DATA 624 Fall 2017: Project 2

Team 3: Michael Lennon, Ken Markus, Albania Nicasio, Dan Smilowitz, Logan Thomson

December 12, 2017

# Data Exploration

# open file  
path <- "https://raw.githubusercontent.com/kennygfm/fall624-group3/master/StudentData.csv"  
con <- file(path, open="r")  
  
# "Student" soft drink data  
ph\_data <- read.csv(con, na.strings = c("", " "), stringsAsFactors = FALSE)  
  
# close file  
close(con)  
  
# convert values read as integers to numeric  
ph\_data[, c(16, 18, 21, 28)] <- sapply(ph\_data[, c(16, 18, 21, 28)], as.numeric)

The structure of the data is presented below:

str(ph\_data)

'data.frame': 2571 obs. of 33 variables:  
 $ Brand.Code : chr "B" "A" "B" "A" ...  
 $ Carb.Volume : num 5.34 5.43 5.29 5.44 5.49 ...  
 $ Fill.Ounces : num 24 24 24.1 24 24.3 ...  
 $ PC.Volume : num 0.263 0.239 0.263 0.293 0.111 ...  
 $ Carb.Pressure : num 68.2 68.4 70.8 63 67.2 66.6 64.2 67.6 64.2 72 ...  
 $ Carb.Temp : num 141 140 145 133 137 ...  
 $ PSC : num 0.104 0.124 0.09 NA 0.026 0.09 0.128 0.154 0.132 0.014 ...  
 $ PSC.Fill : num 0.26 0.22 0.34 0.42 0.16 0.24 0.4 0.34 0.12 0.24 ...  
 $ PSC.CO2 : num 0.04 0.04 0.16 0.04 0.12 0.04 0.04 0.04 0.14 0.06 ...  
 $ Mnf.Flow : num -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...  
 $ Carb.Pressure1 : num 119 122 120 115 118 ...  
 $ Fill.Pressure : num 46 46 46 46.4 45.8 45.6 51.8 46.8 46 45.2 ...  
 $ Hyd.Pressure1 : num 0 0 0 0 0 0 0 0 0 0 ...  
 $ Hyd.Pressure2 : num NA NA NA 0 0 0 0 0 0 0 ...  
 $ Hyd.Pressure3 : num NA NA NA 0 0 0 0 0 0 0 ...  
 $ Hyd.Pressure4 : num 118 106 82 92 92 116 124 132 90 108 ...  
 $ Filler.Level : num 121 119 120 118 119 ...  
 $ Filler.Speed : num 4002 3986 4020 4012 4010 ...  
 $ Temperature : num 66 67.6 67 65.6 65.6 66.2 65.8 65.2 65.4 66.6 ...  
 $ Usage.cont : num 16.2 19.9 17.8 17.4 17.7 ...  
 $ Carb.Flow : num 2932 3144 2914 3062 3054 ...  
 $ Density : num 0.88 0.92 1.58 1.54 1.54 1.52 0.84 0.84 0.9 0.9 ...  
 $ MFR : num 725 727 735 731 723 ...  
 $ Balling : num 1.4 1.5 3.14 3.04 3.04 ...  
 $ Pressure.Vacuum : num -4 -4 -3.8 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 ...  
 $ PH : num 8.36 8.26 8.94 8.24 8.26 8.32 8.4 8.38 8.38 8.5 ...  
 $ Oxygen.Filler : num 0.022 0.026 0.024 0.03 0.03 0.024 0.066 0.046 0.064 0.022 ...  
 $ Bowl.Setpoint : num 120 120 120 120 120 120 120 120 120 120 ...  
 $ Pressure.Setpoint: num 46.4 46.8 46.6 46 46 46 46 46 46 46 ...  
 $ Air.Pressurer : num 143 143 142 146 146 ...  
 $ Alch.Rel : num 6.58 6.56 7.66 7.14 7.14 7.16 6.54 6.52 6.52 6.54 ...  
 $ Carb.Rel : num 5.32 5.3 5.84 5.42 5.44 5.44 5.38 5.34 5.34 5.34 ...  
 $ Balling.Lvl : num 1.48 1.56 3.28 3.04 3.04 3.02 1.44 1.44 1.44 1.38 ...

The data contains 2571 observations across 33 variables. The first variable, Brand.Code, is a character; the remaining 32 variables are numeric. Summary statistics of the variables are presented below:

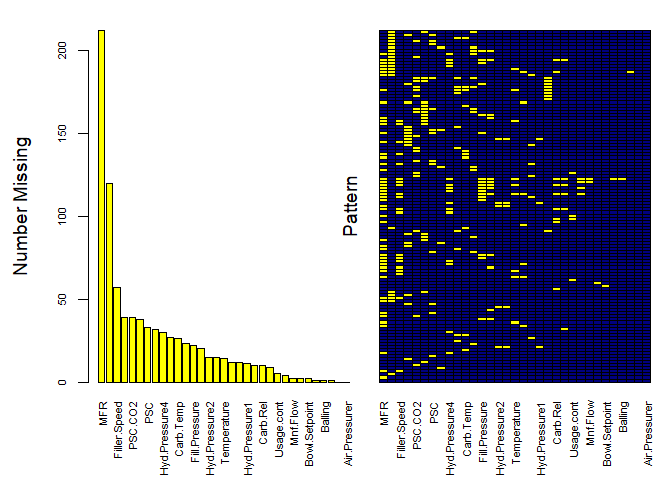
library(e1071)  
library(pander)  
  
means <- sapply(ph\_data[-1], function(y) mean(y, na.rm = TRUE))  
medians <- sapply(ph\_data[-1], function(y) median(y, na.rm = TRUE))  
IQRs <- sapply(ph\_data[-1], function(y) IQR(y, na.rm = TRUE))  
vars <- sapply(ph\_data[-1], function(y) var(y, na.rm = TRUE))  
skews <- sapply(ph\_data[-1], function(y) skewness(as.numeric(y), na.rm = TRUE))  
cors <- as.vector(cor(ph\_data$PH, ph\_data[,2:ncol(ph\_data)], use = "complete.obs"))  
NAs <- sapply(ph\_data[-1], function(y) sum(length(which(is.na(y)))))  
  
soda\_summary <- data.frame(means, medians, IQRs, vars, skews, cors, NAs)  
colnames(soda\_summary) <- c("MEAN", "MEDIAN", "IQR", "Var", "SKEW", "$r\_{PH}$", "NAs")  
soda\_summary <- round(soda\_summary, 2)  
  
pander(soda\_summary)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | MEAN | MEDIAN | IQR | Var | SKEW |  | NAs |
| **Carb.Volume** | 5.37 | 5.35 | 0.16 | 0.01 | 0.39 | 0.07 | 10 |
| **Fill.Ounces** | 23.97 | 23.97 | 0.11 | 0.01 | -0.02 | -0.12 | 38 |
| **PC.Volume** | 0.28 | 0.27 | 0.07 | 0 | 0.34 | 0.1 | 39 |
| **Carb.Pressure** | 68.19 | 68.2 | 5 | 12.52 | 0.18 | 0.08 | 27 |
| **Carb.Temp** | 141.1 | 140.8 | 5.4 | 16.3 | 0.25 | 0.03 | 26 |
| **PSC** | 0.08 | 0.08 | 0.06 | 0 | 0.85 | -0.07 | 33 |
| **PSC.Fill** | 0.2 | 0.18 | 0.16 | 0.01 | 0.93 | -0.02 | 23 |
| **PSC.CO2** | 0.06 | 0.04 | 0.06 | 0 | 1.73 | -0.09 | 39 |
| **Mnf.Flow** | 24.57 | 65.2 | 240.8 | 14276 | 0 | -0.46 | 2 |
| **Carb.Pressure1** | 122.6 | 123.2 | 6.4 | 22.49 | 0.05 | -0.12 | 32 |
| **Fill.Pressure** | 47.92 | 46.4 | 4 | 10.1 | 0.55 | -0.32 | 22 |
| **Hyd.Pressure1** | 12.44 | 11.4 | 20.2 | 154.6 | 0.78 | -0.05 | 11 |
| **Hyd.Pressure2** | 20.96 | 28.6 | 34.6 | 268.5 | -0.3 | -0.22 | 15 |
| **Hyd.Pressure3** | 20.46 | 27.6 | 33.4 | 255.2 | -0.32 | -0.27 | 15 |
| **Hyd.Pressure4** | 96.29 | 96 | 16 | 172.2 | 0.55 | -0.17 | 30 |
| **Filler.Level** | 109.2 | 118.4 | 21.7 | 246.4 | -0.85 | 0.35 | 20 |
| **Filler.Speed** | 3687 | 3982 | 110 | 594164 | -2.87 | -0.04 | 57 |
| **Temperature** | 65.97 | 65.6 | 1.2 | 1.91 | 2.39 | -0.18 | 14 |
| **Usage.cont** | 20.99 | 21.79 | 5.39 | 8.87 | -0.54 | -0.36 | 5 |
| **Carb.Flow** | 2468 | 3028 | 2042 | 1152824 | -0.99 | 0.23 | 2 |
| **Density** | 1.17 | 0.98 | 0.72 | 0.14 | 0.53 | 0.1 | 1 |
| **MFR** | 704 | 724 | 24.7 | 5461 | -5.09 | -0.05 | 212 |
| **Balling** | 2.2 | 1.65 | 1.8 | 0.87 | 0.59 | 0.08 | 1 |
| **Pressure.Vacuum** | -5.22 | -5.4 | 0.6 | 0.32 | 0.53 | 0.22 | 0 |
| **PH** | 8.55 | 8.54 | 0.24 | 0.03 | -0.29 | 1 | 4 |
| **Oxygen.Filler** | 0.05 | 0.03 | 0.04 | 0 | 2.66 | 0.16 | 12 |
| **Bowl.Setpoint** | 109.3 | 120 | 20 | 234.2 | -0.97 | 0.36 | 2 |
| **Pressure.Setpoint** | 47.62 | 46 | 4 | 4.16 | 0.2 | -0.31 | 12 |
| **Air.Pressurer** | 142.8 | 142.6 | 0.8 | 1.47 | 2.25 | -0.01 | 0 |
| **Alch.Rel** | 6.9 | 6.56 | 0.7 | 0.26 | 0.88 | 0.17 | 9 |
| **Carb.Rel** | 5.44 | 5.4 | 0.2 | 0.02 | 0.5 | 0.2 | 10 |
| **Balling.Lvl** | 2.05 | 1.48 | 1.76 | 0.76 | 0.59 | 0.11 | 1 |

## Missing Values

As shown in the above summary, there are missing values across the variables -- the frequency and pattern of these missing values are presented below:

library(VIM)  
  
aggr(ph\_data, sortVars = TRUE, bar = FALSE, prop = FALSE, gap = 1, cex.axis = 0.7,  
 col = c("navyblue", "yellow"), ylab = c("Number Missing", "Pattern"))

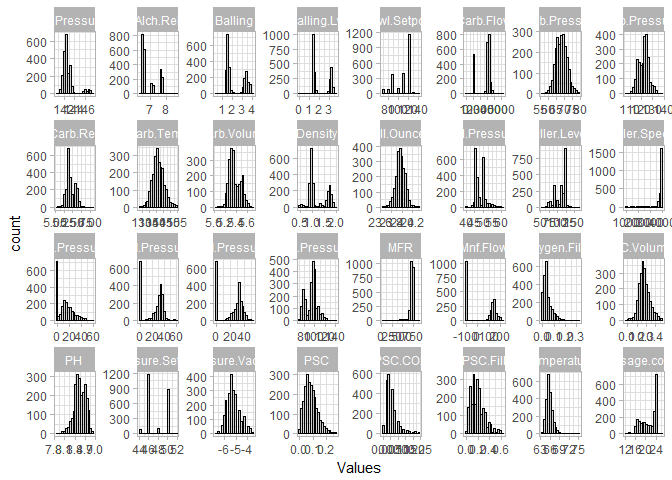


The variable MFR has over 200 missing values, and the variable Brand.Code is missing 120 values. Due to these high proportions of missingness, observations missing these variables are dropped:

library(plyr) # loaded for later dependecies to avoid conflicts with dplyr  
library(tidyverse)  
  
ph\_data <- drop\_na(ph\_data, MFR, Brand.Code)

Numeric variables are plotted below to determine the best method of imputation:

theme\_set(theme\_light())  
  
ph\_data %>%   
 select(-Brand.Code) %>%   
 gather(Variable, Values) %>%   
 ggplot(aes(x = Values)) +  
 geom\_histogram(alpha = 0.25, col = "black", bins = 20) +  
 facet\_wrap(~ Variable, scales = "free", nrow = 4)



Many of the variables have normal or somewhat-normal distributions, and appear to be continuous variables. Many of the variables show varying levels of skewness, and a few of the skewed distributions follow an almost chi-squared or log-normal distribution. These are predictors (PSC.Distribution and PSC.C02.Distribution) where transformations should be considered.

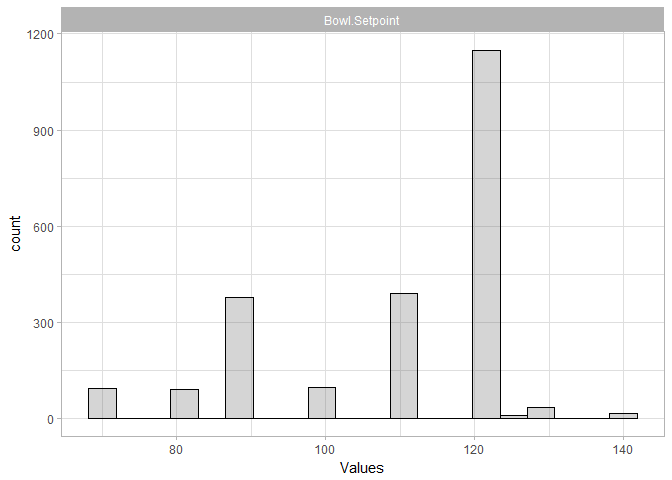
Another interesting pattern are the number of 0 values in the Hyd.Pressure(1-3) variables. The Hyd.Pressure does not follow this same pattern, so depending on the relationship between these variables and the target and/or other variables, they may be removed outright.

Some of the predictors appear to be discrete, rather than continuous distributions (Pressure.Setpoint, Alch.Rel), however, many of these variables are just constrained to a few values, but are still continuous. Out of all the variables, Bowl.Setpoint does seem to be a discrete distribution. A table of all values in the variable is below:

pander(table(ph\_data$Bowl.Setpoint))

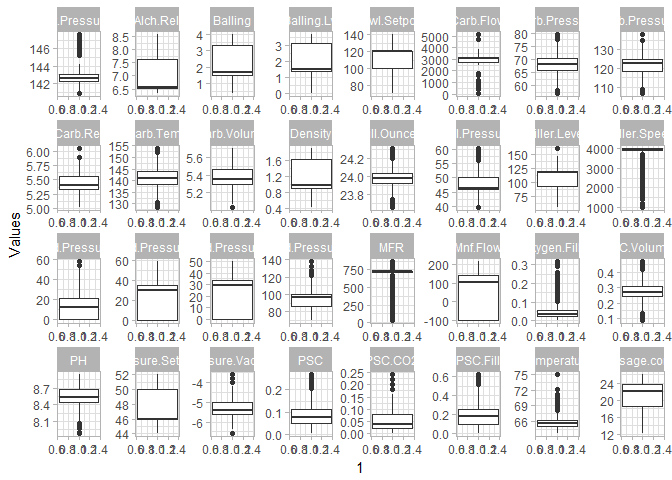
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **70** | **80** | **90** | **100** | **110** | **120** | **122** | **126** | **130** | **134** | **140** |
| 93 | 89 | 376 | 95 | 390 | 1148 | 1 | 8 | 33 | 1 | 16 |

ph\_data %>%   
 select(Bowl.Setpoint) %>%   
 gather(Variable, Values) %>%   
 ggplot(aes(x = Values)) +  
 geom\_histogram(alpha = 0.25, col = "black", bins = 20) +  
 facet\_wrap(~ Variable, scales = "free", nrow = 4)



Boxplots provide an alternate view that allows the spread of the data to be viewed a little more clearly, as well as the presence of outliers and their quantities:

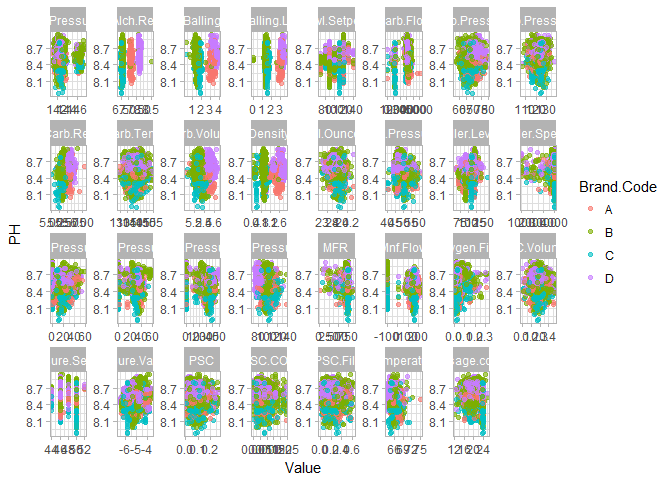
ph\_data %>%   
 select(-Brand.Code) %>%   
 gather(Variable, Values) %>%   
 ggplot(aes(x = 1, y = Values)) +  
 geom\_boxplot() +  
 facet\_wrap(~ Variable, scales = "free", nrow = 4)



A number of these variables are so highly skewed, such as Filler.Speed, Oxygen.Filler, MFR, and Air.Pressurer, that many of the observations for that predictor are recognized as outliers. This suggests that imputing with the mean will not be accurate; an alternate method should be investigated.

Before moving on to correlations between predictors, the relationship between the target variable PH and the other predictors is visualized using scatterplots:

ph\_data %>%   
 gather(-PH, -Brand.Code, key="Var", value="Value") %>%   
 ggplot(aes(x=Value, y=PH, color=Brand.Code)) +  
 geom\_point(alpha=0.6) +  
 facet\_wrap(~ Var, scales = "free", nrow=4)



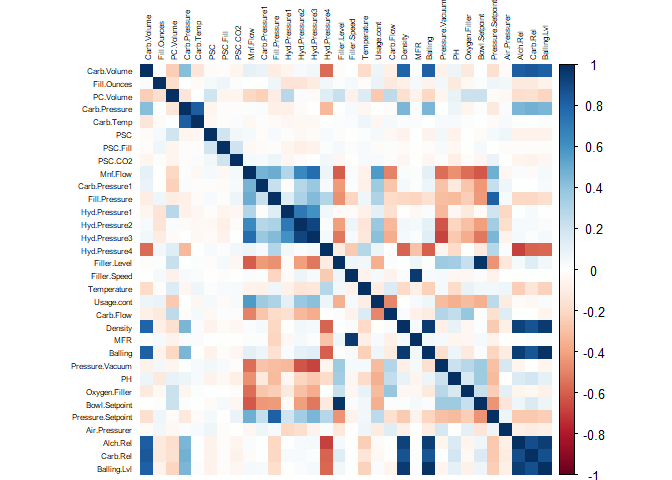
Here we see there are no strong patterns (i.e. no defined linear pattern), but there are some relationships between PH and the other variables that may be more helpful in predicting the PH level of the soft drink. Predictors like Alch.Rel, Density, and Temperature all appear to have a stronger relationship with PH - this makes sense, as these variables have more to do with the make-up of the beverage itself, rather than the filling/bottling process.

Also interesting is the clustering of the Brand.Code within the plots. As correlations are investigated, this predictor does not have a strong relationship with the target, but combined with other predictors, may be more telling.

## Correlation

The correlation between the variables is investigated:

ph\_cors <- cor(ph\_data %>% select(-Brand.Code), use="complete.obs")  
library(corrplot)  
corrplot(as.matrix(ph\_cors), method = "color", tl.cex = 0.5, tl.col = "black")



As the columns are organized in the data, some interesting patterns are present in the correlogram. Two areas show distinct positive correlations -- these are the predictors that have something to do with carbonation, and another area where different pressure levels correlate with each other. Another set of variables are negatively correlated with these pressure predictors, these have to do with the filling of the bottles, so this makes sense (Oxygen.Filler, Bowl.Setpoint, Pressure.Setpoint).

Some of these same predictors are also correlated well with the target PH variable. The top positive correlations are below:

top\_ph\_cors <- ph\_cors %>%   
 as.data.frame() %>%   
 select(Correlation = PH) %>%   
 rownames\_to\_column("Variable") %>%   
 arrange(desc(Correlation))  
  
top\_ph\_cors %>%  
 top\_n(11, Correlation) %>%   
 pander()

|  |  |
| --- | --- |
| Variable | Correlation |
| PH | 1 |
| Bowl.Setpoint | 0.3585 |
| Filler.Level | 0.3488 |
| Carb.Flow | 0.2408 |
| Pressure.Vacuum | 0.2132 |
| Carb.Rel | 0.1896 |
| Oxygen.Filler | 0.162 |
| Alch.Rel | 0.1561 |
| Balling.Lvl | 0.1033 |
| PC.Volume | 0.09775 |
| Density | 0.09534 |

The predictors with the highest negative correlation to PH are as follows:

top\_ph\_cors %>%  
 top\_n(-10, Correlation) %>%  
 arrange(Correlation) %>%  
 pander()

|  |  |
| --- | --- |
| Variable | Correlation |
| Mnf.Flow | -0.4572 |
| Usage.cont | -0.3577 |
| Pressure.Setpoint | -0.317 |
| Fill.Pressure | -0.3131 |
| Hyd.Pressure3 | -0.2605 |
| Hyd.Pressure2 | -0.2157 |
| Hyd.Pressure4 | -0.1845 |
| Temperature | -0.1815 |
| Fill.Ounces | -0.1182 |
| Carb.Pressure1 | -0.1121 |

### Intercorrelated Predictors

In addition, some of the variables are highly correlated with each other. The following pairings seem to have something to do with each other in the bottle filling process:

library(reshape2)  
ph\_cors\_tri <- ph\_cors  
  
ph\_cors\_tri[lower.tri(ph\_cors\_tri, diag = TRUE)] <- NA # prevent duplicates by taking upper triangle of matrix  
  
ph\_cors\_tri %>%   
 melt() %>%  
 as.data.frame() %>%  
 filter(!is.na(value)) %>%  
 arrange(desc(value)) %>%  
 top\_n(10) %>%  
 pander()

|  |  |  |
| --- | --- | --- |
| Var1 | Var2 | value |
| Balling | Balling.Lvl | 0.9884 |
| Filler.Level | Bowl.Setpoint | 0.9772 |
| Density | Balling.Lvl | 0.9564 |
| Density | Balling | 0.9531 |
| Filler.Speed | MFR | 0.9513 |
| Alch.Rel | Balling.Lvl | 0.9473 |
| Balling | Alch.Rel | 0.9457 |
| Density | Alch.Rel | 0.923 |
| Hyd.Pressure2 | Hyd.Pressure3 | 0.9177 |
| Alch.Rel | Carb.Rel | 0.8828 |

Among the variables with positive correlations with each other, many of them are almost perfectly correlated with one another. We may want to consider removing some of these redundant variables, perhaps the ones that are less correlated to the target variable. In addition predictors with high negative correlations with each other are below:

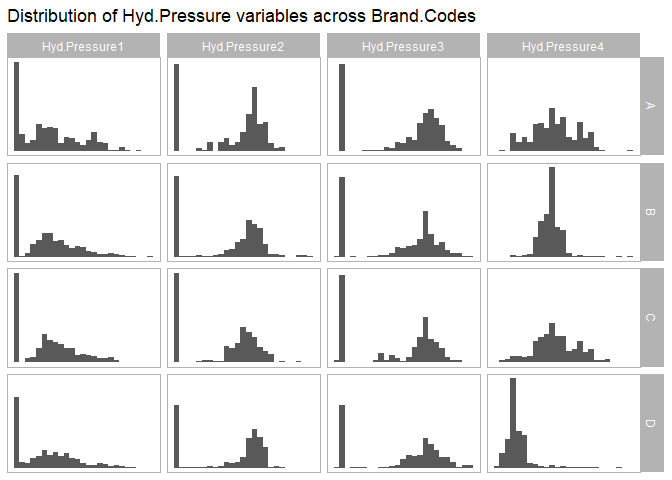
ph\_cors\_tri %>%   
 melt() %>%  
 as.data.frame() %>%  
 filter(!is.na(value)) %>%  
 arrange(value) %>%  
 top\_n(-10) %>%  
 pander()

|  |  |  |
| --- | --- | --- |
| Var1 | Var2 | value |
| Hyd.Pressure4 | Alch.Rel | -0.6856 |
| Hyd.Pressure3 | Pressure.Vacuum | -0.6723 |
| Hyd.Pressure2 | Pressure.Vacuum | -0.6282 |
| Mnf.Flow | Bowl.Setpoint | -0.6116 |
| Mnf.Flow | Filler.Level | -0.6084 |
| Hyd.Pressure4 | Balling | -0.595 |
| Hyd.Pressure4 | Density | -0.5844 |
| Hyd.Pressure4 | Carb.Rel | -0.5818 |
| Hyd.Pressure4 | Balling.Lvl | -0.575 |
| Carb.Volume | Hyd.Pressure4 | -0.5679 |

## Variable Removal

Three similarly-names variables, Hyd.Pressure1, Hyd.Pressure2, and Hyd.Pressure3show large spikes at values of 0 in the histograms above, and show high correlations. This suggests that these variables are candidates for removal. These variables, along with a fourth related variable Hyd.Pressure4, are first investigated across the four Brand.Code values:

ph\_data %>%   
 select(Brand.Code, Hyd.Pressure1:Hyd.Pressure4) %>%   
 gather(HydNum, Value, -Brand.Code) %>%   
 ggplot(aes(x = Value)) +  
 geom\_histogram(bins = 25) +  
 facet\_grid(Brand.Code ~ HydNum, scales = "free") +  
 theme(panel.grid = element\_blank()) +  
 scale\_x\_continuous(NULL, NULL, NULL) +  
 scale\_y\_continuous(NULL, NULL, NULL) +  
 ggtitle("Distribution of Hyd.Pressure variables across Brand.Codes")



These distributions suggest that there is no behavior in Hyd.Pressure1, Hyd.Pressure2, or Hyd.Pressure3 indicated by Brand.Code. For this reason, these three variables are dropped. Hyd.Pressure4 is retained, as there appear to be differences in its behavior across Brand.Code:

ph\_data <- ph\_data %>% select(-(Hyd.Pressure1:Hyd.Pressure3))

# Data Preprocessing

Before imputation, the data is split into predictors and a response:

ph\_pred <- ph\_data %>% select(-PH)  
ph\_resp <- ph\_data %>% select(PH)

## Imputation

Due to the skewness of and relationship between predictors, they are imputed using k-nearest neighbors. Prior to this imputation, predictors are centered and scaled to avoid bias in predictive models, and highly-correlated predictors are removed. The preProcess function from the caret package is capable of performing all of these operations -- per the documentation for this function:

The operations are applied in this order: zero-variance filter, near-zero variance filter, correlation filter, Box-Cox/Yeo-Johnson/exponential transformation, centering, scaling, range, imputation, PCA, ICA then spatial sign.

library(caret)  
# set up pre-processing transformation  
ph\_preproc <- preProcess(ph\_pred, method = c("knnImpute", "center", "scale", "corr"))  
# apply pre-processing to data  
ph\_pred <- predict(ph\_preproc, ph\_pred)

## Partitioning

With pre-processing complete, both predictor and response data are partitioned into a training and testing set:

# get rows for training subsets  
set.seed(100) # for replicability  
train\_rows <- createDataPartition(ph\_resp$PH, p = 0.75, list = FALSE)  
# create training sets  
ph\_pred\_train <- ph\_pred[train\_rows, ]  
ph\_resp\_train <- ph\_resp[train\_rows, ]  
# creae test sets  
ph\_pred\_test <- ph\_pred[-train\_rows, ]  
ph\_resp\_test <- ph\_resp[-train\_rows, ]

For some models used, the character predictor Brand.Code needs to be numeric. Additional data frames (for training & testing) are created with this variable converted:

ph\_pred\_train\_num <- mutate(ph\_pred\_train, Brand.Code = as.numeric(as.factor(Brand.Code)))  
ph\_pred\_test\_num <- mutate(ph\_pred\_test, Brand.Code = as.numeric(as.factor(Brand.Code)))

# Model Creation

The pre-processed data is used to fit an array of models: linear models; non-linear models; and tree based models.

## Linear Models

### Simple Linear Model

A simple linear model is fit using all of the predictors contained in ph\_pred\_train:

# create single df for fomula based model  
regressData <- data.frame(ph\_pred\_train\_num, PH = ph\_resp\_train, stringsAsFactors = FALSE)  
# train model  
lmTune <- train(PH ~ ., data = regressData, method = "lm")  
# get predictions for test set  
lmPred <- predict(lmTune, newdata = ph\_pred\_test\_num)  
# evaluate performance of test set  
lmPerf <- postResample(pred = lmPred, obs = ph\_resp\_test)

### Generalized Linear Model

Generalized linear models are extended versions of linear models which use a link function to describe the relation of the mean and the linear predictor, and a variance function. GLM can be used to model other distributions, such as binomial, Poisson, and exponential.

# fit model  
glmTune <- glm(PH ~ ., data = regressData, family = "quasipoisson")   
# get predictions & performance  
glmPred <- predict(glmTune, newdata = ph\_pred\_test\_num)  
glmPerf <- postResample(pred = glmPred, obs = ph\_resp\_test)

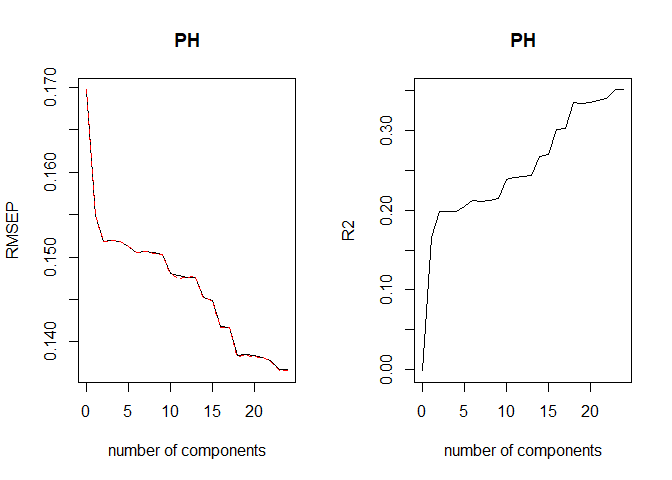
### Principal Components Regression Model

A principal components regression model seeks to lower the number of coefficients in a model by creating combinations of the variables that explain significant portions of the variance in the data. Each of the principal components generated are mutually orthogonal, so there is no risk of autocorrelation between predictors.

library(pls)  
# tune model  
pcrTune <- pcr(PH ~ ., data = regressData, scale = TRUE, validation = "CV")

The performance of the PCR model is investigated across the number of components:

par(mfrow = c(1, 2))  
validationplot(pcrTune, val.type = "RMSE")  
validationplot(pcrTune, val.type = "R2")



When doing PRC modeling, it is important to keep in mind that the main goal is to see a low cross validation error with a lower number of components than the number of variables in the dataset. In these results, it is clear that the number of components are too high for easy interpretation, resulting in no dimensionality reduction. Based on this, 18 principal components are used for prediction.

# predict & get performance  
pcrPred <- predict(pcrTune, newdata = ph\_pred\_test\_num, ncomp = 18)  
pcrPerf <- postResample(pred = pcrPred, ph\_resp\_test)

### Stepwise Backwards Linear Regression Model

The PCM model has too many principal components in this model to be easily interpretable. As an alternate, stepwise backwards linear regression is conducted:

# create full model  
full <- lm(PH ~ ., data = regressData)  
# step backwards  
stepTune <- step(full, data = regressData, direction = "backward", trace = FALSE)  
pander(summary(stepTune))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Estimate | Std. Error | t value | Pr(>|t|) |
| **(Intercept)** | 8.581 | 0.01213 | 707.5 | 0 |
| **Brand.Code** | -0.01259 | 0.004653 | -2.706 | 0.006886 |
| **Fill.Ounces** | -0.01318 | 0.003414 | -3.86 | 0.0001178 |
| **PSC.Fill** | -0.008521 | 0.003347 | -2.546 | 0.01098 |
| **Mnf.Flow** | -0.07017 | 0.006262 | -11.21 | 3.743e-28 |
| **Carb.Pressure1** | 0.02805 | 0.003847 | 7.291 | 4.734e-13 |
| **Hyd.Pressure4** | -0.02423 | 0.005259 | -4.608 | 4.377e-06 |
| **Filler.Level** | 0.02025 | 0.004815 | 4.204 | 2.755e-05 |
| **Temperature** | -0.03726 | 0.003533 | -10.55 | 3.211e-25 |
| **Usage.cont** | -0.02038 | 0.004237 | -4.809 | 1.654e-06 |
| **Carb.Flow** | 0.01966 | 0.004464 | 4.403 | 1.134e-05 |
| **Density** | -0.03645 | 0.006877 | -5.3 | 1.309e-07 |
| **MFR** | -0.01057 | 0.003543 | -2.983 | 0.002899 |
| **Pressure.Vacuum** | -0.009798 | 0.004133 | -2.371 | 0.01786 |
| **Oxygen.Filler** | -0.0243 | 0.003951 | -6.149 | 9.727e-10 |
| **Pressure.Setpoint** | -0.01302 | 0.004086 | -3.188 | 0.001461 |
| **Carb.Rel** | 0.03615 | 0.006869 | 5.263 | 1.601e-07 |

Fitting linear model: PH ~ Brand.Code + Fill.Ounces + PSC.Fill + Mnf.Flow + Carb.Pressure1 + Hyd.Pressure4 + Filler.Level + Temperature + Usage.cont + Carb.Flow + Density + MFR + Pressure.Vacuum + Oxygen.Filler + Pressure.Setpoint + Carb.Rel

|  |  |  |  |
| --- | --- | --- | --- |
| Observations | Residual Std. Error |  | Adjusted |
| 1690 | 0.1352 | 0.3714 | 0.3654 |

The variables chosen for inclusion were Brand.Code, Fill.Ounces, PSC.Fill, Mnf.Flow, Carb.Pressure1, Hyd.Pressure4, Filler.Level, Temperature, Usage.cont, Carb.Flow, Density, MFR, Pressure.Vacuum, Oxygen.Filler, Pressure.Setpoint, and Carb.Rel.

Predictions and performance are calculated from the stepwise model:

stepPred <- predict(stepTune, newdata = ph\_pred\_test\_num)  
stepPerf <- postResample(pred = stepPred, obs = ph\_resp\_test)

## Non-Linear Models

Following linear modeling, five models are created to investigate performance based on non-linear interaction with the response PH.

### MARS Model

*Multivariate adaptive regression splines* are relatively simple to execute as variables are automatically chosen and optimally transformed.

library(earth)  
# create model  
set.seed(100)  
marsTune <- earth(x = ph\_pred\_train, y = ph\_resp\_train)  
# predict & get performance  
marsPred <- predict(marsTune, newdata = ph\_pred\_test)  
marsPerf <- postResample(pred = marsPred, obs = ph\_resp\_test)

### Neural Network Model

Quite a few parameters are necessary for tuning a neural network. We proscribed the breadth in order to maintain a tolerable level of processing time. We opted to vary the number of hidden layers, and keep weight decay to three options. While results are impressive, the results of course are difficult to interpret - a byproduct of neural networks in general.

library(nnet)  
# tune model across specified measures  
nnetGrid <- expand.grid(.decay = c(0, 0.01, .1),  
 .size = c(1:10),  
 .bag = FALSE)  
  
# train model  
set.seed(100)  
nnetTune <- train(x = ph\_pred\_train, y = ph\_resp\_train,  
 method = "avNNet",  
 tuneGrid = nnetGrid,  
 linout = TRUE,  
 trace = FALSE,  
 maxit = 50)  
  
  
# predictions & performance  
nnetPred <- predict(nnetTune, newdata = ph\_pred\_test)  
nnetPerf <- postResample(pred = nnetPred, obs = ph\_resp\_test)

### Support Vector Machine Models

In this case, we expect SVMs to perform well given that we have far more samples (or observations) relative to the number of predictors. We leverage two models because we cannot assume that the appropriate hyperplane would be linear. Thus, in addition to the "linear" kernel, we leverage "radial" as well. Additional kernels should be leveraged in the future, as choosing the best one is non-trivial.

library(kernlab)  
  
#Radial kernel  
set.seed(100)  
svmRadialTune <- train(x = ph\_pred\_train\_num, y = ph\_resp\_train,  
 method = "svmRadial",   
 tuneLength = 14,  
 trControl = trainControl(method = "cv"))  
  
svmRadialPred <- predict(svmRadialTune, newdata = ph\_pred\_test\_num)  
svmRadialPerf <- postResample(pred = svmRadialPred, obs = ph\_resp\_test)   
  
#Linear kernel  
set.seed(100)  
svmLinearTune <- train(x = ph\_pred\_train\_num, y = ph\_resp\_train,  
 method = "svmLinear",   
 tuneLength = 14,  
 trControl = trainControl(method = "cv"))  
  
svmLinearPred <- predict(svmLinearTune, newdata = ph\_pred\_test\_num)  
svmLinearPerf <- postResample(pred = svmLinearPred, obs = ph\_resp\_test)

### K Nearest Neighbors Model

An advantage of K-Nearest Neighbors is its ability to handle "noisy" data and the speed with which it learns complex models. However, given the subject matter, we should not expect KNN to perform especially well as its primary use case is applied toward classifications when we know little or nothing about the distribution of data, and that is not the case here.

set.seed(100)  
knnTune <- train(x = ph\_pred\_train\_num, y = ph\_resp\_train, method = "knn",   
 tuneLength = 10)  
  
knnPred <- predict(knnTune, newdata = ph\_pred\_test\_num)   
knnPerf <- postResample(pred = knnPred, obs = ph\_resp\_test)

## Tree-Based Models

For this project we will see the constraints we encountered with our missing variables and the multiple regression. In the real world, the best model can be only as good as the variables measured by the study. We should also keep in mind that P-values can change based on the specific terms in the model. We can also appreciate that stepwise regression and best subsets regression are great tools and can get you close to the correct model. However, studies have found that they generally don't pick the correct model. With all these complications we agreed that tree-based methods are simple and useful for interpretation, but also Combining many trees can often result in dramatic improvements in prediction accuracy, at the expense of some lost interpretability. For this project we use 6 models below.

For consistency, the same training controls are used for all tree-based models:

# use 15-fold cross-validation for training  
set.seed(100)  
mdl\_ctrl <- trainControl(method = "cv", number = 15)

### Conventional Tree Model

Basic regression trees partition the data based on predictors into groups with a relatively homogeneous distribution of the response variable, and predict the mean of the appropriate group for the response.

library(rpart)  
# tree of max depth  
set.seed(100)  
rpartTune <- train(x = ph\_pred\_train, y = ph\_resp\_train,  
 method = "rpart2", trControl = mdl\_ctrl)  
rpartPred <- predict(rpartTune, newdata = ph\_pred\_test)  
rpartPerf <- postResample(pred = rpartPred, obs = ph\_resp\_test)

### Rule-Based Model

Due to their prediction being based on average, small changes in the data can drastically affect the structure (and therefore predictions) of a simple tree. An approach that may overcome this is the M5 model tree, which uses a linear regression model to predict the repose at each terminal node. A rule-based regression tree simplifies this approach by removing redundant rules resulting from splits at different parts of the tree.

library(RWeka)  
set.seed(100)  
ruleTune <- M5Rules(PH ~ ., data = mutate(ph\_pred\_train\_num, PH = ph\_resp\_train))  
rulePred <- predict(ruleTune, newdata = ph\_pred\_test\_num)  
rulePerf <- postResample(pred = rulePred, obs = ph\_resp\_test)

### Bagged Tree Model

Bootstrap aggregated (bagged) models use bootstrapped samples of the original data to fit regression models, and use these to construct an ensemble model. Each model in the ensemble is used to generate a prediction, and the predictions of each model are averaged to give a single prediction for the bagged model. The ensemble model offers a reduction in variance compared to a single regression model.

library(ipred)  
set.seed(100)  
bagTune <- train(x = ph\_pred\_train, y = ph\_resp\_train,  
 method = "treebag", trControl = mdl\_ctrl)  
bagPred <- predict(bagTune, newdata = ph\_pred\_test)  
bagPerf <- postResample(pred = bagPred, obs = ph\_resp\_test)

### Random Forest Model

While bagging offers a reduction in variance, the trees used are not completely independent of each other, since each tree (and each split of each tree), considers all original predictors. To de-correlate the trees, random forests select a bootstrapped sample of the data, then randomly select a subset of predictors to partition the trees. Once these trees have been fit, the predictions of all models in the ensemble are averaged together.

library(randomForest)  
set.seed(100)  
rfTune <- train(x = ph\_pred\_train\_num, y = ph\_resp\_train,  
 method = "rf", trControl = mdl\_ctrl, importance = TRUE)  
rfPred <- predict(rfTune, newdata = ph\_pred\_test\_num)  
rfPerf <- postResample(pred = rfPred, obs = ph\_resp\_test)

### Boosted Tree Model

Gradient boosted trees function by attempting to reduce the residuals fit by models. Initially, predictions are simply provided by the mean of the response. A regression tree is then fit based on the residuals between the actual & predicted response. This tree is used to predict the residual, and the difference between the previous residual and the predicted residual is added to the previous residual. This process is repeated for a designated number of iterations, and only the final tree is considered. This is, however, still an ensemble method, as each model fit is informed by the model before it.

library(gbm)  
set.seed(100)  
boostTune <- train(x = ph\_pred\_train\_num, y = ph\_resp\_train,  
 method = "gbm", trControl = mdl\_ctrl, verbose = FALSE)  
boostPred <- predict(boostTune, newdata = ph\_pred\_test\_num)  
boostPerf <- postResample(pred = boostPred, obs = ph\_resp\_test)

### Cubist Model

Cubist models are an extension of the rule-based model above. Terminal leaves of trees contain models based on predictors used in previous splits, with intermediate linear models at each step of the tree. Predictions made using terminal leaf's model are "smoothed" by considering predictions from models in previous nodes of the tree. The tree is reduced to a set of rules, which, for simplification, are eliminated via pruning.

library(Cubist)  
set.seed(100)  
cubistTune <- train(x = ph\_pred\_train\_num, y = ph\_resp\_train,  
 method = "cubist", trControl = mdl\_ctrl)  
cubistPred <- predict(cubistTune, newdata = ph\_pred\_test\_num)  
cubistPerf <- postResample(pred = cubistPred, obs = ph\_resp\_test)

# Model Selection & Prediction

## Model Performance Comparison

The resampled RMSE & test set prediction RMSE is shown below for each of the models created:

# function to extract & present model performance  
model\_perf <- function (model\_set, metric = "RMSE") {  
 mdl\_names <- character()  
 Resampled <- numeric()  
 Test <- numeric()  
 for (mdl in model\_set) {  
 mdl\_names <- c(mdl\_names, mdl)  
 Resampled <- c(Resampled, min(get(paste0(mdl, "Tune"))$results[[metric]]))  
 Test <- c(Test, get(paste0(mdl, "Perf"))[metric])  
 }  
 pander(data.frame(Resampled, Test, row.names = mdl\_names),  
 digits = 4, caption = paste(metric, "performance for selected models"))  
}  
# present performance for 15 models created  
model\_perf(c("lm", "glm", "pcr", "step",  
 "mars", "nnet", "svmRadial", "svmLinear", "knn",  
 "rpart", "rule", "bag", "rf", "boost", "cubist"))

RMSE performance for selected models

|  |  |  |
| --- | --- | --- |
|  | Resampled | Test |
| **lm** | 0.1368 | 0.1454 |
| **glm** | Inf | 6.403 |
| **pcr** | Inf | 0.1489 |
| **step** | Inf | 0.1461 |
| **mars** | Inf | 0.1316 |
| **nnet** | 0.1168 | 0.1196 |
| **svmRadial** | 0.1667 | 0.1682 |
| **svmLinear** | 0.1382 | 0.1493 |
| **knn** | 0.1285 | 0.1338 |
| **rpart** | 0.1328 | 0.1416 |
| **rule** | Inf | 0.1288 |
| **bag** | 0.116 | 0.1267 |
| **rf** | 0.1012 | 0.1108 |
| **boost** | 0.1154 | 0.1252 |
| **cubist** | 0.09571 | 0.1047 |

Noting that the models trained using functions other than caret::train do not return a resampled RMSE, the best model based on both resampled & test RMSE is the **Cubist model**. To avoid any bias from looking only at a single metric, the r-squared is also investigated:

model\_perf(c("lm", "glm", "pcr", "step",  
 "mars", "nnet", "svmRadial", "svmLinear", "knn",  
 "rpart", "rule", "bag", "rf", "boost", "cubist"),  
 metric = "Rsquared")

Rsquared performance for selected models

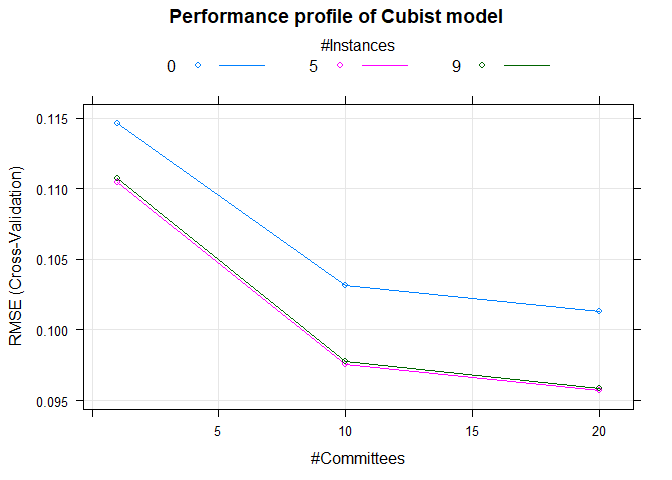
|  |  |  |
| --- | --- | --- |
|  | Resampled | Test |
| **lm** | 0.3469 | 0.2868 |
| **glm** | Inf | 0.2868 |
| **pcr** | Inf | 0.2534 |
| **step** | Inf | 0.2803 |
| **mars** | Inf | 0.4144 |
| **nnet** | 0.1135 | 0.5144 |
| **svmRadial** | 0.111 | 0.114 |
| **svmLinear** | 0.3489 | 0.2711 |
| **knn** | 0.4197 | 0.4133 |
| **rpart** | 0.2334 | 0.3233 |
| **rule** | Inf | 0.4419 |
| **bag** | 0.5407 | 0.4583 |
| **rf** | 0.608 | 0.5898 |
| **boost** | 0.3736 | 0.4679 |
| **cubist** | 0.5675 | 0.6313 |

The Cubist model also performs best as judged by the r-squared metric on the test set. While the random forest model performs slightly better on resampled r-squared, it is not by a significant margin. The Cubist model will be used for prediction.

## Model Investigation

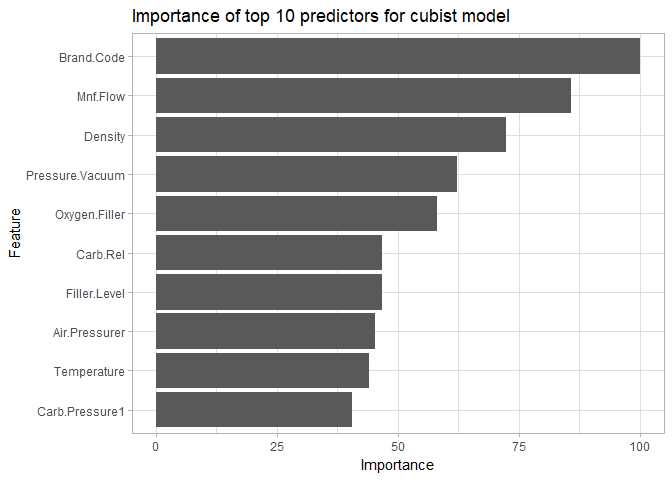
The performance profile of the selected Cubist across different tuning parameters is shown below:

plot(cubistTune, main = "Performance profile of Cubist model")



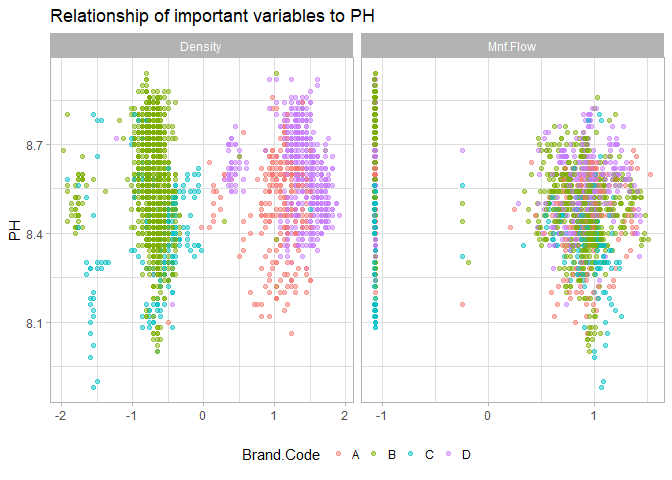
For this model, the optimal RMSE of 0.0957 is obtained for the model with 5 neighbors & 20 committees. The 10 most important variables used in this final model are presented below:

cubistImp <- varImp(cubistTune)  
ggplot(cubistImp, top = 10) + ggtitle("Importance of top 10 predictors for cubist model")



The model is dominated by Brand.Code, with Mnf.Flow, Density, Pressure.Vacuum, and Oxygen.Filler registering as the next-most-important variables. The relationships between Brand.Code and the two most important numeric variables, Mnf.Flow and Density, with PH are presented below:

data.frame(ph\_pred, ph\_resp) %>%   
 select(PH, Brand.Code, Mnf.Flow, Density) %>%   
 gather(Variable, Value, -c(PH, Brand.Code)) %>%   
 ggplot(aes(x = Value, y = PH, col = Brand.Code)) +  
 geom\_point(alpha = 0.5) +  
 facet\_wrap(~ Variable, scales = "free\_x") +  
 theme(legend.position = "bottom") +  
 labs(title = "Relationship of important variables to PH", x = NULL)



There is not an immediately apparent relationship between Mnf.flow and PH nor Brand.Code, but the relationship between Density and Brand.Code shows fairly clear clustering behavior. Within each cluster, there appears to be a weakly positive relationship with PH.

## Prediction of Additional Data

The Cubist model is finally used to predict additional data not part of the training or test set.

# read in data  
new\_data <- read.csv("StudentEvaluation.csv", na.strings = c("", " "), stringsAsFactors = FALSE)

The new data is transformed in the same way as the original data:

# remove ph to get only predictors  
new\_pred <- new\_data %>% select(-PH)  
# remove Hyd.Pressure1, Hyd.Pressure2, & Hyd.Pressure3  
new\_pred <- new\_pred %>% select(-(Hyd.Pressure1:Hyd.Pressure3))  
# perform transformations used for original data  
new\_pred <- predict(ph\_preproc, new\_pred)  
# convert Brand.Code to numeric  
new\_pred <- mutate(new\_pred, Brand.Code = as.numeric(as.factor(Brand.Code)))

Finally, the response is predicted and output:

# predict  
new\_resp <- predict(cubistTune, new\_pred)  
# write predictors & response to csv  
data.frame(new\_data, PH = new\_resp) %>%   
 write\_csv("Team3\_Project2\_Fall624.csv")

The predicted data is included alongside this submission as *Team3\_Project2\_Fall624.csv.*