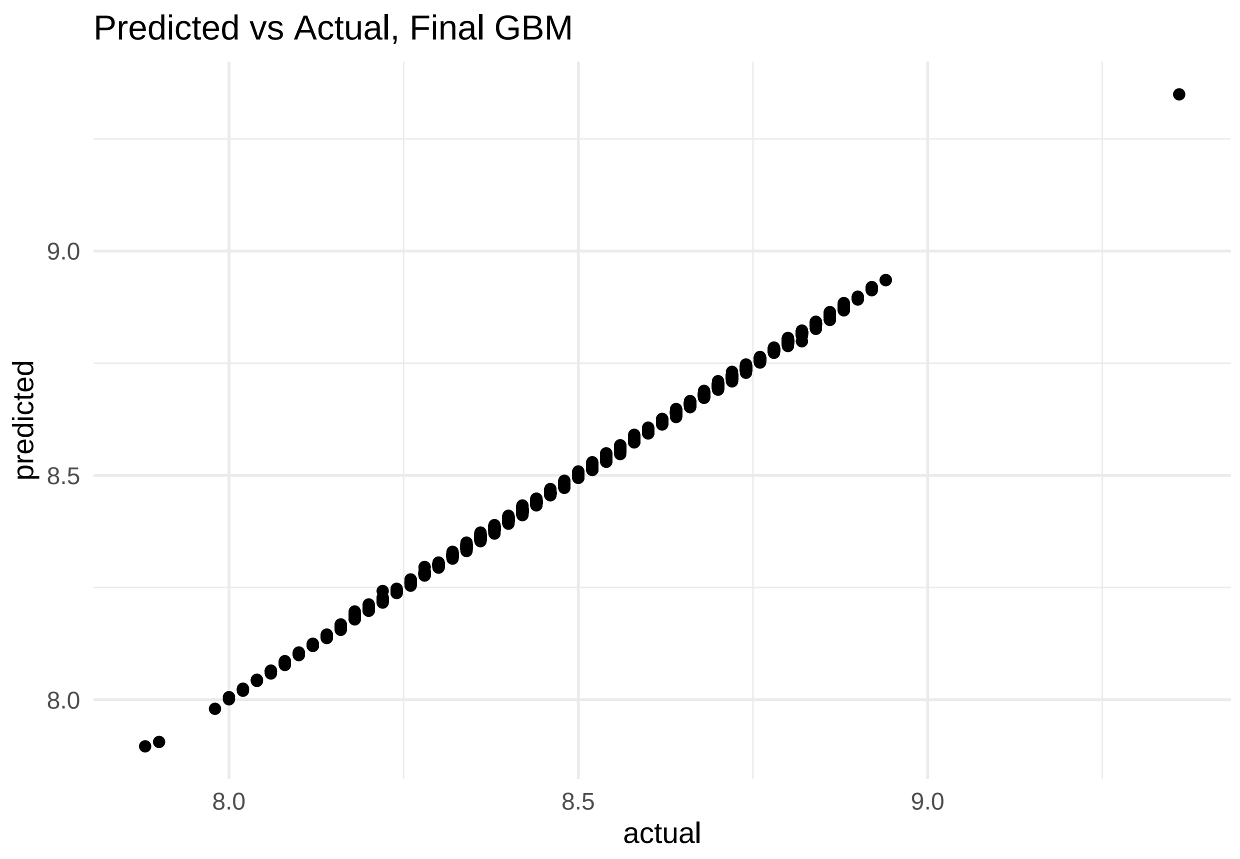
Interpretation: The RMSE is slightly less than the model run on the partitioned data. Though unanticipated, this result indicates a slightly better model fit. The R-squared value, also a slight improvement of around 0.05, is optimized and adequate for our purposes.

The following code chunk plots the predicted values versus the actual values in the final model.

```
yhat_dfm3 <- predict(gbmFit_Final, newdata = data_for_modeling3[,-which(colnames(data_for_modeling3)=='

plot_df <- as.data.frame(cbind(predicted = yhat_dfm3,actual = data_for_modeling3$PH))

ggplot(plot_df, aes(actual, predicted)) +
  geom_point() +
  theme_minimal() +
  ggtitle("Predicted vs Actual, Final GBM")
```



Interpretation: The model performs very well, though there could be a danger of overfitting. Residuals, however are uniformly distributed. We'll proceed with the predictions.

Predicting pH: The Final Steps

Approach: The Excel spreadsheet provided from which we must make our predictions was uploaded to GitHub in a public repositiory. The code chunk below loads the data and returns a dataframe.

Spaces within column names are removed and the variable *Brand Code* is manually dummy-coded. Imputation using `mice` follows as before, with the resulting imputed data added to the dataframe. Columns with missing data are dropped and the *'_imp'* suffix removed from the column titles.

Import Data: Data Provided for Prediction and Submission

Data are loaded:

```r
url <- paste0('https://github.com/AngelClaudio/data-sources/blob/master/',
              'excel/StudentEvaluation-%20TO%20PREDICT.xls?raw=true')

#X <- read.csv(url)
download.file(url, "temp_predict.xls", mode = "wb")

pred_data <- read_excel("temp_predict.xls")
```

Data Preparation

In the code chunk below, spaces are removed from column names and *Brand Code* is dummy-coded. Data are imputed using the method employed earlier in this document. The resulting imputed data are merged with the data provided for submission. Processing time is lengthy, but the result is worth the wait.

```r
tpred_data <- pred_data
colnames(tpred_data) <- str_replace(colnames(pred_data), '\\s', '')

tpred_data$BrandA <- 0
tpred_data$BrandA[which(tpred_data$BrandCode=='A')] <- 1
tpred_data$BrandB <- 0
tpred_data$BrandB[which(tpred_data$BrandCode=='B')] <- 1
tpred_data$BrandC <- 0
tpred_data$BrandC[which(tpred_data$BrandCode=='C')] <- 1
tpred_data$BrandD <- 0
tpred_data$BrandD[which(tpred_data$BrandCode=='D')] <- 1
tpred_data$BrandNA <- 0
tpred_data$BrandNA[which(is.na(tpred_data$BrandCode)==TRUE)] <- 1

# set aside BrandCode
BrandCode <- tpred_data$BrandCode
tpred_data$BrandCode <- NULL

tpred_noPH <- tpred_data[,-which(colnames(tpred_data)=='PH')]

# USING PARLMICE INSTEAD TO LEVERAGE PARALLEL PROCESSING
pred_imp <- parlmice(data = tpred_noPH, m = 5, method = "pmm",
                    n.core = number_of_cores, maxit = 50, seed = 500, print = FALSE)


NAnames <- names(tpred_noPH)[sapply(tpred_noPH, anyNA)]
tpred_noPH2 <- tpred_noPH[,-which(names(tpred_noPH) %in% NAnames)]

data_for_predicting <- merge_imputations(tpred_noPH,pred_imp,tpred_noPH2)
```

Predict

Finally, we make our predictions. Again, we'll remove the '_imp' suffix before running the model.

```r
data_for_predicting2 <- data_for_predicting
colnames(data_for_predicting2) <- str_replace(colnames(data_for_predicting), '_imp', '')
data_for_predicting3 <- data_for_predicting2[colnames(data_for_modeling3[,-which(colnames(data_for_model

# test 2
PH <- predict(gbmFit_Final, newdata = data_for_predicting3, type = 'raw')
data_for_predicting3$PH <- PH
```

```r
pred_data$PH <- PH
```

Construct Submission

The code chunk below creates an Excel workbook with two spreadsheets in the working directory, Predicted values are merged with the imputed data in a sheet entitled "PH_IMputedData". Predicted values are merged with the original data in a sheet entitled "PH_OriginalData". Due to the mixed variable types in the original dataframe the package `writexl` is used; these original data include the string variable *Brand Code* as provided. Predicted values without rounding are recorded to miminize error.

```r
library(writexl)

PH_OriginalData <- pred_data
PH_ImputedData <- data_for_predicting3

write_xlsx(x=list(PH_OriginalData = PH_OriginalData, PH_ImputedData = PH_ImputedData), path = "StudentE
```

Conclusion

The Linear Regression - OLS model only accounted for 40% of the variation in the data which makes Ordinary Linear Regression unfit for our prediction purposes.

The Decision Tree is used in regression problems if and only if the target variable is inside the range of values in the training dataset.

Decision Trees may be unfit for continuous variables such as the dependent variable (PH) in our dataset; a small variation in data may lead to a completely different tree being generated. Despite the limitations of Decision Tree, it helps to see the splits based on the variable importance.

However, we could not depend on the Decision Tree due to the nature of our dependent variable and the tendency to overfit. Hence, we used an improved tree-based Gradient Boost Model (GBM) which is more appropriate for our dependent variable and handle the overfitting tendency of the Decision Tree.

Training and tuning a Gradient Boost Model requires time to experiment, but the results are worth the effort. Due to the predictive power of the model, we were able to exactly match 71% of the pH values in during testing. Further, formulating a strategy to impute missing data necessitated a comprehensive overview of all elements involved with little knowledge of the variables involved or the guaranteed success of a specific approach. Trial and error was key.

Centering and scaling, while a good approach, could not be performed equally (and automatically) in two separate sets of data. Reproducibility in data cleaning and was also critical.

Missing data had to be imputed consistently to optimize model performance. Despite the complexities, the accuracy of the model yielded satisfactory results. May the RMSE forever be in our favor.

Reference

Singh, H. (2018). Understanding Gradient Boosting Machines. Retrieved from https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab