Data624 Project 2

Group 4

2021-07-17

Table of Contents

Group Members

* Subhalaxmi Rout
* Kenan Sooklall
* Devin Teran
* Christian Thieme
* Leo Yi

## Introduction

We have been given a dataset from a beverage manufacturing company that consists of 2,571 rows of data and 33 columns. The dataset contains information on different beverages and their chemical composition. The goal of this analysis is to use the 32 predictive features to predict the *Potential for hydrogen* (pH), which is a measure of the acidity/alkalinity of the beverage. pH is the key KPI in this analysis.

We’ll begin by reading in the dataset and looking at each column’s data type:

data\_raw <- readr::read\_csv('https://raw.githubusercontent.com/christianthieme/Predictive-Analytics/main/Project2/data/StudentData%20-%20TO%20MODEL.csv')  
data\_raw$obs\_type <- "train"  
eval\_raw <- readr::read\_csv('https://raw.githubusercontent.com/christianthieme/Predictive-Analytics/main/Project2/data/StudentEvaluation-%20TO%20PREDICT.csv')  
eval\_raw$obs\_type <- "test"  
combined <- data\_raw %>%  
 rbind(eval\_raw)  
# convert column names to all lowercase  
names(combined) <- lapply(names(combined), tolower)  
# convert column name spaces to underscore  
names(combined) <- str\_replace\_all(names(combined), ' ', '\_')  
df <- combined %>%  
 filter(obs\_type == 'train') %>%  
 select(-obs\_type)  
eval <- combined %>%  
 filter(obs\_type == 'test') %>%  
 select(-obs\_type)  
glimpse(combined)

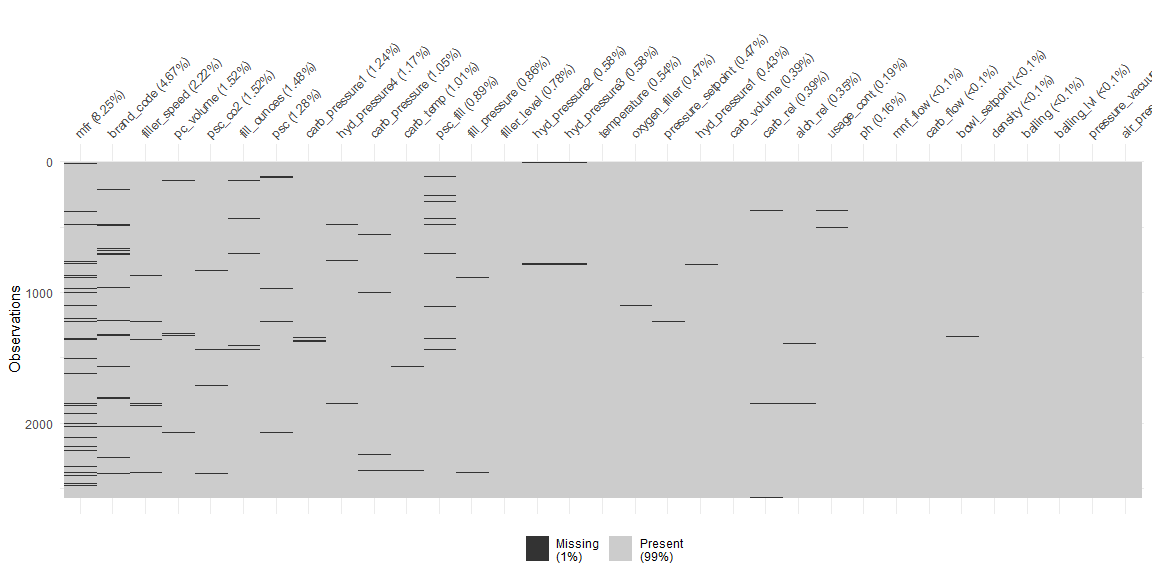
## Rows: 2,838  
## Columns: 34  
## $ brand\_code <chr> "B", "A", "B", "A", "A", "A", "A", "B", "B", "B",...  
## $ carb\_volume <dbl> 5.340000, 5.426667, 5.286667, 5.440000, 5.486667,...  
## $ fill\_ounces <dbl> 23.96667, 24.00667, 24.06000, 24.00667, 24.31333,...  
## $ pc\_volume <dbl> 0.2633333, 0.2386667, 0.2633333, 0.2933333, 0.111...  
## $ carb\_pressure <dbl> 68.2, 68.4, 70.8, 63.0, 67.2, 66.6, 64.2, 67.6, 6...  
## $ carb\_temp <dbl> 141.2, 139.6, 144.8, 132.6, 136.8, 138.4, 136.8, ...  
## $ psc <dbl> 0.104, 0.124, 0.090, NA, 0.026, 0.090, 0.128, 0.1...  
## $ 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, -...  
## $ carb\_pressure1 <dbl> 118.8, 121.6, 120.2, 115.2, 118.4, 119.6, 122.2, ...  
## $ fill\_pressure <dbl> 46.0, 46.0, 46.0, 46.4, 45.8, 45.6, 51.8, 46.8, 4...  
## $ hyd\_pressure1 <dbl> 0, 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, ...  
## $ filler\_speed <dbl> 4002, 3986, 4020, 4012, 4010, 4014, NA, 1004, 401...  
## $ temperature <dbl> 66.0, 67.6, 67.0, 65.6, 65.6, 66.2, 65.8, 65.2, 6...  
## $ usage\_cont <dbl> 16.18, 19.90, 17.76, 17.42, 17.68, 23.82, 20.74, ...  
## $ 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, ...  
## $ pressure\_vacuum <dbl> -4.0, -4.0, -3.8, -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, ...  
## $ bowl\_setpoint <dbl> 120, 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, 4...  
## $ air\_pressurer <dbl> 142.6, 143.0, 142.0, 146.2, 146.2, 146.6, 146.2, ...  
## $ 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...  
## $ obs\_type <chr> "train", "train", "train", "train", "train", "tra...

We see that all columns, with the exception of brand, are doubles and continuous. Excluding the response variable, this means that we have 1 categorical variable and 31 continuous variables to work with.

## Exploratory Data Analysis

In the output above, we can see that there are missing values (NAs). Let’s see how pervasive this issue is within our dataset:

df %>%  
 visdat::vis\_miss(sort\_miss = TRUE)

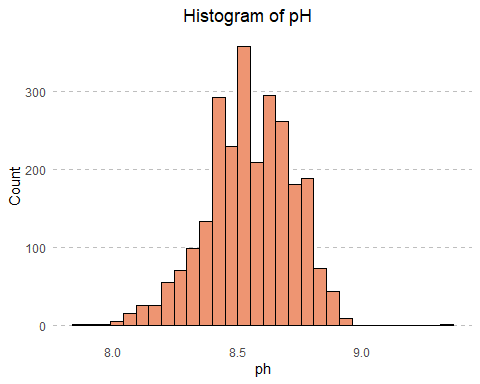


In total, only about 1% of our data is missing. We can see that most of the columns are only missing a negligible amount of data. mfr and brand code have the largest amount of missing values and are missing 8.25% and 4.67% of their data, respectively. Additionally, there does not appear to be a pattern in which values are missing. Now that we understand that our missing values are not a pervasive issue, we’ll continue with our analysis.

#### Distribution of Response Variable: pH

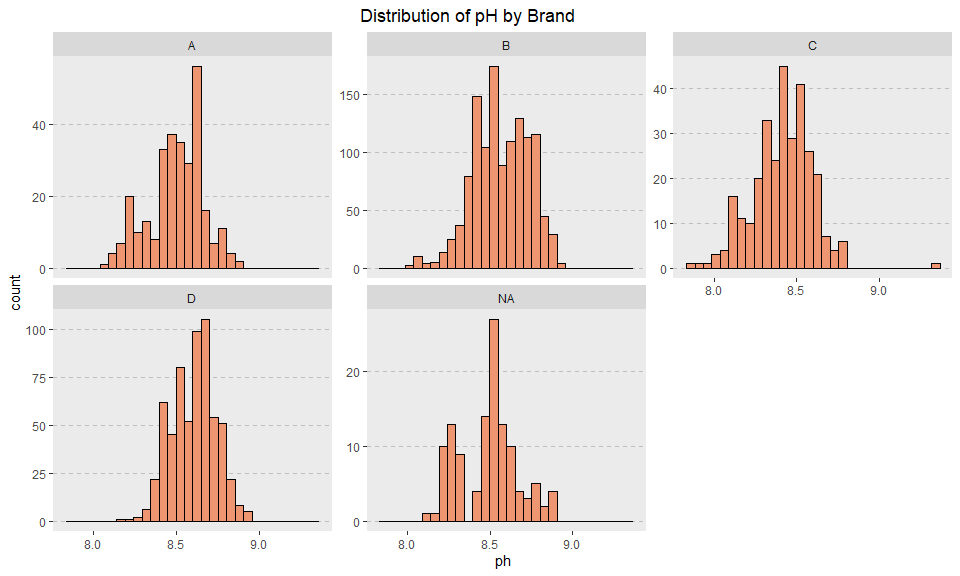
Let’s get an understanding of the distribution of our response variable:

df %>%  
 select(ph) %>%  
 ggplot() +   
 aes(x = ph) +   
 geom\_histogram(fill = "lightsalmon2", color = "black") +   
 labs(title = "Histogram of pH", y = "Count") +   
 theme\_minimal() +   
 theme(  
 plot.title = element\_text(hjust = 0.45),  
 # plot.margin = ggplot2::margin(10, 20, 10, 10),  
 panel.grid.major.y = element\_line(color = "grey", linetype = "dashed"),  
 panel.grid.major.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.minor.x = element\_blank()  
 )



The distribution of pH is left-skewed and multi-modal. Generally speaking, when we see a multi-modal distribution, often times that is an indication that there are sub-populations within the distribution. We know from looking at our dataset that there is a brand code with values A, B, C, and D. Let’s break up the above distribution into 4 distributions based on these values:

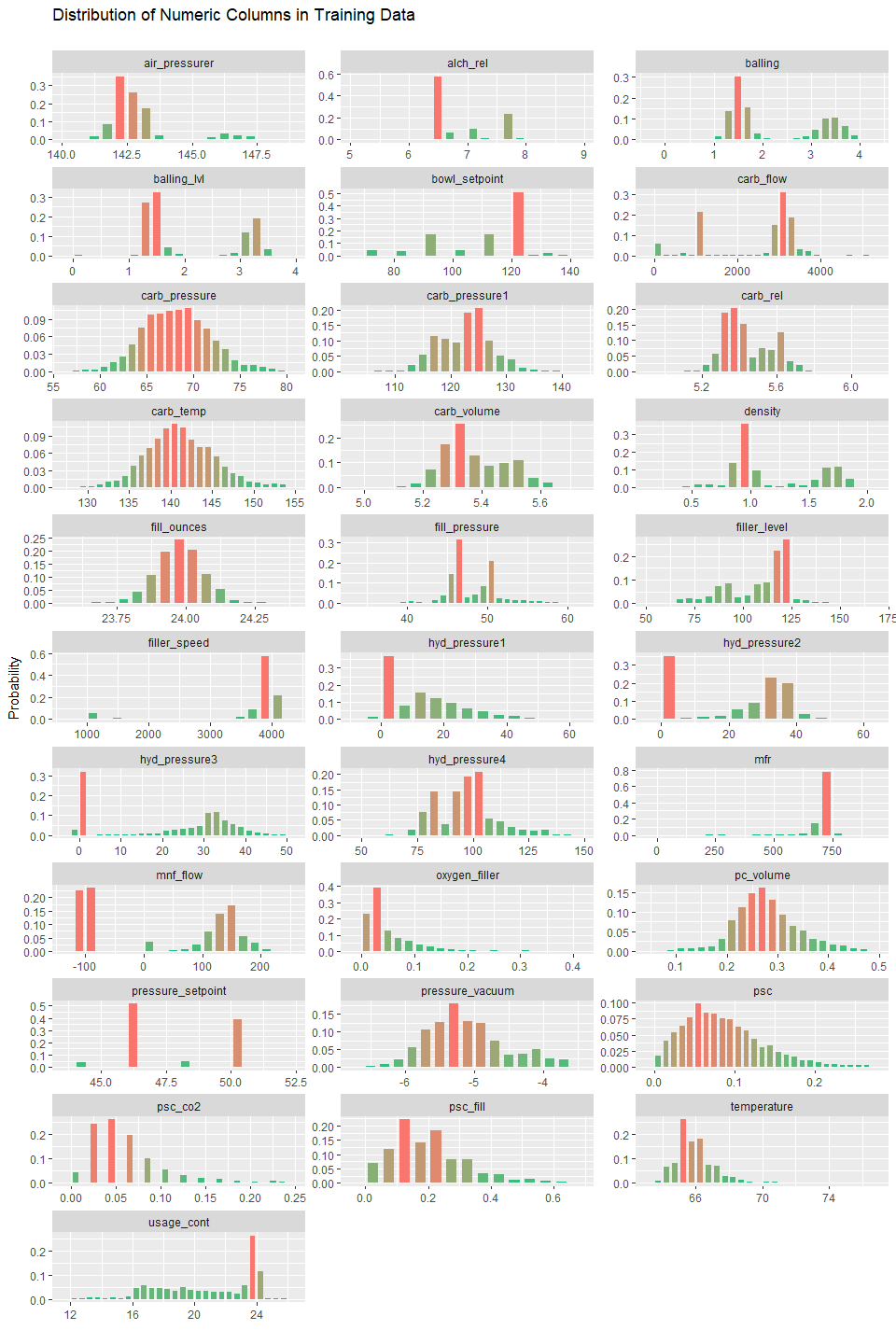
df %>%  
 ggplot() +   
 aes(x = ph) +   
 geom\_histogram(fill = "lightsalmon2", color = "black") +  
 labs(title = "Distribution of pH by Brand") +   
 facet\_wrap(~brand\_code, scales = 'free\_y') +   
 theme(  
 plot.title = element\_text(hjust = 0.45),  
 # plot.margin = ggplot2::margin(10, 20, 10, 10),  
 panel.grid.major.y = element\_line(color = "grey", linetype = "dashed"),  
 panel.grid.major.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.minor.x = element\_blank()  
 )



Breaking down to this further grain does not seem to be much more helpful in eliminating some of the bi/multi-modal nature of these distributions. There may be even more granular sub-populations within this data that we are not seeing.

Now we’ll turn our attention to the numeric features within our dataset:

df %>%  
 select(-ph) %>%  
 inspectdf::inspect\_num() %>%  
 show\_plot() +  
 labs(title = 'Distribution of Numeric Columns in Training Data')



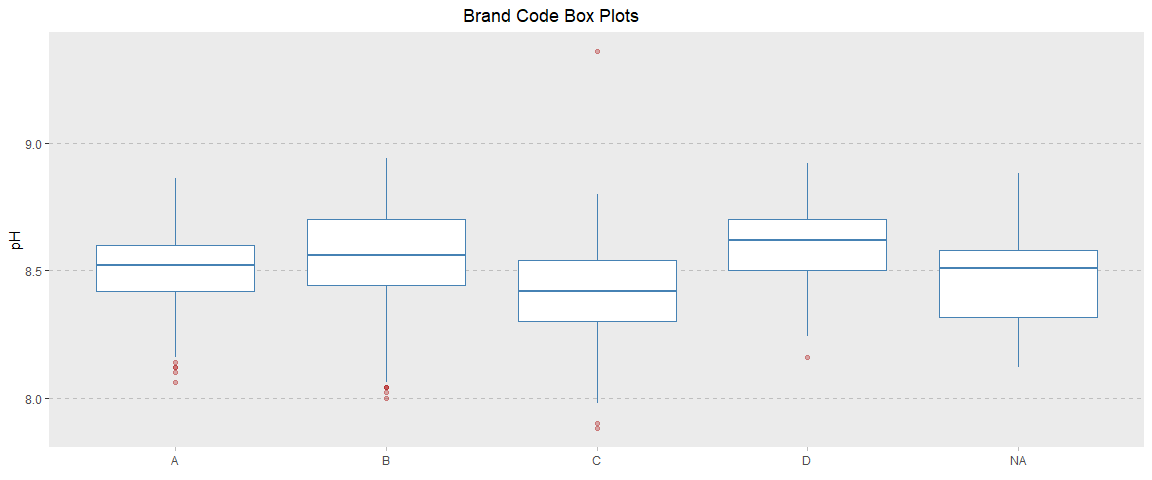
We note the following about these distributions:

* air\_pressurer - there appears to be either two distributions here, or a single distribution and a pocket of outliers
* balling, balling\_lvl, density,fill\_ \_pressure, hyd\_pressure1, hyd\_pressure2, hyd\_pressure3, hyd\_pressure4, mnf\_flow, pressure\_setpoint- there appears to be two distributions here. This could potentially be connected to the type of brand\_code or something else not as easily distinguishable.
* bowl\_setpoint - half of all the values are around 120
* carb\_flow - most values fall between 3,000 and 4,000 with a large pocket of values at 1,000 as well
* filler\_speed, mfr, oxygen\_filler - either appears to have two distributions or a few significant outliers
* general note: it appears that many of these distributions are skewed in one way or another. We note that a transformation may be helpful when generating predictions.

#### Explanatory Variable Relationships with the Response Variable

Now that we’ve looked at our response variable, let’s look at our explanatory variables. We’ll begin first by looking at brand code, which is our only categorical variable:

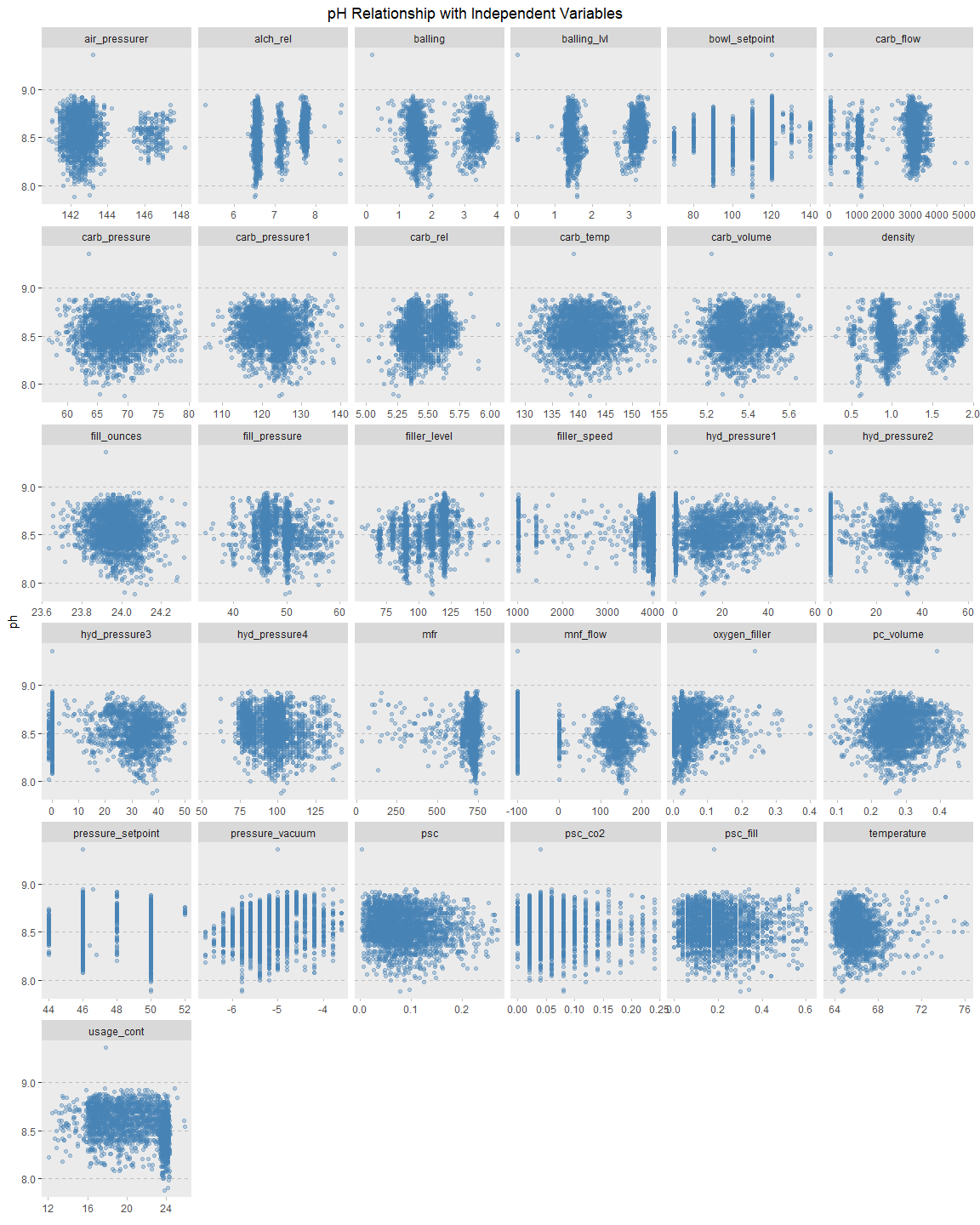
df %>%  
 dplyr::select(brand\_code, ph) %>%  
 ggplot() +   
 aes(x = brand\_code, y = ph) +   
 geom\_boxplot(color = 'steelblue', outlier.color = 'firebrick', outlier.alpha = 0.35)+  
 labs(title = 'Brand Code Box Plots',  
 x = element\_blank(),  
 y = 'pH') +   
 # facet\_grid(~brand\_code) +  
# theme\_minimal() +   
 theme(  
 plot.title = element\_text(hjust = 0.45),  
 panel.grid.major.y = element\_line(color = "grey", linetype = "dashed"),  
 panel.grid.major.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.minor.x = element\_blank(),  
 # panel.background = element\_blank(),  
 axis.ticks.x = element\_line(color = "grey")  
 )



We can see from the above boxplots that brand\_code does have a meaningful relationship with pH. We can also see some significant outliers in C and possibly D that will need to be evaluated further. We’ll now turn our attention to the numeric features in our dataset.

**Numeric Features**

df %>%  
 select(-brand\_code) %>%   
 gather(variable, value, -ph) %>%  
 ggplot(aes(x = value, y = ph)) +  
 geom\_point(color = 'steelblue', alpha = 0.35) +  
 facet\_wrap(~variable, scales = "free\_x") +  
 labs(title = "pH Relationship with Independent Variables", x = element\_blank())+   
 theme(  
 plot.title = element\_text(hjust = 0.45),  
 panel.grid.major.y = element\_line(color = "grey", linetype = "dashed"),  
 panel.grid.major.x = element\_blank(),  
 panel.grid.minor.y = element\_blank(),  
 panel.grid.minor.x = element\_blank(),  
 axis.ticks.x = element\_line(color = "grey")  
 )



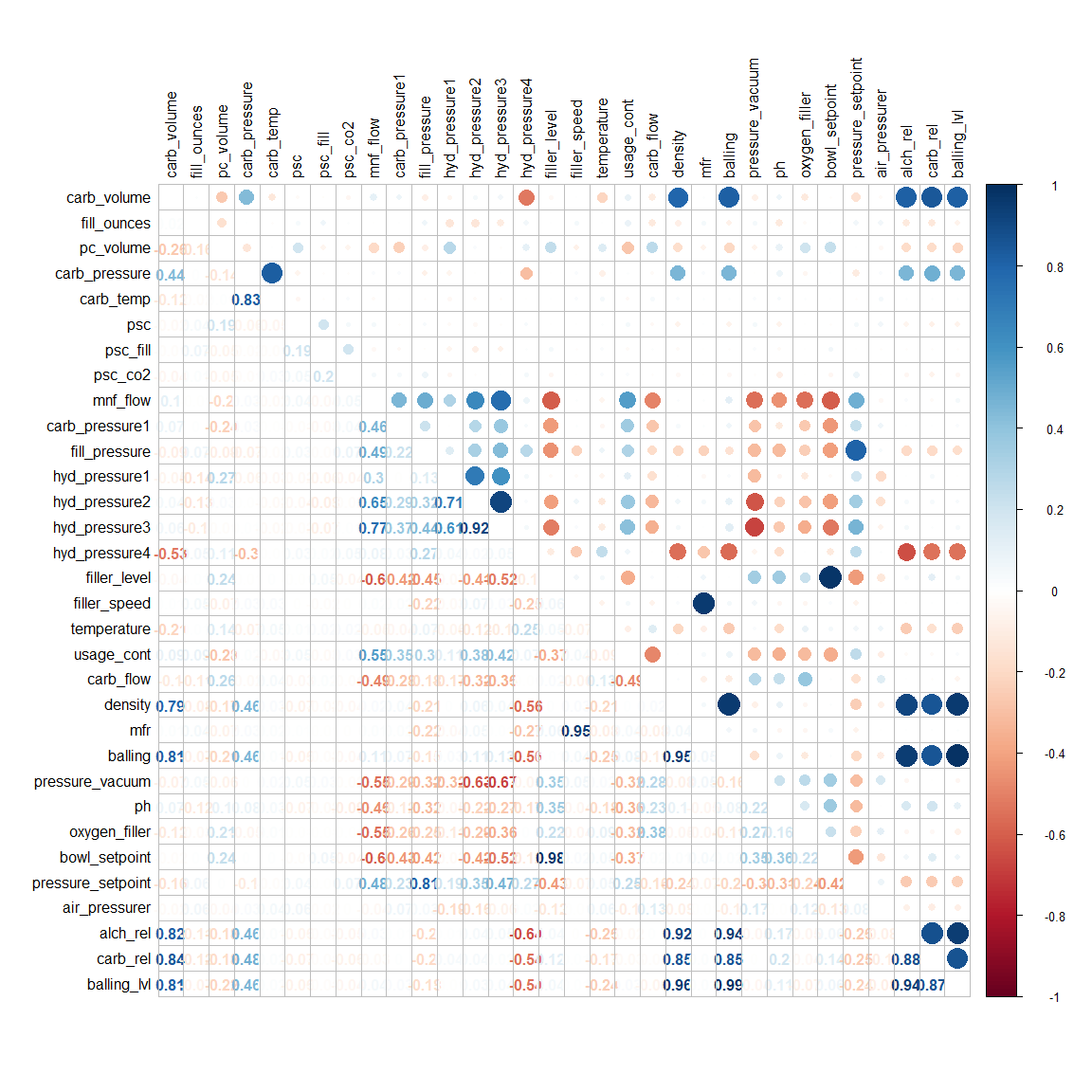
We note the following about the relationship between pH and these variables:

* air\_pressurer - it appears that there are two sub-groups here. In looking to see if this was due to brand\_code we found that these sub-groups exist even at individual group levels
* alch\_rel - most points appear to be gathered in 3 distinct areas, however there do appear to be 7 outliers
* balling, balling\_lvl - it appears that there are two sub-groups here. These sub-groups do potentially look to be associated with brand\_code
* bowl\_setpoint - appears to potentially be a categorical variable as the values don’t appear to be continuous. Also appear to potentially be some outliers
* carb\_flow - appear to be two or three groups of data points. Also note the presence of outliers
* density - appear to be two to three groups of points. We note the presence of an outlier
* filler\_speed - appear to mostly fall within the low or high range. Values in the middle are less frequent.
* pressure\_setpoint - appear to be discrete values with the exception of 4 outliers
* pressure\_vacuum - appear to be discrete values with a potentially positive linear relationship. We note the presence of an outlier
* psc\_co2 - appear to be discrete values. We note the presence of an outlier
* psc\_fill - there appear to be 5 bands that values can fall into with certain areas that do not have values. We may consider adding a categorical variable to capture this
* carb\_pressure, carb\_pressure1, carb\_rel, carb\_volume, fill\_ounces, fill\_pressure, filler\_level, hyd\_pressure1-4, mfr, oxygen\_filler, pc\_volume, psc, temperature, usage\_cont- no visible relationship. We do note the presence of outliers
* General note: It appears that many of these variables are on different scales. We’ll take care of this during our data prep phase. Additionally, there looks to be one outlier in most of these visuals. This outlier may need to be removed.

**Correlated Features**

For many models, correlation between features can be an issue. Let’s see what the correlation between our variables looks like:

numeric\_values <- df %>%   
 dplyr::select\_if(is.numeric)  
numeric\_values <- numeric\_values[complete.cases(numeric\_values),] %>%   
 data.frame()  
train\_cor <- cor(numeric\_values)  
corrplot::corrplot.mixed(train\_cor, tl.col = 'black', tl.pos = 'lt')



We see many of our features are *highly* correlated. There are several methods we could use to solve this, however, because we have many features, it may make sense to use principal component analysis, which will allow us to reduce the number of columns in our model and hopefully produce a simpler model. Additionally, many tree-based models are not as sensitive to collinearity like linear models, therefore, we’ll focus our efforts on non-linear models in the modeling phase.

**Summary EDA Notes**

* Feature distributions are skewed and may benefit from a transformation
* Missing data will most likely not be a significant issue
* There are several outliers in our features - we should think about using a modeling technique that is robust against outliers
* Many of our features are significantly correlated with each other. PCA or another method may be helpful in reducing collinearity
* There appear to be sub-populations even within brand\_codes. It may be helpful to do some feature engineering to tease this information from the data

## Data Processing

#### Outliers

In our analysis of the data, we noted several outliers. Most of those outliers come from one value with a pH of 9.36. We removed this outlier, but saw a marked reduction in accuracy and so made the decision to keep the value in the training set.

#### Feature Engineering

Many attempts were made at feature engineering, including combining features, rounding features, using other features as a percentage of other features, etc., however, any additional features seemed to decrease the performance of our model. It may be that each feature has fairly strong signal on its own, and our engineering may have been confounding that signal. As such, we have not included any additional features in our dataset.

#### Data Imputation

We’ll need to impute missing data for both numeric and categorical variables. We’ll use the knnImpute method for the numeric variables and a multinomial logistic regression model to impute the brand\_code variable.

We do not use the combined dataset to train our model or impute values under the assumption that this evaluation data wouldn’t be available to us when training our models. In practice, when we’re faced with a new dataset, it would be impossible to use that data in the process of training our models.

# remove rows without a response variable  
df3 <- df %>%  
 filter(!is.na(ph)) %>%  
 mutate(brand\_code = factor(brand\_code))  
# remove near zero variance variables  
isNZV <- nearZeroVar(df3)  
df3 <- df3[, -isNZV]  
# imputing numeric variables  
pp <- df3 %>%  
 as.data.frame() %>%  
 preProcess(method = c('knnImpute'))   
# apply imputation of numeric variables  
df3 <- predict(pp, df3)   
# remove rows without a brand code  
df3b <- df3 %>%  
 filter(!is.na(brand\_code))  
# predict class using multinomial logistic regression  
pc <- train(brand\_code ~ .,  
 data = df3b,  
 method = 'multinom',  
 trControl = trainControl(method = 'cv', number = 10),  
 trace = F)  
# create new field with predictions of brand code  
df3$brand2 <- predict(pc, df3)  
# fill in missing brand codes with imputed values  
df3$brand\_code <- ifelse(!is.na(df3$brand\_code), df3$brand\_code, df3$brand2)  
# remove brand prediction field  
df3$brand2 <- NULL

**Processing Note**

As part of our processing, we attempted many different transformations and adjustments, but saw no improvement in our RMSE or MAE, in fact, many of the changes negatively affected our accuracy, as such, we chose a simpler processing approach. We experimented with YeoJohnson transformations, centering, scaling, excluding highly correlated features, and principal component analysis. Ultimately, we determined that *only* using KNN to impute missing values in our numerical columns and a multinomial model to impute missing values in our categorical column produced the most accurate results.

#### Train Test Split

After imputing, we’ll use caret’s createDataPartition to determine a stratified 80/20 split. We will use this test dataset to evaluate all our models, however, if two or more models have close error and variance measures, we can continue with cross validation on additional splits. During our EDA we noted that brand\_code was a significant feature within our dataset, as such we chose to use a stratified sampling approach in an effort to make our training and testing sets more uniform.

set.seed(101)  
trainIndex <- createDataPartition(df3$ph,  
 p = 0.8,  
 list = F)  
train <- df3[trainIndex, ]  
test <- df3[-trainIndex, ]

## Modeling

Having split our data into a training and testing set, we’re now ready to fit and train our models. Due to the non-linear relationships of our features to the response variable as well the outliers we observed and other issues noted during our exploratory data analysis, we have determined that non-linear models would most likely perform best on this dataset when creating predictions. With this in mind, we’ll fit and train several different non-linear models in an effort to to see which one predicts with the lowest error and least variance on the test dataset. Once we determine which model is the most accurate, we’ll move forward with creating predictions on the provided scoring set. We have chosen the following models to train:

* Multivariate adaptive regression splines (MARS)
* Support-vector machines (SVM)
* K-Nearest neighbors (KNN)
* Random Forest regression (RF)
* Cubist regression
* Neural Network (NN)

As part of our training, we are using cross fold validation in an effort to select the optimal parameters for each model.

# MARS  
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)  
set.seed(101)  
mars <- train(ph ~ .,  
 data = train,  
 method = 'earth',  
 tuneGrid = marsGrid,  
 trControl = trainControl(method = 'cv')  
 )  
# SVM  
svm <- train(ph ~ .,  
 data = train,  
 method = 'svmRadial',  
 preProc = c('center', 'scale'),  
 tuneLength = 14,  
 trControl = trainControl(method = 'cv')  
 )  
# KNN  
set.seed(101)  
knn <- train(ph ~ .,  
 data = train,  
 method = 'knn',  
 preProc = c('center', 'scale'),  
 tuneGrid = data.frame(.k = 1:20),  
 trControl = trainControl(method = 'cv')  
 )  
# random forest  
rf <- randomForest(ph ~ .,  
 data = train,  
 ntrees = 1000)  
# cubist  
cubist <- train(ph ~ .,  
 data = train,  
 method = 'cubist')

* *Note: Due to the long length of time it takes to train our Neural Nets models, we have configured the below code chunk not to run, but have saved the results of the model run in the table below in the Compare Models section.*

# Avg Neural Nets  
set.seed(100)  
  
  
trainingData <- select(train, -ph)  
trainingData <- data.frame(trainingData)  
testData <- select(test, -ph)  
testData <- data.frame(testData)  
  
neural\_model <- expand.grid(.decay = c(0, 0.01, .1),  
 .size = c(1:10),  
 .bag = FALSE)  
  
# get the maximum number of hidden units  
maxSize <- max(neural\_model$.size)  
# there are M(p+1)+M+1 parameters in total  
numWts <- 1\*(maxSize \* (length(trainingData) + 1) + maxSize + 1)  
  
ctrl <- trainControl(method = "cv")  
  
avNNet\_model <- train(trainingData, train$ph,  
 method = "avNNet",  
 tuneGrid = neural\_model,  
 trControl = ctrl,  
 preProcess = c("center", "scale"),  
 linout = TRUE,  
 trace = FALSE,  
 maxit = 500,  
 MaxNWts = numWts  
 )  
  
plot(avNNet\_model)

avNNet\_predict <- predict(avNNet\_model, newdata = testData)  
avNNet\_results <- postResample(pred = avNNet\_predict, obs = test$ph)

#### Compare Models

Having trained each of our models, we’ll now turn our attention to evaluating the predictive power of each of the models. We’ll use postResample to measure the error between each of the model’s predictions and the holdout test data and display the accuracy metrics in the table below:

# predictions  
test$mars <- predict(mars, test)  
test$svm <- predict(svm, test)  
test$knn <- predict(knn, test)  
test$rf <- predict(rf, test)  
test$cubist <- predict(cubist, test)  
  
# measure of error and variance  
metrics <- data.frame(rbind(  
 postResample(pred = test$mars, obs = test$ph),  
 postResample(pred = test$svm, obs = test$ph),  
 postResample(pred = test$knn, obs = test$ph),  
 postResample(pred = test$rf, obs = test$ph),  
 postResample(pred = test$cubist, obs = test$ph)  
),  
 row.names = c('MARS', 'SVM', 'KNN', 'Random Forest', 'Cubist')  
)

|  |  |  |  |
| --- | --- | --- | --- |
| Model Type | RMSE |  | MAE |
| MARS | 0.6972039 | 0.4737391 | 0.5321751 |
| SVM | 0.6375411 | 0.5610928 | 0.4663123 |
| KNN | 0.6852058 | 0.4988402 | 0.515803 |
| RF | 0.5489842 | 0.6908398 | 0.4046192 |
| Cubist | 0.544259 | 0.6792009 | 0.3908086 |
| Neural Nets | 0.6246003 | 0.5758669 | 0.4811444 |

#### Cross Validation

The measure of error for the random forest and the cubist models were close, so we’ll do one pass of cross validation with a new train/test split in order to have additional confidence in the chosen model.

# create new train/test split with different seed and 75/25 split  
set.seed(2)  
trainIndex <- createDataPartition(df3$ph,  
 p = 0.75,  
 list = F)  
train2 <- df3[trainIndex, ]  
test2 <- df3[-trainIndex, ]  
  
# random forest  
rf2 <- randomForest(ph ~ .,  
 data = train2,  
 ntrees = 1000)  
  
# cubist  
cubist2 <- train(ph ~ .,  
 data = train2,  
 method = 'cubist')  
  
# predictions  
test2$rf <- predict(rf2, test2)  
test2$cubist <- predict(cubist2, test2)  
  
# measure of error and variance  
metrics2 <- data.frame(rbind(  
 postResample(pred = test$cubist, obs = test$ph),  
 postResample(pred = test2$rf, obs = test2$ph),  
 postResample(pred = test2$cubist, obs = test2$ph)  
),  
 row.names = c('Cubist v1', 'Random Forest v2', 'Cubist v2')  
)

|  |  |  |  |
| --- | --- | --- | --- |
| Model Type | RMSE |  | MAE |
| Cubist v1 | 0.544259 | 0.6792009 | 0.3908086 |
| RF v2 | 0.591846 | 0.6525806 | 0.4260858 |
| Cubist v2 | 0.5842717 | 0.6447772 | 0.4159849 |

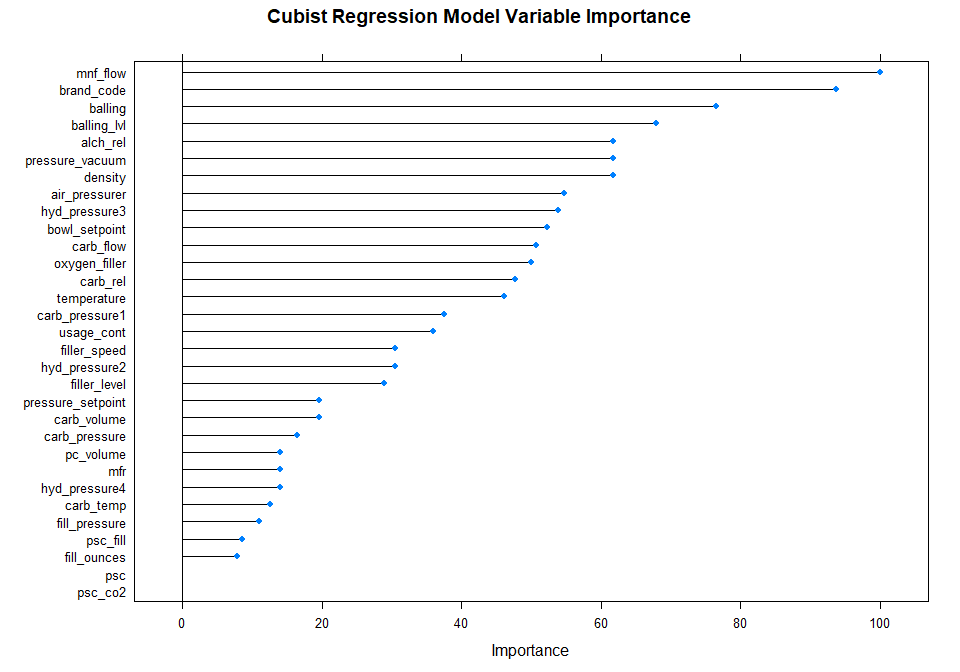
Comparing the 2nd fit of random forest vs cubist, the 2nd random forest fit shows a relatively greater error when predicting on the new test set, so the cubist is the better model again. When comparing the first and second cubist models, the first cubist outperforms the second.

Based on the model results above, we have chosen the **Cubist regression** model as our final model. The cubist model has the lowest RMSE and MAE.The values should only be used to compare models of the same type.

#### Variable Importance

Having selected our best preforming model, we can now use caret’s varImp method to look at what features within our dataset are most important when fitting the model.

plot(caret::varImp(cubist), main = "Cubist Regression Model Variable Importance")

 It appears that mnf\_flow, brand code, balling and balling\_lvl are key to our model’s performance. We saw in our correlation plot that mnf\_flow had the strongest relationship with pH. Additionally, we saw in our box plots that brand code had a significant relationship with pH as well.

balling, balling\_lvl, and density are all within the top 7 most important variables in this model, with significant correlations. We attempted to model after excluding the collinear variables, but that resulted in a less accurate model. It’s possible that the variable importance of these three variables might be misrepresented due to correlation.

## Predictions on the Test Set

We’ll use the Cubist regression model to predict on our test set. Before making predictions, we’ll need to perform the same data processing steps as we have done with our training dataset.

#### Imputing Evaluation Data

We’ll need to impute missing data on the final evaluation data. To do this, we’ll train another multinomial logistic regression model for brand code, excluding pH. Finally, we’ll use knnImpute again for the numeric variables. We’ll be using the entire combined dataset to impute these values under the assumption that we have trained our models on existing data and can leverage that to help us get a more accurate insight into the true population distributions.

# get all data excluding ph  
eval2 <- combined %>%  
 select(-ph)  
# convert brand code to factor  
eval2$brand\_code <- as.factor(eval2$brand\_code)  
# pre process evaluation data  
ppe <- eval2 %>%  
 as.data.frame() %>%  
 preProcess(method = 'knnImpute')  
# apply imputation of numeric variables  
eval2 <- predict(ppe, eval2)  
# remove rows without a brand code  
eval2b <- eval2 %>%  
 filter(!is.na(brand\_code))  
# predict class using multinomial logistic regression, excluding pH  
pc2 <- train(brand\_code ~ .,  
 data = eval2b,  
 method = 'multinom',  
 trControl = trainControl(method = 'cv', number = 10),  
 trace = F)  
# create new field with predictions of brand code  
eval2$brand2 <- predict(pc2, eval2)  
# fill in missing brand codes with imputed values  
eval2$brand\_code <- ifelse(!is.na(eval2$brand\_code), eval2$brand\_code, eval2$brand2)  
# remove brand prediction field  
eval2$brand2 <- NULL  
# isolate evaluation data only  
eval2 <- eval2 %>%  
 filter(obs\_type == 'test') %>%  
 select(-obs\_type)

Having imputed any missing values in our test set, we are now ready to make our final predictions.

#### Final Predictions

We make our final predictions by calling the predict method and passing in the eval2 dataframe which is our test set with all the necessary imputations for missing values. Once the predictions have been made, we’ll push them to a CSV file where they can be evaluated.

final\_predictions <- predict(cubist, eval2)  
head(final\_predictions)

## [1] -0.55650777 0.10106064 0.49030539 0.60111630 0.01196357 0.67511576

#write.csv(final\_predictions, "C:/Users/chris/OneDrive/Master Of Data Science - CUNY/Summer 2021/Predictive\_Analytics/Predictive-Analytics/Project2/final\_predictions.csv")

## Conclusion and Next Steps

Throughout this project, we imported the data, performed extensive exploratory data analysis, processed our data, and finally trained a model and made predictions. Throughout this process, we determined that the Cubist Regression model was the most accurate model, and used it to generate our final predictions. We did an extensive amount of experimenting with tranformations, imputation methods, and feature engineering, but at the end of the day, the simplist dataset seemed to perform the best.

With the unique nature of this dataset, we believe strong domain knowledge would be necessary to take this model to the next level. Often with these detailed datasets with many columns, it is hard to create new meaningful features without strong domain knowledge in the area. We believe feature engineering would be the place to focus and that additional modeling gains could be made by creating additional features from this rich dataset.