

Data624 - Project2

Amanda Arce, Jatin Jain, Amit Kapoor

5/2/2021

Contents

Overview	1
R packages	2
Data Exploration	2
Data summary	2
Variables Distribution	4
Missing Data	6
Correlation	6
Outliers	7
Data Preparation	8
Handling missing and outliers	8
Create Dummy Variables	8
Correlation	8
Preprocess using transformation	9
Training and Test Partition	9
Build Models	9
Linear Regression	9
Non Linear Regression	11
Trees	14
Select Model	19
Prediction	21
Conclusion	22
References	22
Code Appendix	22

Overview

ABC Beverage has new regulations in place and the leadership team requires the data scientists team to understand the manufacturing process, the predictive factors and be able to report to them predictive model of PH. The selection of model depends upon various factors like model accuracy, data relevance, cross validation etc.

R packages

We will use `r` for data modeling. All packages used for data exploration, visualization, preparation and modeling are listed in Code Appendix.

Data Exploration

We will first get the historical dataset, provided in excel and use it to analyze and eventually predict the PH of beverages.

Data summary

There are 31 predictor variables that are numeric and 1 predictor variable `Brand Code` which is factor. The training dataset has 2,571 observations.

```
## Rows: 2,571
## Columns: 33
## $ `Brand Code`      <fct> B, A, B, A, A, A, A, B, B, B, B, B, B, B, B, B, C, ~
## $ `Carb Volume`     <dbl> 5.340000, 5.426667, 5.286667, 5.440000, 5.486667, ~
## $ `Fill Ounces`     <dbl> 23.96667, 24.00667, 24.06000, 24.00667, 24.31333, ~
## $ `PC Volume`       <dbl> 0.2633333, 0.2386667, 0.2633333, 0.2933333, 0.1113~
## $ `Carb Pressure`   <dbl> 68.2, 68.4, 70.8, 63.0, 67.2, 66.6, 64.2, 67.6, 64~
## $ `Carb Temp`       <dbl> 141.2, 139.6, 144.8, 132.6, 136.8, 138.4, 136.8, 1~
## $ PSC               <dbl> 0.104, 0.124, 0.090, NA, 0.026, 0.090, 0.128, 0.15~
## $ `PSC Fill`        <dbl> 0.26, 0.22, 0.34, 0.42, 0.16, 0.24, 0.40, 0.34, 0.~
## $ `PSC CO2`         <dbl> 0.04, 0.04, 0.16, 0.04, 0.12, 0.04, 0.04, 0.04, 0.~
## $ `Mnf Flow`        <dbl> -100, -100, -100, -100, -100, -100, -100, -100, -1~
## $ `Carb Pressure1`  <dbl> 118.8, 121.6, 120.2, 115.2, 118.4, 119.6, 122.2, 1~
## $ `Fill Pressure`   <dbl> 46.0, 46.0, 46.0, 46.4, 45.8, 45.6, 51.8, 46.8, 46~
## $ `Hyd Pressure1`   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ `Hyd Pressure2`   <dbl> NA, NA, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ `Hyd Pressure3`   <dbl> NA, NA, NA, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ `Hyd Pressure4`   <dbl> 118, 106, 82, 92, 92, 116, 124, 132, 90, 108, 94, ~
## $ `Filler Level`    <dbl> 121.2, 118.6, 120.0, 117.8, 118.6, 120.2, 123.4, 1~
## $ `Filler Speed`    <dbl> 4002, 3986, 4020, 4012, 4010, 4014, NA, 1004, 4014~
## $ Temperature       <dbl> 66.0, 67.6, 67.0, 65.6, 65.6, 66.2, 65.8, 65.2, 65~
## $ `Usage cont`      <dbl> 16.18, 19.90, 17.76, 17.42, 17.68, 23.82, 20.74, 1~
## $ `Carb Flow`       <dbl> 2932, 3144, 2914, 3062, 3054, 2948, 30, 684, 2902,~
## $ Density           <dbl> 0.88, 0.92, 1.58, 1.54, 1.54, 1.52, 0.84, 0.84, 0.~
## $ MFR               <dbl> 725.0, 726.8, 735.0, 730.6, 722.8, 738.8, NA, NA, ~
## $ Balling           <dbl> 1.398, 1.498, 3.142, 3.042, 3.042, 2.992, 1.298, 1~
## $ `Pressure Vacuum` <dbl> -4.0, -4.0, -3.8, -4.4, -4.4, -4.4, -4.4, -4.4, -4~
## $ PH               <dbl> 8.36, 8.26, 8.94, 8.24, 8.26, 8.32, 8.40, 8.38, 8.~
## $ `Oxygen Filler`   <dbl> 0.022, 0.026, 0.024, 0.030, 0.030, 0.024, 0.066, 0~
## $ `Bowl Setpoint`   <dbl> 120, 120, 120, 120, 120, 120, 120, 120, 120, ~
## $ `Pressure Setpoint` <dbl> 46.4, 46.8, 46.6, 46.0, 46.0, 46.0, 46.0, 46.0, 46~
## $ `Air Pressurer`   <dbl> 142.6, 143.0, 142.0, 146.2, 146.2, 146.6, 146.2, 1~
## $ `Alch Rel`        <dbl> 6.58, 6.56, 7.66, 7.14, 7.14, 7.16, 6.54, 6.52, 6.~
## $ `Carb Rel`        <dbl> 5.32, 5.30, 5.84, 5.42, 5.44, 5.44, 5.38, 5.34, 5.~
## $ `Balling Lvl`     <dbl> 1.48, 1.56, 3.28, 3.04, 3.04, 3.02, 1.44, 1.44, 1.~

##           n    mean    sd  median    min    max   range  skew
## Brand Code* 2451   2.51   1.00    2.00    1.00    4.00    3.00  0.38
## Carb Volume 2561   5.37   0.11    5.35    5.04    5.70    0.66  0.39
## Fill Ounces 2533  23.97   0.09   23.97   23.63   24.32    0.69 -0.02
```

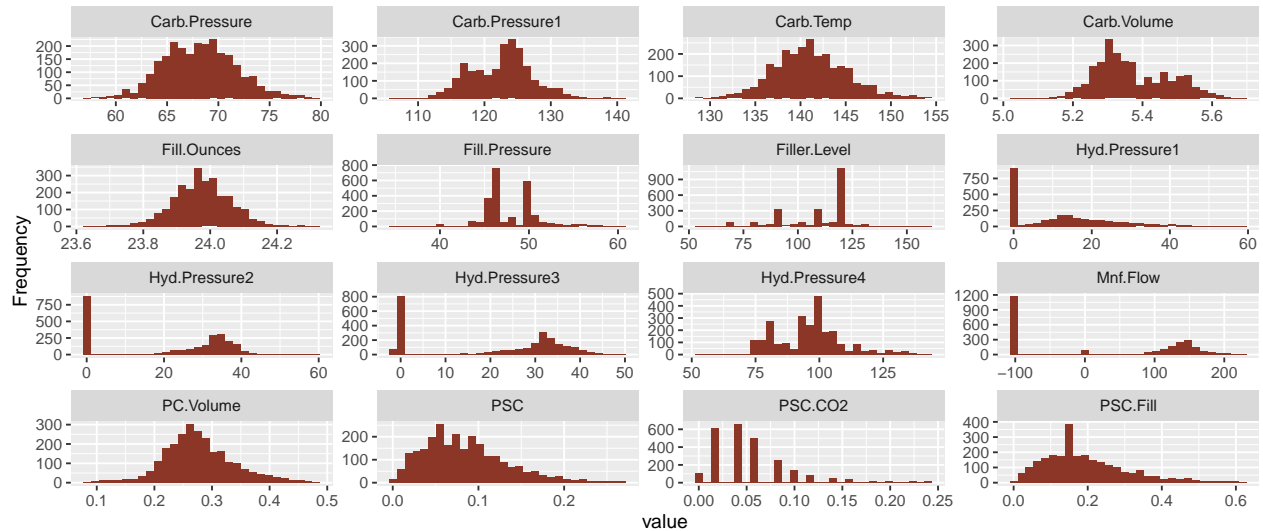
## PC Volume	2532	0.28	0.06	0.27	0.08	0.48	0.40	0.34
## Carb Pressure	2544	68.19	3.54	68.20	57.00	79.40	22.40	0.18
## Carb Temp	2545	141.09	4.04	140.80	128.60	154.00	25.40	0.25
## PSC	2538	0.08	0.05	0.08	0.00	0.27	0.27	0.85
## PSC Fill	2548	0.20	0.12	0.18	0.00	0.62	0.62	0.93
## PSC C02	2532	0.06	0.04	0.04	0.00	0.24	0.24	1.73
## Mnf Flow	2569	24.57	119.48	65.20	-100.20	229.40	329.60	0.00
## Carb Pressure1	2539	122.59	4.74	123.20	105.60	140.20	34.60	0.05
## Fill Pressure	2549	47.92	3.18	46.40	34.60	60.40	25.80	0.55
## Hyd Pressure1	2560	12.44	12.43	11.40	-0.80	58.00	58.80	0.78
## Hyd Pressure2	2556	20.96	16.39	28.60	0.00	59.40	59.40	-0.30
## Hyd Pressure3	2556	20.46	15.98	27.60	-1.20	50.00	51.20	-0.32
## Hyd Pressure4	2541	96.29	13.12	96.00	52.00	142.00	90.00	0.55
## Filler Level	2551	109.25	15.70	118.40	55.80	161.20	105.40	-0.85
## Filler Speed	2514	3687.20	770.82	3982.00	998.00	4030.00	3032.00	-2.87
## Temperature	2557	65.97	1.38	65.60	63.60	76.20	12.60	2.39
## Usage cont	2566	20.99	2.98	21.79	12.08	25.90	13.82	-0.54
## Carb Flow	2569	2468.35	1073.70	3028.00	26.00	5104.00	5078.00	-0.99
## Density	2570	1.17	0.38	0.98	0.24	1.92	1.68	0.53
## MFR	2359	704.05	73.90	724.00	31.40	868.60	837.20	-5.09
## Balling	2570	2.20	0.93	1.65	-0.17	4.01	4.18	0.59
## Pressure Vacuum	2571	-5.22	0.57	-5.40	-6.60	-3.60	3.00	0.53
## PH	2567	8.55	0.17	8.54	7.88	9.36	1.48	-0.29
## Oxygen Filler	2559	0.05	0.05	0.03	0.00	0.40	0.40	2.66
## Bowl Setpoint	2569	109.33	15.30	120.00	70.00	140.00	70.00	-0.97
## Pressure Setpoint	2559	47.62	2.04	46.00	44.00	52.00	8.00	0.20
## Air Pressurer	2571	142.83	1.21	142.60	140.80	148.20	7.40	2.25
## Alch Rel	2562	6.90	0.51	6.56	5.28	8.62	3.34	0.88
## Carb Rel	2561	5.44	0.13	5.40	4.96	6.06	1.10	0.50
## Balling Lvl	2570	2.05	0.87	1.48	0.00	3.66	3.66	0.59
##	kurtosis							
## Brand Code*	-1.06							
## Carb Volume	-0.47							
## Fill Ounces	0.86							
## PC Volume	0.67							
## Carb Pressure	-0.01							
## Carb Temp	0.24							
## PSC	0.65							
## PSC Fill	0.77							
## PSC C02	3.73							
## Mnf Flow	-1.87							
## Carb Pressure1	0.14							
## Fill Pressure	1.41							
## Hyd Pressure1	-0.14							
## Hyd Pressure2	-1.56							
## Hyd Pressure3	-1.57							
## Hyd Pressure4	0.63							
## Filler Level	0.05							
## Filler Speed	6.71							
## Temperature	10.16							
## Usage cont	-1.02							
## Carb Flow	-0.58							
## Density	-1.20							
## MFR	30.46							

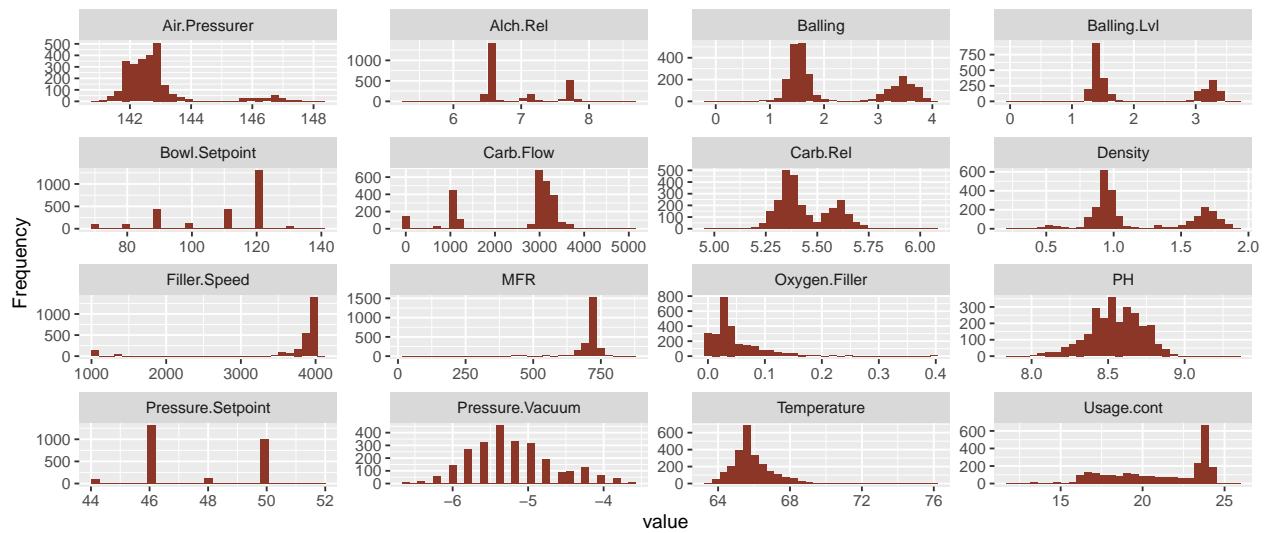
```
## Balling -1.39
## Pressure Vacuum -0.03
## PH 0.06
## Oxygen Filler 11.09
## Bowl Setpoint -0.06
## Pressure Setpoint -1.60
## Air Pressurer 4.73
## Alch Rel -0.85
## Carb Rel -0.29
## Balling Lvl -1.49
```

Based on above description, we can see the dataset has missing values so it would need imputation. The predictors **Oxygen Filler**, **MFR**, **Filler Speed** and **Temperature** seems highly skewed and would require transformation. This could be seen in below histogram plots as well.

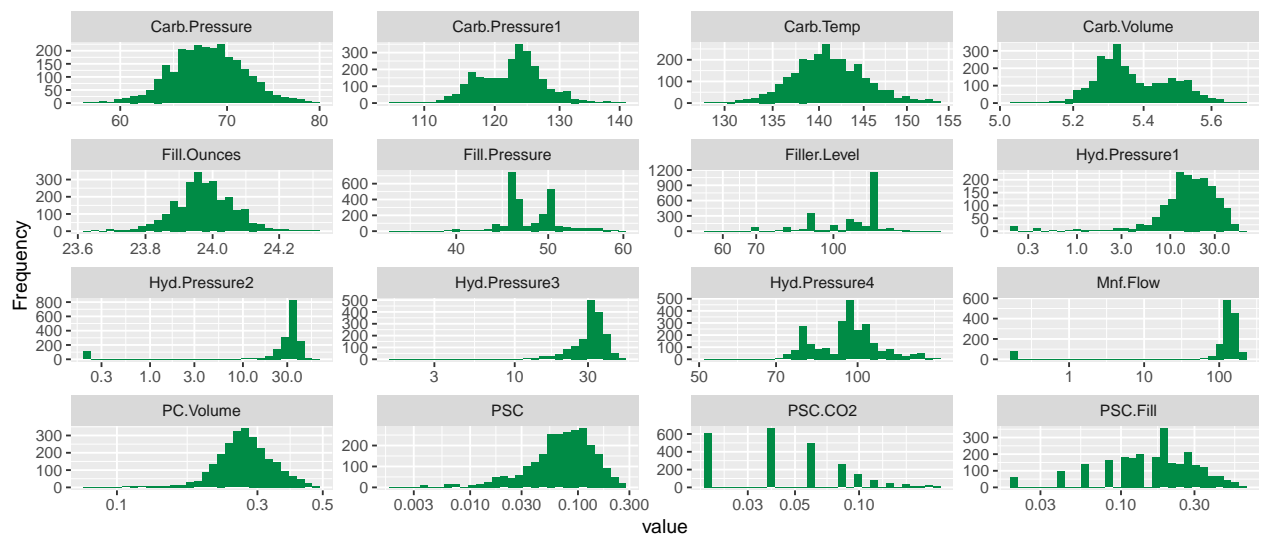
Variables Distribution

Below we have shown the distribution of dataset variables. There are 2 sets of histograms; the one in red is natural distribution and the ones in green are logarithmic distribution

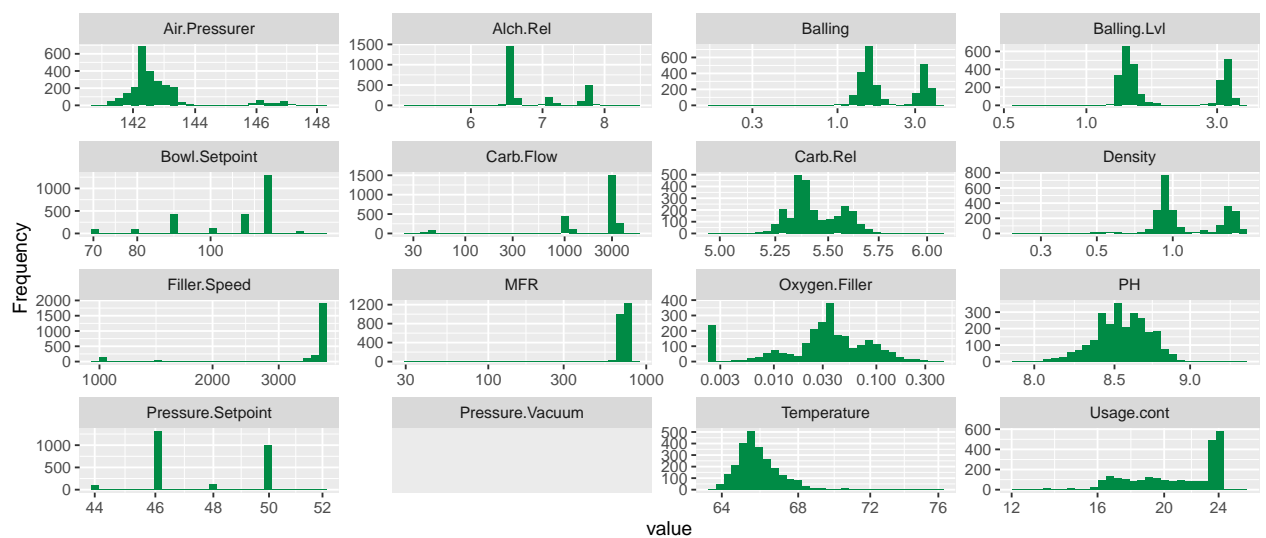




Page 2



Page 1

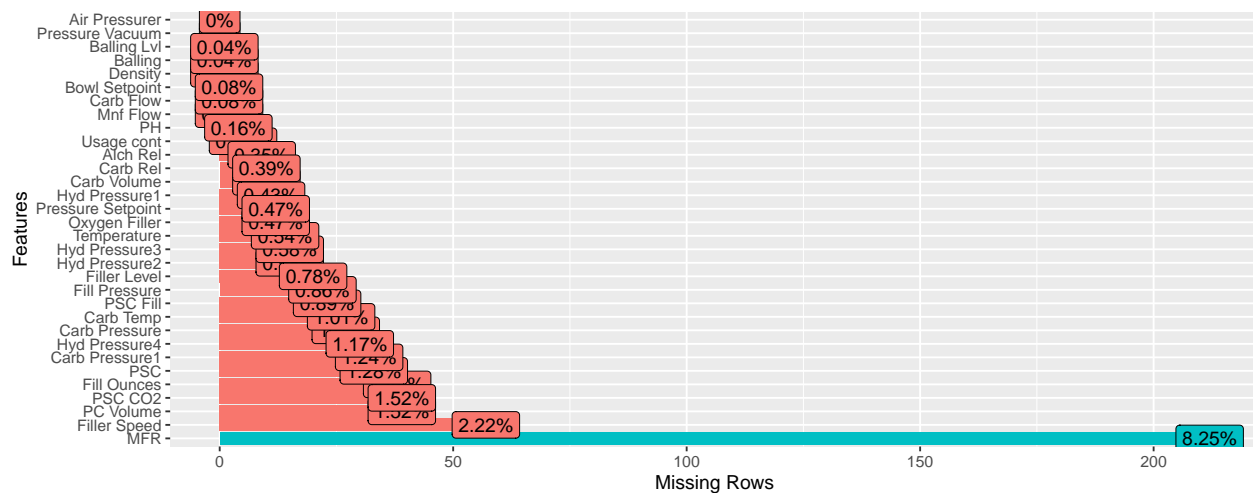


Page 2

Missing Data

The summary and following graphs show the missing data in training dataset. The plot below shows more than 8% data is missing for MFR variable. Next feature that has missing data is Filler Speed which shows more than 2% missing data. The missing data will be handled through imputation.

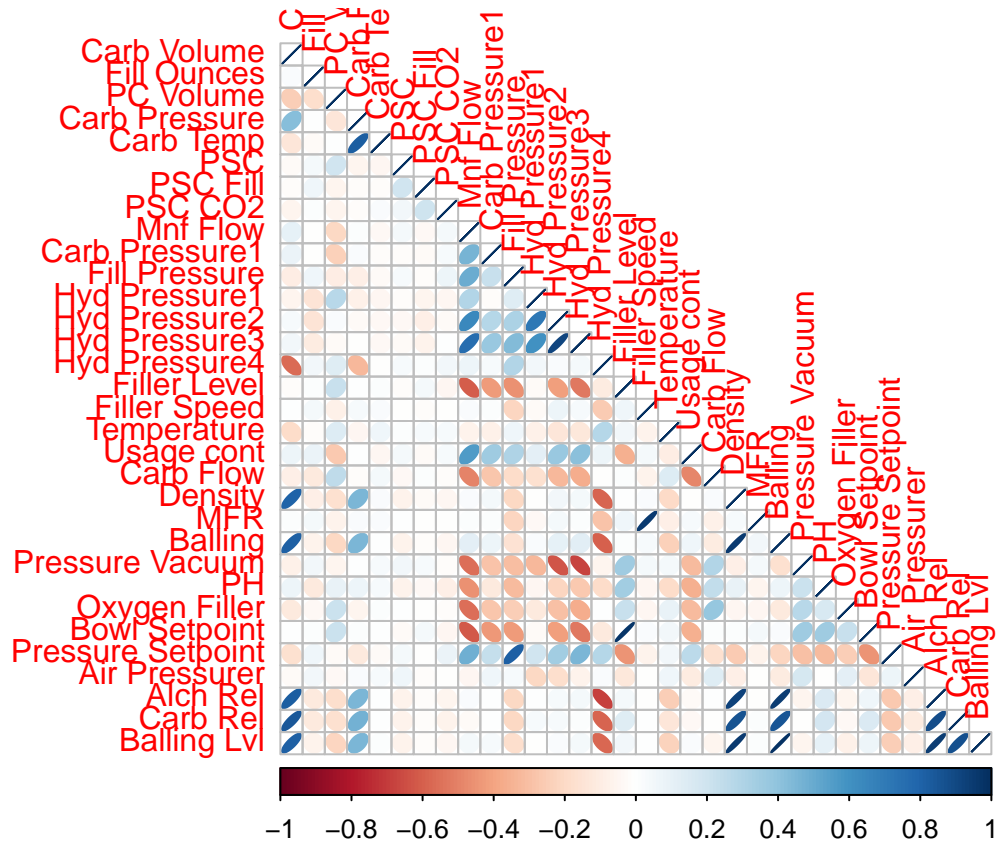
##	Brand Code	Carb Volume	Fill Ounces	PC Volume
##	120	10	38	39
##	Carb Pressure	Carb Temp	PSC	PSC Fill
##	27	26	33	23
##	PSC CO2	Mnf Flow	Carb Pressure1	Fill Pressure
##	39	2	32	22
##	Hyd Pressure1	Hyd Pressure2	Hyd Pressure3	Hyd Pressure4
##	11	15	15	30
##	Filler Level	Filler Speed	Temperature	Usage cont
##	20	57	14	5
##	Carb Flow	Density	MFR	Balling
##	2	1	212	1
##	Pressure Vacuum	PH	Oxygen Filler	Bowl Setpoint
##	0	4	12	2
##	Pressure Setpoint	Air Pressurer	Alch Rel	Carb Rel
##	12	0	9	10
##	Balling Lvl			
##	1			



Band ■ Good ■ OK

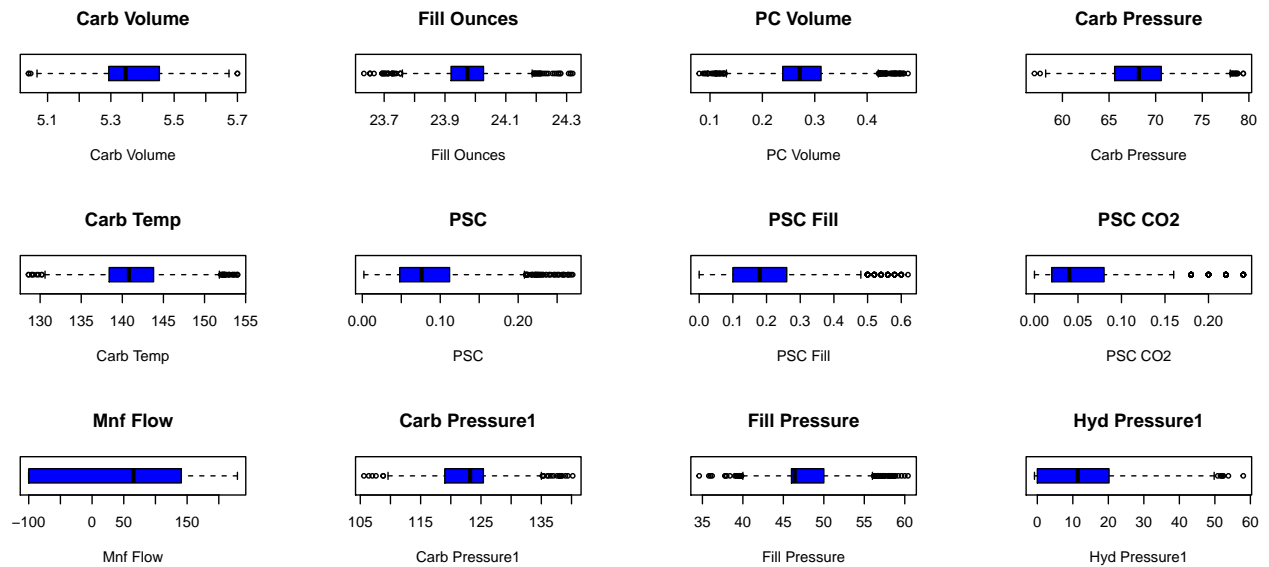
Correlation

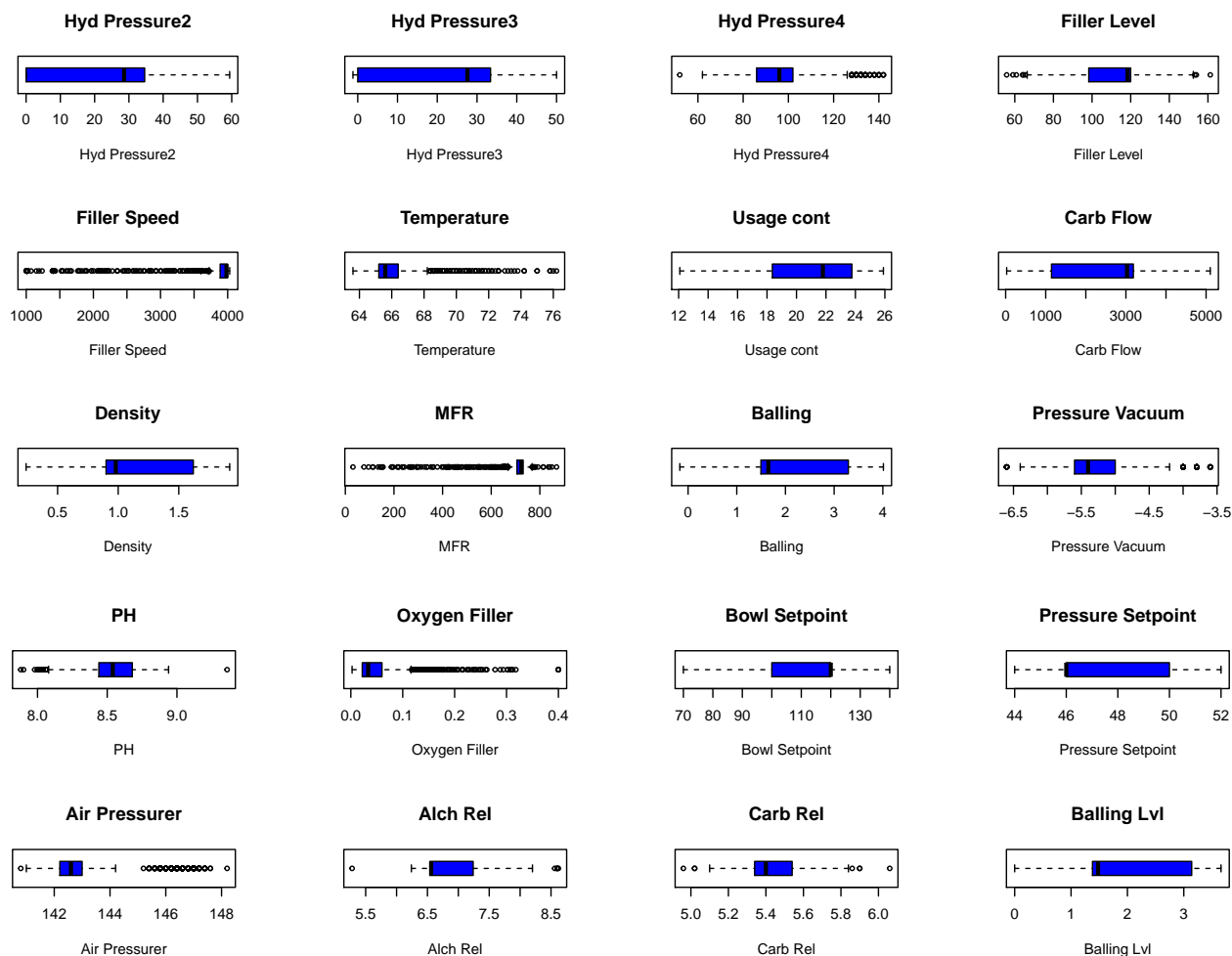
below plot shows the correlation among numeric variables in the dataset. We can see few variables are highly correlated. We will handle the pairwise predictors that has correlation above 0.90 in data preparation section.



Outliers

In this section we will check the outliers in the data. An outlier is an observation that lies an unusual distance from other values in a random sample. These outlier could impact predictions so will be handled through imputation





Data Preparation

Handling missing and outliers

The very first in data preparation we will perform is handling missing data and outliers through imputation. We will use mice package to perform imputation here. MICE (Multivariate Imputation via Chained Equations) is one of the commonly used package for this activity. It creates multiple imputations for multivariate missing data. Also we will perform `nearZeroVar` to see if a variable has very little change or variation and not useful for prediction. If we found any predictor variable satisfying this condition we would remove it.

Create Dummy Variables

The variable Brand Code is a categorical variable, having 4 classes (A, B, C, and D). For modeling, we got to convert into set of dummy variables. We will use `dummyVars` function for this purpose that creates a full set of dummy variables.

Correlation

Next step is to remove highly correlated predictor variables. we will use the cutoff as 0.90 here.

Preprocess using transformation

In this step we will use caret `preprocess` method using transformation as `YeoJohnson` which applies Yeo-Johnson transformation, like a BoxCox, but values can be negative as well.

Training and Test Partition

Finally in this step for data preparation we will partition the training dataset for training and validation using `createDataPartition` method from `caret` package. We will reserve 75% for training and rest 25% for validation purpose.

Build Models

Linear Regression

Simple Linear Regression

We will start with Simple Linear Regression model. It will include all the predictor variables in training dataset.

```
##
## Call:
## lm(formula = y.train ~ ., data = X.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.50865 -0.08187  0.01030  0.08869  0.82191
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -7.448e+02  4.113e+03  -0.181  0.85633
## Brand.Code.A   -5.371e-02  1.773e-02  -3.030  0.00248 **
## Brand.Code.B    4.502e-02  2.794e-02   1.611  0.10729
## Brand.Code.C   -8.575e-02  2.891e-02  -2.966  0.00305 **
## Brand.Code.D           NA           NA           NA           NA
## Carb.Volume    -5.906e-02  9.057e-02  -0.652  0.51443
## Fill.Ounces    -2.363e-03  9.073e-04  -2.605  0.00927 **
## PC.Volume      -2.143e-01  9.730e-02  -2.202  0.02777 *
## Carb.Pressure   2.425e-01  6.661e-01   0.364  0.71593
## Carb.Temp       1.481e+03  8.134e+03   0.182  0.85554
## PSC             -4.662e-02  6.751e-02  -0.691  0.48992
## PSC.Fill        -7.216e-02  5.804e-02  -1.243  0.21393
## PSC.CO2         -1.478e-01  7.517e-02  -1.966  0.04948 *
## Mnf.Flow        -6.253e-04  5.385e-05 -11.612 < 2e-16 ***
## Carb.Pressure1   8.485e-02  1.007e-02   8.426 < 2e-16 ***
## Fill.Pressure    2.690e+00  8.374e-01   3.212  0.00134 **
## Hyd.Pressure2    7.293e-03  1.229e-03   5.932 3.54e-09 ***
## Hyd.Pressure4    5.719e-02  1.123e-01   0.509  0.61070
## Temperature     -1.270e-02  2.553e-03  -4.976 7.06e-07 ***
## Usage.cont       -8.134e-03  1.341e-03  -6.068 1.56e-09 ***
## Carb.Flow        9.961e-07  4.356e-07   2.287  0.02232 *
## MFR             -5.603e-05  3.119e-05  -1.797  0.07255 .
## Pressure.Vacuum  9.796e-05  1.700e-04   0.576  0.56451
## Oxygen.Filler    -2.461e-01  7.787e-02  -3.160  0.00160 **
## Bowl.Setpoint    2.408e-03  3.106e-04   7.752 1.46e-14 ***
```

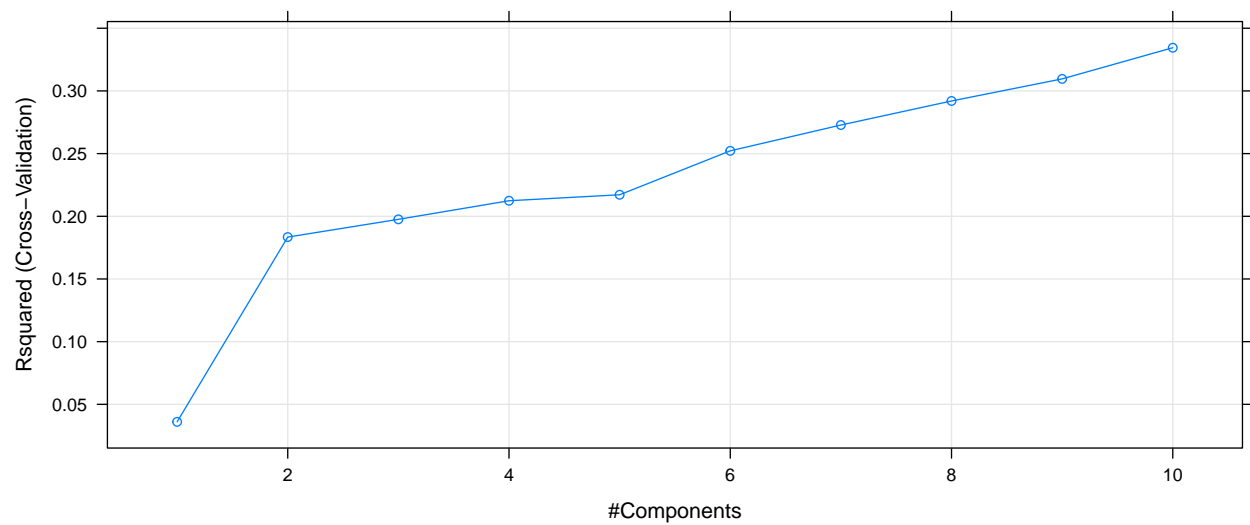
```
## Pressure.Setpoint -7.424e-03 2.266e-03 -3.276 0.00107 **
## Air.Pressurer      4.131e-04 2.784e-03 0.148 0.88205
## Alch.Rel           6.044e-02 2.220e-02 2.722 0.00654 **
## Carb.Rel           4.822e-03 5.215e-02 0.092 0.92634
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1359 on 1902 degrees of freedom
## Multiple R-squared:  0.3875, Adjusted R-squared:  0.3788
## F-statistic: 44.56 on 27 and 1902 DF,  p-value: < 2.2e-16
```

We can see that Simple Linear Regression model only covers 38% of variability of data. Next we will check for better models which covers better variability, RMSE and MAE. We can consider this Simple regression as a benchmark model among others we are going to check.

Partial Least Squares

Partial least squares (PLS) is an alternative to ordinary least squares (OLS) regression. It reduces the predictors to a smaller set of uncorrelated components and then performs least squares regression on these components, instead of on the original data. PLS finds linear combinations of the predictors called components. PLS finds components that attempts to maximally summarize the variation of the predictors while at the same time attempts these components to have maximum correlation with the response.

```
## Partial Least Squares
##
## 1930 samples
## 28 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1737, 1737, 1737, 1737, 1737, 1737, ...
## Resampling results across tuning parameters:
##
##  ncomp  RMSE      Rsquared    MAE
##  1      0.1695850  0.03602419  0.1357999
##  2      0.1557700  0.18341292  0.1233261
##  3      0.1545817  0.19755922  0.1216561
##  4      0.1532406  0.21239112  0.1210109
##  5      0.1528414  0.21716670  0.1205175
##  6      0.1491851  0.25219747  0.1167355
##  7      0.1471796  0.27279224  0.1153866
##  8      0.1450812  0.29197059  0.1140281
##  9      0.1431131  0.30960456  0.1118836
## 10      0.1405086  0.33444735  0.1097224
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 10.
##
##  ncomp
## 10    10
```



```
##      ncomp      RMSE Rsquared
## 1      10 0.1405086 0.3344473

##      Rsquared      RMSE
## 1 0.3344473 0.1405086
```

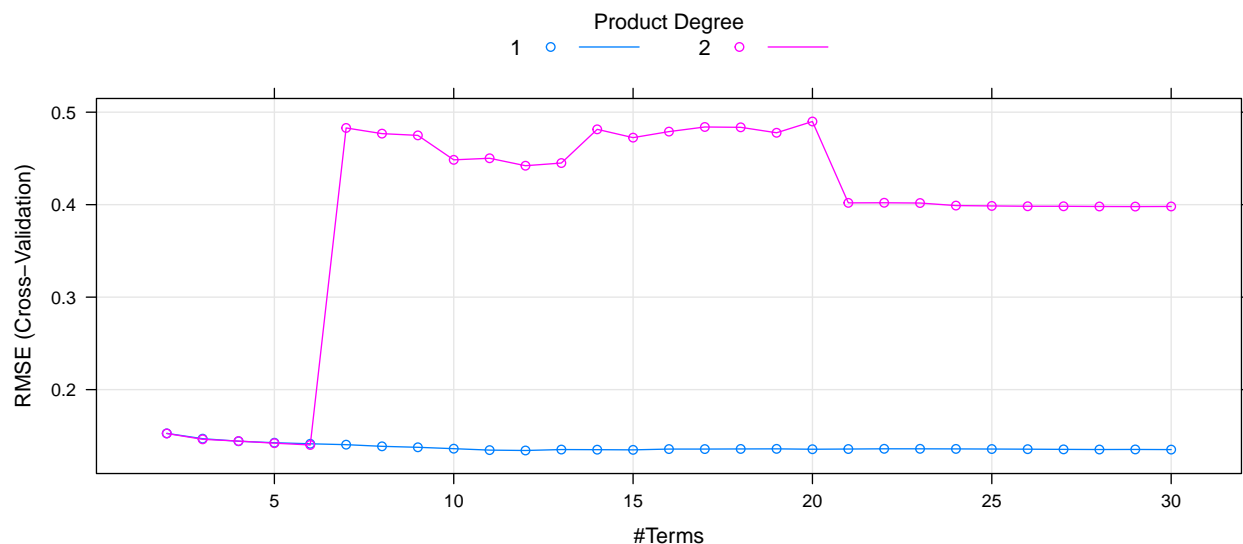
Rquared was used to select the optimal model using the largest value. The final value used for the model was $ncomp = 10$ which corresponds to best tune model. In this case we see that R^2 is 0.33 so only covers 33% variability in data but it produces small RMSE.

Non Linear Regression

MARS

MARS creates a piecewise linear model which provides an intuitive stepping block into non-linearity after grasping the concept of multiple linear regression. MARS provided a convenient approach to capture the nonlinear relationships in the data by assessing cutpoints (knots) similar to step functions. The procedure assesses each data point for each predictor as a knot and creates a linear regression model with the candidate features

```
##      nprune degree
## 11      12      1
```



```
## Call: earth(x=data.frame[1930,28], y=c(8.36,8.26,8.9...), keepxy=TRUE,
##           degree=1, nprune=12)
##
##               coefficients
## (Intercept)      8.5433911
## Brand.Code.C    -0.1187748
## h(0.199353-Mnf.Flow) 0.0011789
## h(19.8091-Carb.Pressure1) -0.0845067
## h(2.1938-Hyd.Pressure2) -0.0196752
## h(Temperature-65) -0.0307466
## h(Temperature-68.4) 0.0443562
## h(Usage.cont-22.2) -0.0443013
## h(Pressure.Vacuum- -63.7021) -0.0031964
## h(Bowl.Setpoint-90) 0.0024596
## h(7.16-Alch.Rel) 0.1127772
## h(Alch.Rel-7.16) 0.1804968
##
## Selected 12 of 32 terms, and 9 of 28 predictors (nprune=12)
## Termination condition: RSq changed by less than 0.001 at 32 terms
## Importance: Mnf.Flow, Brand.Code.C, Alch.Rel, Usage.cont, Carb.Pressure1, ...
## Number of terms at each degree of interaction: 1 11 (additive model)
## GCV 0.0178492   RSS 33.63279   GRSq 0.3998146   RSq 0.4134266
##
## Rsquared      RMSE
## 1 0.337374 0.1404835
```

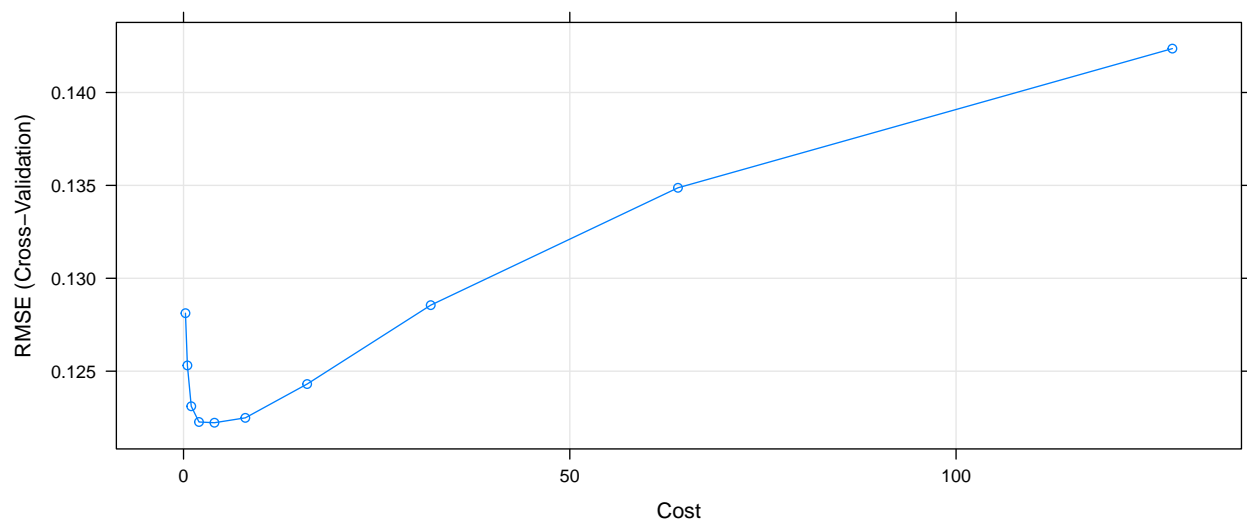
RMSE was used to select the optimal model using the smallest value. The final values used for the model were $nprune = 12$ and $degree = 1$ which corresponds to best tune model. In this case we see that R^2 is 0.33 so only covers 33% variability in data but it also produces small RMSE.

Support Vector Machines

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N being the number of features) that classifies the data points. Hyperplanes are decision boundaries to classify the data points. Data points that falls on either side of the hyperplane can be qualified for different classes. Support vectors are data points that are closer to the hyperplane and effect the position and orientation of the hyperplane. Using these support vectors, we do maximize the margin of the classifier.

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1930 samples
## 28 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1737, 1737, 1737, 1737, 1737, 1737, ...
## Resampling results across tuning parameters:
##
## C      RMSE      Rsquared  MAE
## 0.25  0.1281215  0.4579577  0.09713141
## 0.50  0.1253089  0.4791453  0.09426183
## 1.00  0.1231103  0.4955504  0.09220593
## 2.00  0.1222647  0.5021667  0.09130259
## 4.00  0.1222248  0.5026551  0.09101076
## 8.00  0.1224871  0.5032033  0.09123477
## 16.00 0.1243066  0.4963536  0.09255112
## 32.00 0.1285533  0.4780063  0.09605588
## 64.00 0.1348701  0.4507836  0.10107242
## 128.00 0.1423623  0.4203709  0.10632568
##
## Tuning parameter 'sigma' was held constant at a value of 0.02339058
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.02339058 and C = 4.

## Length Class Mode
## 1 ksvm S4
```



```
## Rsquared RMSE
## 1 0.5026551 0.1222248
```

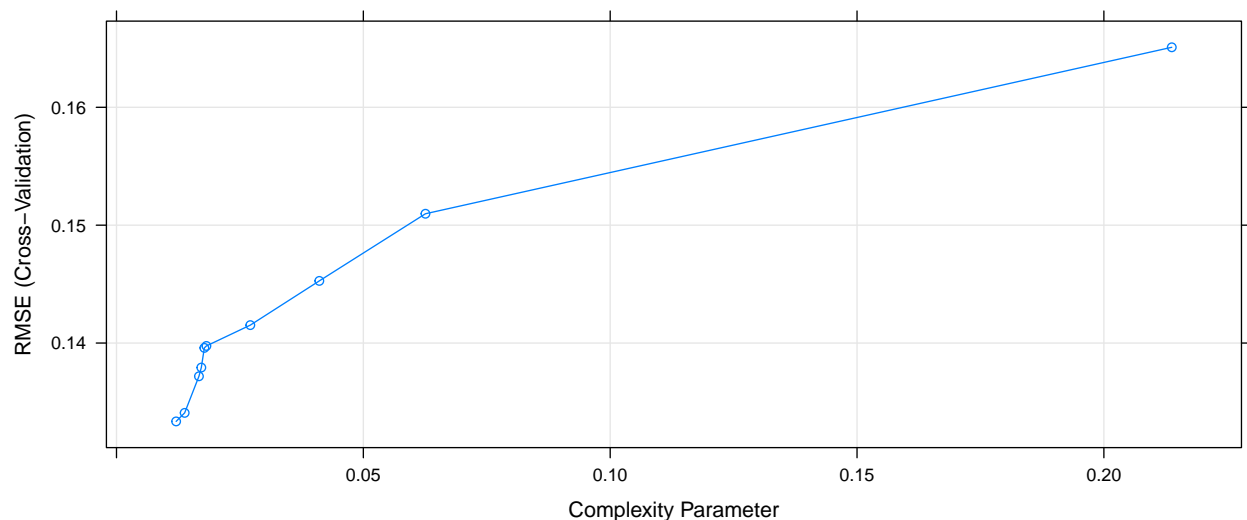
RMSE was used to select the optimal model using the smallest value. The final values used for the model were $\sigma = 0.02339058$ and $C = 4$. We see an improvement here in R^2 value which is 0.50 so this model covers 50% variability in the data and RMSE is smallest as well among the models used so far.

Trees

Single Tree

Regression trees partition a data set into smaller groups and then fit a simple model for each subgroup. Basic regression trees partition the data into smaller groups that are more homogenous against the response. To achieve outcome consistency, regression trees determine the predictor to split on and value of the split, the depth or complexity of the tree and the prediction equation in the terminal nodes

```
## CART
##
## 1930 samples
## 28 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1737, 1737, 1737, 1737, 1737, ...
## Resampling results across tuning parameters:
##
##   cp          RMSE      Rsquared    MAE
## 0.01207650 0.1333503 0.4028576 0.1044068
## 0.01380773 0.1340749 0.3955543 0.1053418
## 0.01669720 0.1371790 0.3668106 0.1080092
## 0.01718055 0.1379191 0.3599851 0.1087860
## 0.01779192 0.1395878 0.3461880 0.1103633
## 0.01820999 0.1397537 0.3446394 0.1103571
## 0.02710422 0.1415195 0.3281409 0.1121354
## 0.04108223 0.1452717 0.2904693 0.1150555
## 0.06257960 0.1509679 0.2330729 0.1184916
## 0.21375175 0.1650893 0.1676081 0.1313650
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.0120765.
##
##      cp
## 1 0.0120765
```



```
##      Rsquared      RMSE
## 1 0.4028576 0.1333503
```

RMSE was used to select the optimal model using the smallest value. The final value used for the model was $cp = 0.0120765$. We see R^2 value is 0.40 so this model covers 40% variability in the data and RMSE as 0.133. The Rsquared value is comparatively low as compared to previous best value.

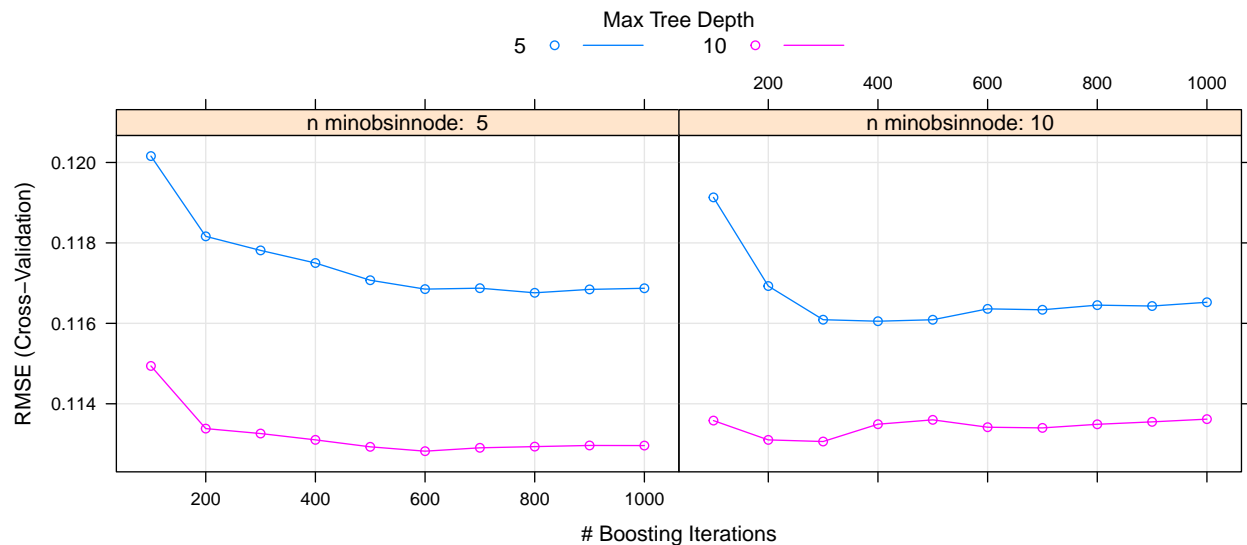
Boosted Tree

Boosting algorithms are influenced by learning theory. Boosting algorithm seeks to improve the prediction power by training a sequence of weak models where each of them compensates the weaknesses of its predecessors. The trees in boosting are dependent on past trees, have minimum depth and do not contribute equally to the final model. It requires us to specify a weak model (e.g. regression, shallow decision trees etc) and then improves it.

```
## Stochastic Gradient Boosting
##
## 1930 samples
## 28 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1737, 1737, 1737, 1737, 1737, 1737, ...
## Resampling results across tuning parameters:
##
## interaction.depth n.minobsinnode n.trees RMSE Rsquared MAE
## 5 5 100 0.1201589 0.5146306 0.09207892
## 5 5 200 0.1181629 0.5292995 0.08948714
## 5 5 300 0.1178137 0.5332979 0.08884903
## 5 5 400 0.1175003 0.5368565 0.08843330
## 5 5 500 0.1170720 0.5410113 0.08774659
## 5 5 600 0.1168508 0.5432174 0.08755074
## 5 5 700 0.1168744 0.5432030 0.08758473
## 5 5 800 0.1167593 0.5443230 0.08744776
## 5 5 900 0.1168440 0.5440714 0.08756333
## 5 5 1000 0.1168717 0.5444031 0.08748994
## 5 10 100 0.1191319 0.5228682 0.09129258
## 5 10 200 0.1169277 0.5393692 0.08910966
## 5 10 300 0.1160904 0.5467017 0.08774226
## 5 10 400 0.1160526 0.5473892 0.08782760
## 5 10 500 0.1160895 0.5472092 0.08763751
## 5 10 600 0.1163602 0.5457502 0.08766444
## 5 10 700 0.1163370 0.5469193 0.08780567
## 5 10 800 0.1164528 0.5466926 0.08801583
## 5 10 900 0.1164304 0.5470955 0.08798276
## 5 10 1000 0.1165244 0.5467362 0.08792449
## 10 5 100 0.1149405 0.5552740 0.08711143
## 10 5 200 0.1133828 0.5668386 0.08549391
## 10 5 300 0.1132601 0.5674924 0.08523105
## 10 5 400 0.1131027 0.5689499 0.08534686
## 10 5 500 0.1129297 0.5704178 0.08511479
## 10 5 600 0.1128214 0.5713018 0.08518070
## 10 5 700 0.1129052 0.5708420 0.08517432
## 10 5 800 0.1129350 0.5707847 0.08513301
## 10 5 900 0.1129632 0.5705997 0.08512540
## 10 5 1000 0.1129606 0.5706867 0.08507831
## 10 10 100 0.1135823 0.5656884 0.08625103
## 10 10 200 0.1131019 0.5700441 0.08536386
```

```
##      10      10      300      0.1130614  0.5710046  0.08495719
##      10      10      400      0.1134918  0.5682695  0.08524655
##      10      10      500      0.1136005  0.5675580  0.08535124
##      10      10      600      0.1134173  0.5689492  0.08540763
##      10      10      700      0.1134000  0.5693393  0.08532770
##      10      10      800      0.1134899  0.5686146  0.08543362
##      10      10      900      0.1135501  0.5682792  0.08553778
##      10      10     1000      0.1136177  0.5678470  0.08556643
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 600, interaction.depth =
## 10, shrinkage = 0.1 and n.minobsinnode = 5.

##      n.trees interaction.depth shrinkage n.minobsinnode
## 26      600              10      0.1              5
```



```
##      Rsquared      RMSE
## 1 0.5469193 0.116337
```

Tuning parameter 'shrinkage' was held constant at a value of 0.1. RMSE was used to select the optimal model using the smallest value. The final values used for the model were n.trees = 600, interaction.depth = 10, shrinkage = 0.1 and n.minobsinnode = 5. The R^2 and RMSE are 0.54 and 0.11 respectively on training data. This is the best Rsquared so far.

Random Forest

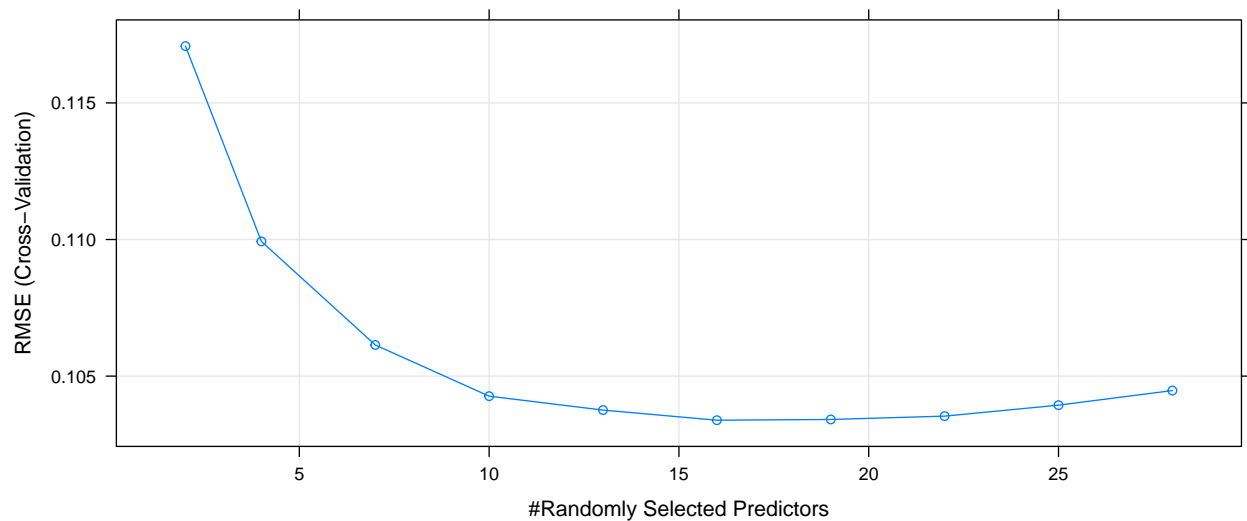
Random forest consists of a large number of individual decision trees that work as an ensemble. Each model in the ensemble is used to generate a prediction for a new sample and these predictions are then averaged to give the forest's prediction. Since the algorithm randomly selects predictors at each split, tree correlation gets reduced as compared to bagging. In random forest algorithm, we first select the number of models to build and then loop through this number and train a tree model. Once done then average the predictions to get overall prediction. In random forests, trees are created independently, each tree is created having maximum depth and each tree contributes equally in the final model.

```
## Random Forest
##
## 1930 samples
```



```
## 28 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1737, 1737, 1737, 1737, 1737, 1737, ...
## Resampling results across tuning parameters:
##
## mtry RMSE Rsquared MAE
## 2 0.1170800 0.5833714 0.09028846
## 4 0.1099305 0.6240126 0.08329171
## 7 0.1061407 0.6419264 0.07955917
## 10 0.1042662 0.6505196 0.07754109
## 13 0.1037564 0.6514222 0.07669700
## 16 0.1033861 0.6513098 0.07623180
## 19 0.1034130 0.6491251 0.07593447
## 22 0.1035350 0.6469633 0.07567737
## 25 0.1039367 0.6423775 0.07573747
## 28 0.1044708 0.6374838 0.07603538
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 16.

## mtry
## 6 16
```



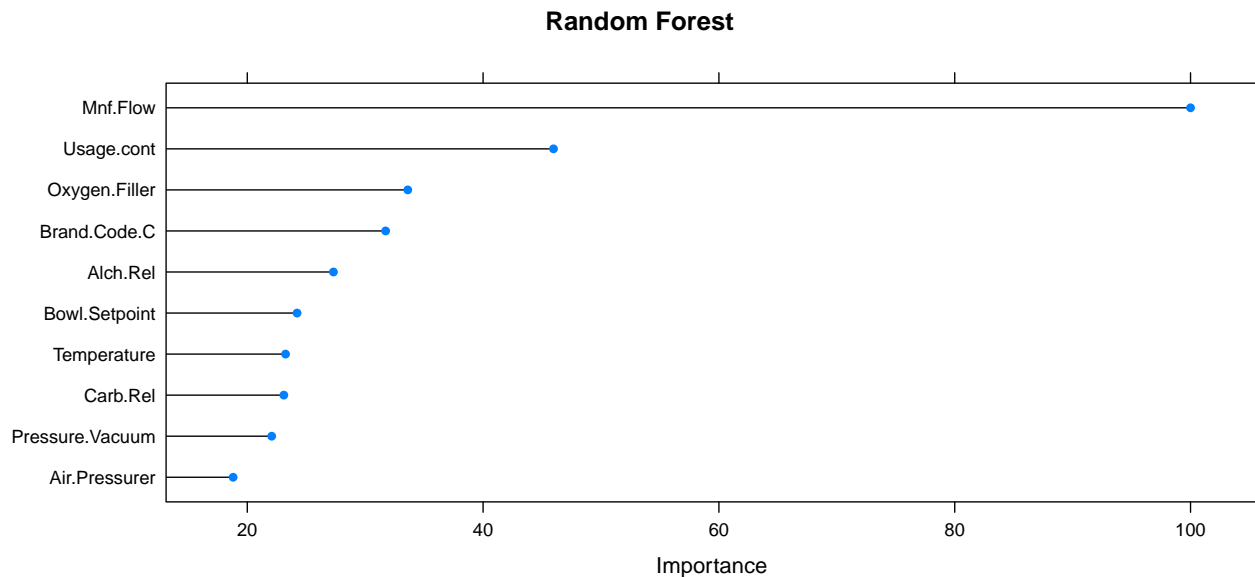
```
## Rsquared RMSE
## 1 0.6513098 0.1033861
```

RMSE was used to select the optimal model using the smallest value. The final value used for the model was $mtry = 16$. It has R^2 as 0.65 and RMSE as 0.10. Both these values are best among the models used so far.

Lets see the informative variables found by Random Forest models. we will use `varImp` method to find these variables.

```
## rf variable importance
##
## only 20 most important variables shown (out of 28)
##
```

```
## Overall
## Mnf.Flow 100.000
## Usage.cont 45.970
## Oxygen.Filler 33.613
## Brand.Code.C 31.732
## Alch.Rel 27.308
## Bowl.Setpoint 24.221
## Temperature 23.245
## Carb.Rel 23.100
## Pressure.Vacuum 22.072
## Air.Pressurer 18.798
## Carb.Flow 17.838
## Carb.Pressure1 17.346
## MFR 13.262
## PC.Volume 10.958
## Fill.Pressure 9.119
## Carb.Volume 8.747
## Hyd.Pressure2 8.268
## Fill.Ounces 7.433
## PSC 6.693
## Hyd.Pressure4 5.604
```



From above plot, it is evident **Mnf.Flow** is the most informative variable for PH response variable.

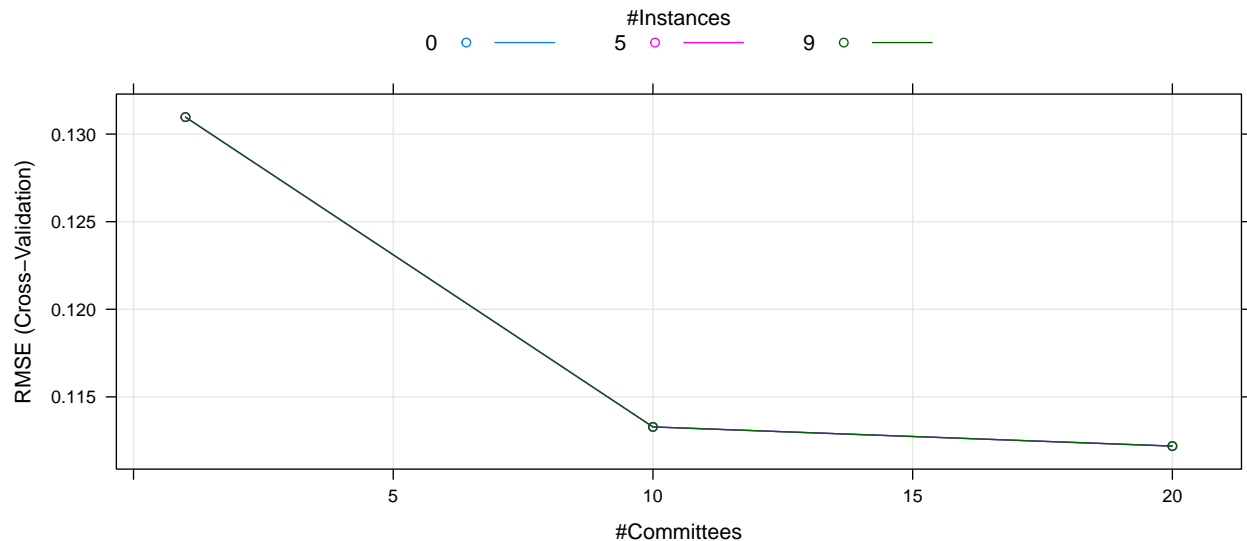
Cubist

Cubist is a rule-based model. A tree is built where the terminal leaves contain linear regression models. These models are based upon the predictors used in previous splits along with intermediate models. The tree is reduced to a set of rules which initially are paths from the top of the tree to the bottom. Rules are eliminated via pruning or combined and the candidate variables for the models are the predictors that were pruned away. .

```
## Cubist
##
## 1930 samples
## 28 predictor
```

```
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1737, 1737, 1737, 1737, 1737, 1737, ...
## Resampling results across tuning parameters:
##
##   committees neighbors RMSE      Rsquared  MAE
##   1           0       0.1309689  0.4563136  0.09299230
##   1           5       0.1309689  0.4563136  0.09299230
##   1           9       0.1309689  0.4563136  0.09299230
##   10          0       0.1132853  0.5698369  0.08344530
##   10          5       0.1132853  0.5698369  0.08344530
##   10          9       0.1132853  0.5698369  0.08344530
##   20          0       0.1121916  0.5786100  0.08294355
##   20          5       0.1121916  0.5786100  0.08294355
##   20          9       0.1121916  0.5786100  0.08294355
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were committees = 20 and neighbors = 0.

##   committees neighbors
## 7           20         0
```



```
##   Rsquared    RMSE
## 1  0.57861  0.1121916
```

RMSE was used to select the optimal model using the smallest value. The best tune for the cubist model which resulted in the smallest root mean squared error was with 20 committees. It had $RMSE = 0.112$, and $R^2 = 0.578$. So far, it covered 57% of the variability in the data than all other variables and with the low RMSE.

Select Model

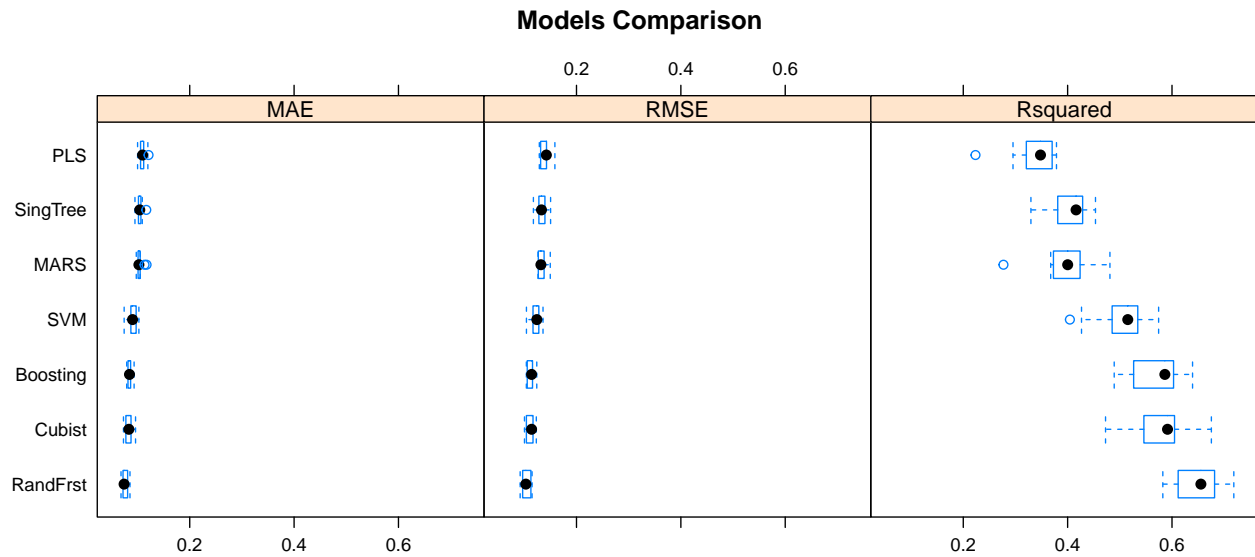
To select the best model for making predictions for evaluation data, we will look at 3 parameters.

- R^2 , which shows the variance explained by given model.
- RMSE (Root Mean Squared Error), which is the std deviation of the residuals.

- MAE (Mean Absolute Error), which is avg of all absolute errors.

Here we will summarize the resampling to compare the above 3 values among all the models followed by checking the prediction on validation data which we reserved earlier during data partition.

```
##
## Call:
## summary.resamples(object = resamples(list(PLS = pls_model, MARS =
##   mars_model, SVM = svm_model, RandFrst = rf_model, Cubist =
##   cubist_model, SingTree = st_model, Boosting = gbm_model)))
##
## Models: PLS, MARS, SVM, RandFrst, Cubist, SingTree, Boosting
## Number of resamples: 10
##
## MAE
##           Min.      1st Qu.      Median      Mean      3rd Qu.      Max. NA's
## PLS      0.10007430 0.10632857 0.10955896 0.10972243 0.11147862 0.12102507    0
## MARS      0.09755457 0.10103709 0.10257144 0.10430716 0.10479898 0.11728296    0
## SVM       0.07431813 0.08783748 0.09039245 0.09101076 0.09603653 0.10257755    0
## RandFrst  0.06831115 0.07221768 0.07417746 0.07623180 0.08048781 0.08547669    0
## Cubist    0.07304460 0.07819699 0.08336354 0.08294355 0.08680539 0.09616592    0
## SingTree  0.09505792 0.10204748 0.10426442 0.10440681 0.10611650 0.11670546    0
## Boosting  0.07920377 0.08203256 0.08455857 0.08518070 0.08656302 0.09338149    0
##
## RMSE
##           Min.      1st Qu.      Median      Mean      3rd Qu.      Max. NA's
## PLS      0.12877712 0.13210324 0.1422288 0.1405086 0.1425127 0.1585968    0
## MARS      0.12621745 0.12681807 0.1317765 0.1341767 0.1372495 0.1497212    0
## SVM       0.10408448 0.11689184 0.1240176 0.1222248 0.1274946 0.1358440    0
## RandFrst  0.09204647 0.09706146 0.1029516 0.1033861 0.1112461 0.1148481    0
## Cubist    0.10031577 0.10550214 0.1139246 0.1121916 0.1163640 0.1230009    0
## SingTree  0.11741724 0.12900263 0.1329154 0.1333503 0.1380354 0.1503560    0
## Boosting  0.10388119 0.10737048 0.1139532 0.1128214 0.1154602 0.1238834    0
##
## Rsquared
##           Min.      1st Qu.      Median      Mean      3rd Qu.      Max. NA's
## PLS      0.2234196 0.3216855 0.3480238 0.3344473 0.3671433 0.3786303    0
## MARS      0.2770408 0.3734935 0.4001130 0.3951300 0.4217082 0.4810568    0
## SVM       0.4043236 0.4854691 0.5152359 0.5026551 0.5327833 0.5743832    0
## RandFrst  0.5824412 0.6175968 0.6553779 0.6513098 0.6796676 0.7184821    0
## Cubist    0.4726667 0.5517988 0.5914461 0.5786100 0.6037048 0.6754578    0
## SingTree  0.3296664 0.3836444 0.4161688 0.4028576 0.4281188 0.4533947    0
## Boosting  0.4892704 0.5377983 0.5863510 0.5713018 0.6007364 0.6394236    0
```



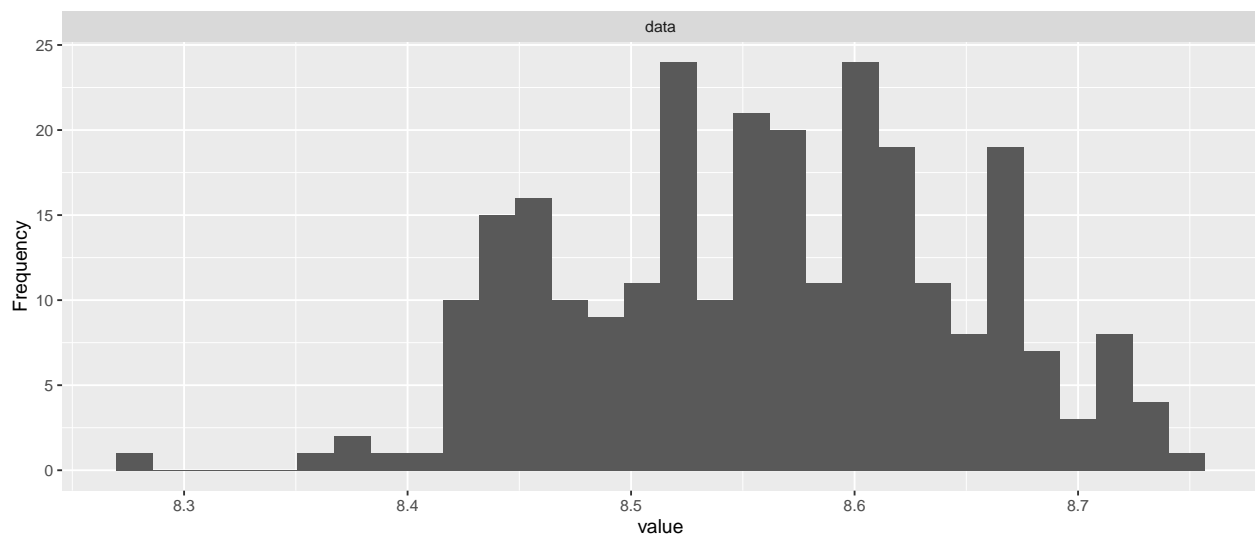
```
##          RMSE  Rsquared    MAE
## PLS      0.1371313 0.3722429 0.10861129
## MARS     0.1302255 0.4347408 0.10245641
## SVM      0.1175582 0.5395039 0.08694041
## SingTree 0.1314174 0.4238484 0.10420760
## RandFrst 0.1004640 0.6737302 0.07310555
## Boosting 0.1113162 0.5897395 0.08321032
## Cubist   0.1029136 0.6505194 0.07567148
```

We can see here Random Forest performed the best among all the models tried considering the 3 metrics Rsquared, RMSE and MAE, we identified earlier.

Prediction

Based on the analysis so far, it is confirmed that the Random Forest model is the optimal model. we will now use it to predict PH values of evaluation dataset and then write it in csv.

Here are the predicted values of PH for evaluation dataset.



Conclusion

After extracting the data from the given files, we did first perform data exploration which helped us to find missing data, correlation among the variables and outliers. Next we performed steps for data preparation that included handling missing and outliers through mice, creating dummy vars for a categorical variable Brand Code, remove highly correlated variables, transform data for Normality and finally data partition of 75% and 25% for training and validation respectively. We then trained various models using linear regression, non linear and Tree based models. We finally found the optimal model as Random Forest for predicting PH values for evaluation data.

We notice that all the values predicted are greater than 8. This value translates that the beverage made is alkaline. At the start of this study, we were not known about the nature of the ABC Beverage company i.e. what type of beverage manufacturer it was. But from this study we can conclude that this company mainly produces alkaline beverages like water, tea, fruit drinks and all.

References

- Applied Predictive Modeling. Max Kuhn and Kjell Johnson
- <https://machinelearningmastery.com/pre-process-your-dataset-in-r/>
- <https://www.analyticsvidhya.com/blog/2016/03/tutorial-powerful-packages-imputing-missing-values/>
- <https://newalbanysmiles.com/ph-values-of-common-beverages/>

Code Appendix

```
knitr::opts_chunk$set(echo=FALSE, error=FALSE, warning=FALSE, message=FALSE, fig.align="center", fig.wid
# Libraries
library(readxl)
library(tidyverse)
library(caret)
library(doParallel)
library(DataExplorer)
library(psych)
library(mice)
library(MASS)
library(caret)
library(AppliedPredictiveModeling)
library(lars)
library(pls)
library(earth)
library(Cubist)
library(randomForest)
library(DT)

set.seed(624)
# download training data from git repo
temp.file <- tempfile(fileext = ".xlsx")
download.file(url="https://github.com/DATA624-PredictiveAnalytics-Project2/Project2/blob/main/StudentDa
              destfile = temp.file,
              mode = "wb",
              quiet = TRUE)

# read excel for training data
```

```

train.df <- read_excel(temp.file, skip=0)

# download testing data from git repo
download.file(url="https://github.com/DATA624-PredictiveAnalytics-Project2/Project2/blob/main/StudentEv
             destfile = temp.file,
             mode = "wb",
             quiet = TRUE)

# read excel for testing data
test.df <- read_excel(temp.file, skip=0)

# transform Brand.code to factor
train.df$`Brand Code` = as.factor(train.df$`Brand Code`)
test.df$`Brand Code` = as.factor(test.df$`Brand Code`)
glimpse(train.df)
describe(train.df) %>% dplyr::select(-vars, -trimmed, -mad, -se)
plot_histogram(train.df, geom_histogram_args = list("fill" = "tomato4"))
# log histograms
plot_histogram(train.df, scale_x = "log10", geom_histogram_args = list("fill" = "springgreen4"))
colSums(is.na(train.df))
plot_missing(train.df[-1])
forcorr <- train.df[complete.cases(train.df),-1]
corrplot::corrplot(cor(forcorr), method = 'ellipse', type = 'lower')

# boxplot
par(mfrow = c(3,4))
for(i in colnames(train.df[-1])){
  boxplot(train.df[,i], xlab = names(train.df[i]),
          main = names(train.df[i]), col="blue", horizontal = T)
}
set.seed(317)

# Training set
train.df.clean <- mice(data.frame(train.df), method = 'rf', m=2, maxit = 2, print=FALSE)
train.df.clean <- complete(train.df.clean)

nzv_preds <- nearZeroVar(train.df.clean)
train.df.clean <- train.df.clean[,-nzv_preds]
set.seed(317)

# Testing set
test.df.clean <- mice(data.frame(test.df), method = 'rf', m=2, maxit = 2, print=FALSE)
test.df.clean <- complete(test.df.clean)
set.seed(317)
dum.brandcode <- dummyVars(PH ~ Brand.Code, data = train.df.clean)
dum.train.predict <- predict(dum.brandcode, train.df.clean)
train.df.clean <- cbind(dum.train.predict, train.df.clean) %>% dplyr::select(-Brand.Code)
set.seed(317)
dum.brandcode <- dummyVars( ~ Brand.Code, data = test.df.clean)
dum.test.predict <- predict(dum.brandcode, test.df.clean)
test.df.clean <- cbind(dum.test.predict, test.df.clean) %>% dplyr::select(-Brand.Code)
highCorr <- findCorrelation(cor(train.df.clean), 0.90)

```

```

train.df.clean <- train.df.clean[, -highCorr]
set.seed(317)
preproc_traindf <- preProcess(train.df.clean, method = "YeoJohnson")
train.df.clean <- predict(preproc_traindf, train.df.clean)
set.seed(317)
preproc_testdf <- preProcess(test.df.clean, method = "YeoJohnson")
test.df.clean <- predict(preproc_testdf, test.df.clean)
set.seed(317)

partition <- createDataPartition(train.df.clean$PH, p=0.75, list = FALSE)

# training/validation partition for independent variables
X.train <- train.df.clean[partition, ] %>% dplyr::select(-PH)
X.test <- train.df.clean[-partition, ] %>% dplyr::select(-PH)

# training/validation partition for dependent variable PH
y.train <- train.df.clean$PH[partition]
y.test <- train.df.clean$PH[-partition]
set.seed(317)

lm_model <- lm(y.train ~ ., data = X.train)
summary(lm_model)
set.seed(317)

# tune pls model
pls_model <- train(x=X.train,
                  y=y.train,
                  method="pls",
                  metric="Rsquared",
                  tuneLength=10,
                  trControl=trainControl(method = "cv")
                  )

pls_model
pls_model$bestTune
plot(pls_model)
pls_model$results %>%
  filter(ncomp == pls_model$bestTune$ncomp) %>%
  dplyr::select(ncomp, RMSE, Rsquared)
data.frame(Rsquared=pls_model[["results"]][["Rsquared"]][as.numeric(rownames(pls_model$bestTune))],
           RMSE=pls_model[["results"]][["RMSE"]][as.numeric(rownames(pls_model$bestTune))])
set.seed(317)
marsGrid <- expand.grid(.degree=1:2, .nprune=2:30)
mars_model <- train(x=X.train,
                  y=y.train,
                  method = "earth",
                  tuneGrid = marsGrid,
                  trControl = trainControl(method = "cv"))

# final parameters
mars_model$bestTune
# plot RMSE
plot(mars_model)

```



```

summary(mars_model$finalModel)
data.frame(Rsquared=mars_model[["results"]][["Rsquared"]][as.numeric(rownames(mars_model$bestTune))],
           RMSE=mars_model[["results"]][["RMSE"]][as.numeric(rownames(mars_model$bestTune))])
set.seed(317)
svm_model <- train(x=X.train,
                  y=y.train,
                  method = "svmRadial",
                  tuneLength = 10,
                  trControl = trainControl(method = "cv"))

svm_model
summary(svm_model$finalModel)
# plot RMSE
plot(svm_model)
data.frame(Rsquared=svm_model[["results"]][["Rsquared"]][as.numeric(rownames(svm_model$bestTune))],
           RMSE=svm_model[["results"]][["RMSE"]][as.numeric(rownames(svm_model$bestTune))])
set.seed(317)

st_model <- train(x=X.train,
                 y=y.train,
                 method = "rpart",
                 tuneLength = 10,
                 trControl = trainControl(method = "cv"))

st_model
st_model$bestTune
# plot RMSE
plot(st_model)
data.frame(Rsquared=st_model[["results"]][["Rsquared"]][as.numeric(rownames(st_model$bestTune))],
           RMSE=st_model[["results"]][["RMSE"]][as.numeric(rownames(st_model$bestTune))])
set.seed(317)

# boosting regression trees via stochastic gradient boosting machines

gbmGrid <- expand.grid(interaction.depth = c(5,10),
                      n.trees = seq(100, 1000, by = 100),
                      shrinkage = 0.1,
                      n.minobsinnode = c(5,10))

gbm_model <- train(x=X.train,
                  y=y.train,
                  method = "gbm",
                  tuneGrid = gbmGrid,
                  trControl = trainControl(method = "cv"),
                  verbose = FALSE)

gbm_model
gbm_model$bestTune
plot(gbm_model)
data.frame(Rsquared=gbm_model[["results"]][["Rsquared"]][as.numeric(rownames(gbm_model$bestTune))],
           RMSE=gbm_model[["results"]][["RMSE"]][as.numeric(rownames(gbm_model$bestTune))])
set.seed(317)

```

```

rf_model <- train(x=X.train,
                 y=y.train,
                 method = "rf",
                 tuneLength = 10,
                 trControl = trainControl(method = "cv"))

rf_model
rf_model$bestTune
plot(rf_model)
data.frame(Rsquared=rf_model[["results"]][["Rsquared"]][as.numeric(rownames(rf_model$bestTune))],
           RMSE=rf_model[["results"]][["RMSE"]][as.numeric(rownames(rf_model$bestTune))])
varImp(rf_model)
plot(varImp(rf_model), top=10, main="Random Forest")
set.seed(317)

cubist_model <- train(x=X.train,
                     y=y.train,
                     method = "cubist",
                     tuneLength = 10,
                     trControl = trainControl(method = "cv"))

cubist_model
cubist_model$bestTune
plot(cubist_model)
data.frame(Rsquared=cubist_model[["results"]][["Rsquared"]][as.numeric(rownames(cubist_model$bestTune))],
           RMSE=cubist_model[["results"]][["RMSE"]][as.numeric(rownames(cubist_model$bestTune))])
set.seed(317)
summary(resamples(list(PLS=pls_model, MARS=mars_model, SVM=svm_model, RandFrst=rf_model, Cubist=cubist_model)))
bwplot(resamples(list(PLS=pls_model, MARS=mars_model, SVM=svm_model, RandFrst=rf_model, Cubist=cubist_model)))
set.seed(317)

pls_pred <- predict(pls_model, newdata = X.test)
mars_pred <- predict(mars_model, newdata = X.test)
svm_pred <- predict(svm_model, newdata = X.test)
rf_pred <- predict(rf_model, newdata = X.test)
cubist_pred <- predict(cubist_model, newdata = X.test)
st_pred <- predict(st_model, newdata = X.test)
gbm_pred <- predict(gbm_model, newdata = X.test)

data.frame(rbind(PLS=postResample(pred=pls_pred,obs = y.test),
                    MARS=postResample(pred=mars_pred,obs = y.test),
                    SVM=postResample(pred=svm_pred,obs = y.test),
                    SingTree=postResample(pred=st_pred,obs = y.test),
                    RandFrst=postResample(pred=rf_pred,obs = y.test),
                    Boosting=postResample(pred=gbm_pred,obs = y.test),
                    Cubist=postResample(pred=cubist_pred,obs = y.test)))
set.seed(317)

# remove PH from evaluation data
test.df.clean <- test.df.clean %>% dplyr::select(-PH)

# predict final PH values
test.df.clean$PH <- predict(rf_model, newdata = test.df.clean)

```

```
plot_histogram(test.df.clean$PH)
write.csv(test.df.clean$PH, "StudentEvaluations_PHPredictions.csv")
```