# Project 2 in R

## DATA 624-01 Group 3

Z. Palmore, K. Popkin, K. Potter, C. Nan, J. Ramalingam

7/8/2021

## Table of Contents

## Summary

Our objective was to predict the acidity or alkalinity of samples as closely as possible to actual values based on data provided from a food and beverage manufacturing company. Each sample must fit within a critical range. With 7 being neutral and pH set on a logarithm scale from 1 -10, each sample leaned towards higher alkalinity (basicness) with an average 'PH' of about 8.5. There were roughly 2571 samples and 33 potential predictors with measurements of those samples. We found that a nonparametric random forest model performed best based on the mean absolute error (MAE), root mean squared error (RMSE) and coefficient of determination (Rsquared). We also discovered that the samples were already generally bounded within a concise range of 'PH' values having 99.84% of samples between 8.0 and 9.0. We were not given a specific critical range, but this one appears to occur naturally. There are also regular variations in the target variable 'PH,' resembling an eb and flow during the manufacturing process. Measurements of 'PH,' including those missing, were randomly distributed and data required pre-processing for analysis.

## Data Exploration

Our data consists of 33 variables from a food and beverage manufacturing company. Our goal is to predict the hydronium ion concentration, or pH, which is a measure of acidity or alkalinity. This 'PH' measure, as shown in the data, is a Key Performance Indicator (KPI) and must conform to a critical range.

### Initial Observations

The data is loaded, and all packages used throughout the report are provided. We view a random sample of the data to inspect initial observations. It shows the randomly selected data as 1 – 5 denoting the first, second, third, fourth, and fifth observation for each variable.

```
library(utils)
library(psych)
library(pls)
library(tidyverse)
library(corrplot)
library(elasticnet)
library(kernlab)
library(plotrix)
library(ggcorrplot)
library(ggpubr)
library(party)
library(MASS)
library(mice)
library(mboost)
library(VIM)
library(rpart)
library(caret)
library(zoo)
```

```
library(rpart)
library(rpart.plot)
library(naniar)
library(partykit)
library(flextable)
library(bestNormalize)
library(doParallel) # Used for computation
library(earth) # Package necessary for marsModel
registerDoParallel(cores=2)
theme_set(theme_minimal())
set.seed(004)
ph <- read.csv(
  "https://raw.githubusercontent.com/palmorezm/msds/main/624/Projects/Project
2/StudentData%20-%20TO%20MODEL.csv")
obs.sample <- as.data.frame(t(head(sample_n(ph[1:33], 5), 5)))
obs.sample <- as.data.frame(lapply(obs.sample[2:33,],
function(x) round(as.numeric(as.character(x)),1)))
obs.sample <- rbind(ph$ï..Brand.Code, obs.sample)
obs.sample <- obs.sample %>%
  mutate(Variable = colnames(ph)) %>%
  dplyr::select(Variable, X1, X2, X3, X4, X5)
flextable(obs.sample) %>%
  theme_vanilla() %>%
  set_table_properties(layout = "autofit")
```

| Variable | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ï..Brand.Code | B | A | B | A | A |
| Carb.Volume | 5 | 5.4 | 5.2 | 5.3 | 5.3 |
| Fill.Ounces | 24.2 | 24.1 | 24 | 24 | 24.1 |
| PC.Volume | 0.1 | 0.3 | 0.2 | 0.2 | 0.3 |
| Carb.Pressure | 61.2 | 68.4 | 63 | 75.4 | 66.2 |
| Carb.Temp | 139.2 | 141.8 | 135.4 | 152.2 | 138.8 |
| PSC | 0 | 0.1 | 0.2 | 0.1 | 0.1 |
| PSC.Fill |  | 0.2 | 0.4 | 0.3 | 0.1 |
| PSC.CO2 | 0.1 | 0 | 0.2 | 0 | 0.1 |
| Mnf.Flow | 0.2 | -100.2 | 0.2 | -100 | -100.2 |
| Carb.Pressure1 | 122.4 | 111.8 | 125.8 | 117.6 | 114.4 |
| Fill.Pressure |  | 46 | 43.8 | 46.6 | 46.2 |

| Variable | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Hyd.Pressure1 | 0 | 0 | 0.2 | 0 | 0 |
| Hyd.Pressure2 | 0.2 | 0 | 0.2 | 0 | 0 |
| Hyd.Pressure3 | -1.2 | 0 | -1.2 | 0 | 0 |
| Hyd.Pressure4 | 98 | 74 | 126 | 88 | 94 |
| Filler.Level | | 120 | 128.2 | 119.4 | 120 |
| Filler.Speed | 1010 | 4010 | 1010 | 4010 | 3978 |
| Temperature | 67 | 64.4 | 71.6 | 66 | 65.4 |
| Usage.cont | 24.4 | 16.8 | 23.8 | 19.2 | 16.7 |
| Carb.Flow | 3014 | 2980 | 40 | 3110 | 2984 |
| Density | 0.8 | 1.7 | 0.7 | 0.9 | 0.9 |
| MFR | 466.4 | 732.6 | | 730.6 | 726.8 |
| Balling | 1.3 | 3.4 | 0.8 | 1.4 | 1.3 |
| Pressure.Vacuum | -5 | -5 | -5.6 | -4.4 | -4.6 |
| PH | 8.5 | 8.8 | 8.5 | 8.6 | 8.8 |
| Oxygen.Filler | 0 | 0 | 0 | 0 | 0.1 |
| Bowl.Setpoint | 90 | 120 | 90 | 120 | 120 |
| Pressure.Setpoint | 48 | 46 | 50 | 46 | 46 |
| Air.Pressurer | 142.4 | 142.4 | 142.6 | 146.6 | 142.6 |
| Alch.Rel | 6.5 | 7.1 | 6.5 | 6.5 | 6.5 |
| Carb.Rel | 5.5 | 5.5 | 5.2 | 5.3 | 5.4 |
| Balling.Lvl | 1.6 | 3.3 | 1.4 | 1.4 | 1.5 |

From these initial observations, we notice that data is missing. These appear as empty spots on the table. Some data points are also zero or negative, but it is unclear if this is intentional. For example, variable 'Pressure.Vacuum' is negative as one might expect when holding pressure in a vacuum, but the variables 'Mnf.Flow,' 'PSC,' 'Hyd.Pressure1,' 'Hyd.Pressure2,' 'Hyd.Pressure3,' and others are less intuitive.

Our table also displays differences in the scale of the variables' raw values. This will need to be reviewed to ensure it does not harm the model's predictive capabilities. It also appears that due to the conversion of the spreadsheet format from excel to 'csv' when hosting the

data remotely, our first column variable name is legible but needs the "i.." symbol removed. None of the other 33 variables need to be renamed. We take a closer look at these variables with some descriptive statistics.

## Inferential Statistics

Our target variable, PH, appears to be on a scale between 1 and 10, as it should be, while the others vary widely between -100 (Mnf.Flow) and greater than 4000 (Filler.Speed). For this reason, we review each variable's statistics individually. We calculate their total observations (obs), percent missing (pm), mean, median (med), standard deviation (sd), minimum value (min), maximum value (max), skewness (skew), and standard error (se) and combine them into a table by variable.

```
ph.desc <- ph %>%
  describe() %>%
  mutate("pm" = round(((2571 - n)/2571)*100, 2),
         obs = n,
         med = median) %>%
  round(digits = 1) %>%
  mutate(var = colnames(ph)) %>%
  dplyr::select(var, obs, pm, mean, med,
                sd, min, max, skew, se)
flextable::flextable(ph.desc) %>%
  theme_vanilla() %>%
  set_table_properties(layout = "autofit")
```

| var | obs | pm | mean | med | sd | min | max | skew | se |
|---|---|---|---|---|---|---|---|---|---|
| ï..Brand.Code | 2,571 | 0.0 | 3.4 | 3.0 | 1.1 | 1.0 | 5.0 | 0.0 | 0.0 |
| Carb.Volume | 2,561 | 0.4 | 5.4 | 5.3 | 0.1 | 5.0 | 5.7 | 0.4 | 0.0 |
| Fill.Ounces | 2,533 | 1.5 | 24.0 | 24.0 | 0.1 | 23.6 | 24.3 | 0.0 | 0.0 |
| PC.Volume | 2,532 | 1.5 | 0.3 | 0.3 | 0.1 | 0.1 | 0.5 | 0.3 | 0.0 |
| Carb.Pressure | 2,544 | 1.0 | 68.2 | 68.2 | 3.5 | 57.0 | 79.4 | 0.2 | 0.1 |
| Carb.Temp | 2,545 | 1.0 | 141.1 | 140.8 | 4.0 | 128.6 | 154.0 | 0.2 | 0.1 |
| PSC | 2,538 | 1.3 | 0.1 | 0.1 | 0.0 | 0.0 | 0.3 | 0.8 | 0.0 |
| PSC.Fill | 2,548 | 0.9 | 0.2 | 0.2 | 0.1 | 0.0 | 0.6 | 0.9 | 0.0 |
| PSC.CO2 | 2,532 | 1.5 | 0.1 | 0.0 | 0.0 | 0.0 | 0.2 | 1.7 | 0.0 |
| Mnf.Flow | 2,569 | 0.1 | 24.6 | 65.2 | 119.5 | -100.2 | 229.4 | 0.0 | 2.4 |
| Carb.Pressure1 | 2,539 | 1.2 | 122.6 | 123.2 | 4.7 | 105.6 | 140.2 | 0.1 | 0.1 |
| Fill.Pressure | 2,549 | 0.9 | 47.9 | 46.4 | 3.2 | 34.6 | 60.4 | 0.5 | 0.1 |

| var | obs | pm | mean | med | sd | min | max | skew | se |
|---|---|---|---|---|---|---|---|---|---|
| Hyd.Pressure1 | 2,560 | 0.4 | 12.4 | 11.4 | 12.4 | -0.8 | 58.0 | 0.8 | 0.2 |
| Hyd.Pressure2 | 2,556 | 0.6 | 21.0 | 28.6 | 16.4 | 0.0 | 59.4 | -0.3 | 0.3 |
| Hyd.Pressure3 | 2,556 | 0.6 | 20.5 | 27.6 | 16.0 | -1.2 | 50.0 | -0.3 | 0.3 |
| Hyd.Pressure4 | 2,541 | 1.2 | 96.3 | 96.0 | 13.1 | 52.0 | 142.0 | 0.5 | 0.3 |
| Filler.Level | 2,551 | 0.8 | 109.3 | 118.4 | 15.7 | 55.8 | 161.2 | -0.8 | 0.3 |
| Filler.Speed | 2,514 | 2.2 | 3,687.2 | 3,982.0 | 770.8 | 998.0 | 4,030.0 | -2.9 | 15.4 |
| Temperature | 2,557 | 0.5 | 66.0 | 65.6 | 1.4 | 63.6 | 76.2 | 2.4 | 0.0 |
| Usage.cont | 2,566 | 0.2 | 21.0 | 21.8 | 3.0 | 12.1 | 25.9 | -0.5 | 0.1 |
| Carb.Flow | 2,569 | 0.1 | 2,468.4 | 3,028.0 | 1,073.7 | 26.0 | 5,104.0 | -1.0 | 21.2 |
| Density | 2,570 | 0.0 | 1.2 | 1.0 | 0.4 | 0.2 | 1.9 | 0.5 | 0.0 |
| MFR | 2,359 | 8.2 | 704.0 | 724.0 | 73.9 | 31.4 | 868.6 | -5.1 | 1.5 |
| Balling | 2,570 | 0.0 | 2.2 | 1.6 | 0.9 | -0.2 | 4.0 | 0.6 | 0.0 |
| Pressure.Vacuum | 2,571 | 0.0 | -5.2 | -5.4 | 0.6 | -6.6 | -3.6 | 0.5 | 0.0 |
| PH | 2,567 | 0.2 | 8.5 | 8.5 | 0.2 | 7.9 | 9.4 | -0.3 | 0.0 |
| Oxygen.Filler | 2,559 | 0.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.4 | 2.7 | 0.0 |
| Bowl.Setpoint | 2,569 | 0.1 | 109.3 | 120.0 | 15.3 | 70.0 | 140.0 | -1.0 | 0.3 |
| Pressure.Setpoint | 2,559 | 0.5 | 47.6 | 46.0 | 2.0 | 44.0 | 52.0 | 0.2 | 0.0 |
| Air.Pressurer | 2,571 | 0.0 | 142.8 | 142.6 | 1.2 | 140.8 | 148.2 | 2.3 | 0.0 |
| Alch.Rel | 2,562 | 0.3 | 6.9 | 6.6 | 0.5 | 5.3 | 8.6 | 0.9 | 0.0 |
| Carb.Rel | 2,561 | 0.4 | 5.4 | 5.4 | 0.1 | 5.0 | 6.1 | 0.5 | 0.0 |
| Balling.Lvl | 2,570 | 0.0 | 2.1 | 1.5 | 0.9 | 0.0 | 3.7 | 0.6 | 0.0 |

No variable is missing greater than 8.2% (MFR) of its cases with 25 of the 33 variables at or under 1% missing. Our target 'PH,' is only missing 0.2%. A simple imputation should fix this without issue, as long as the errors are randomly dispersed.

The standard deviations inform us that the spread of each variable is also wide in several cases like 'Filler.Speed' and 'Carb.Flow,' while extremely narrow or even 0 for variables 'Oxygen.Filler,' 'PSC,' PSC.CO2,' 'PSC.Fill,' 'Carb.Volume,' and several more. These statistics confirm that we will be working with data objects of completely different scales each centered around their own averages. Our target 'PH,' has very little deviation as well, but in

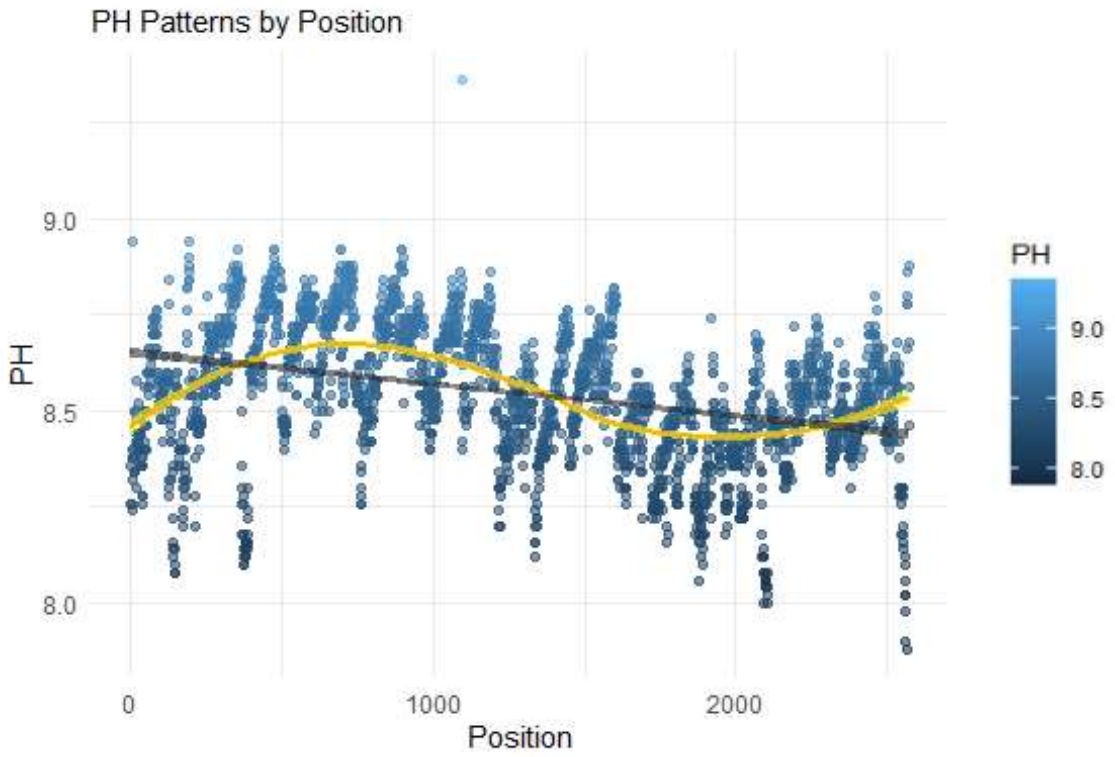
this case, it is to be expected since by nature it follows a logarithmic pattern, and we are only seeing values between 7.9 and 9.4.

There is some skewness among the variables and thereby potential outliers. This is most pronounced in the variables, 'MFR,' and 'Filler.Speed,' where the data has skewed left from its averages. Fortunately, all but two variables (Carb.Flow and Filler.Speed) have low standard errors indicating that nearly all variables will serve as good reference data sets for prediction due to low variation in spread from their means. To learn how good these variables may be at predicting, we need to see the relationship of each with our target 'PH,' interpret, and apply the results during modeling.

## Variable Relationships

To gain a better understanding of the relationships between our variables and the target 'PH,' we visualize the points as scatterplots with fitted regression lines. Before we make our first plot, we create an index value for 'PH' in the order it was read. This shows the pattern in individual measures of acidity or alkalinity as they were taken. We call this index value the 'Position' given its purpose of describing measurement position for each 'PH' value.

```
# Bounds estimates
cap99 <- (length(ph$PH) - 4) / length(ph$PH)
# Plot of Target PH
ph %>%
  dplyr::select(PH, Oxygen.Filler) %>%
  mutate(Index = 1:length(ph$PH)) %>%
  ggplot(aes(Index, PH, color = PH, alpha=.01)) +
  geom_point(aes()) +
  geom_smooth(method = "loess",
              color="goldenrod2",
              lty = "solid",
              fill = "yellow1") +
  geom_smooth(method = "lm",
              color="grey34",
              lty = "dotted",
              fill = "grey13") +
  labs(subtitle = "PH Patterns by Position",
       x = "Position", y = "PH") +
  theme(legend.position = "none")
```



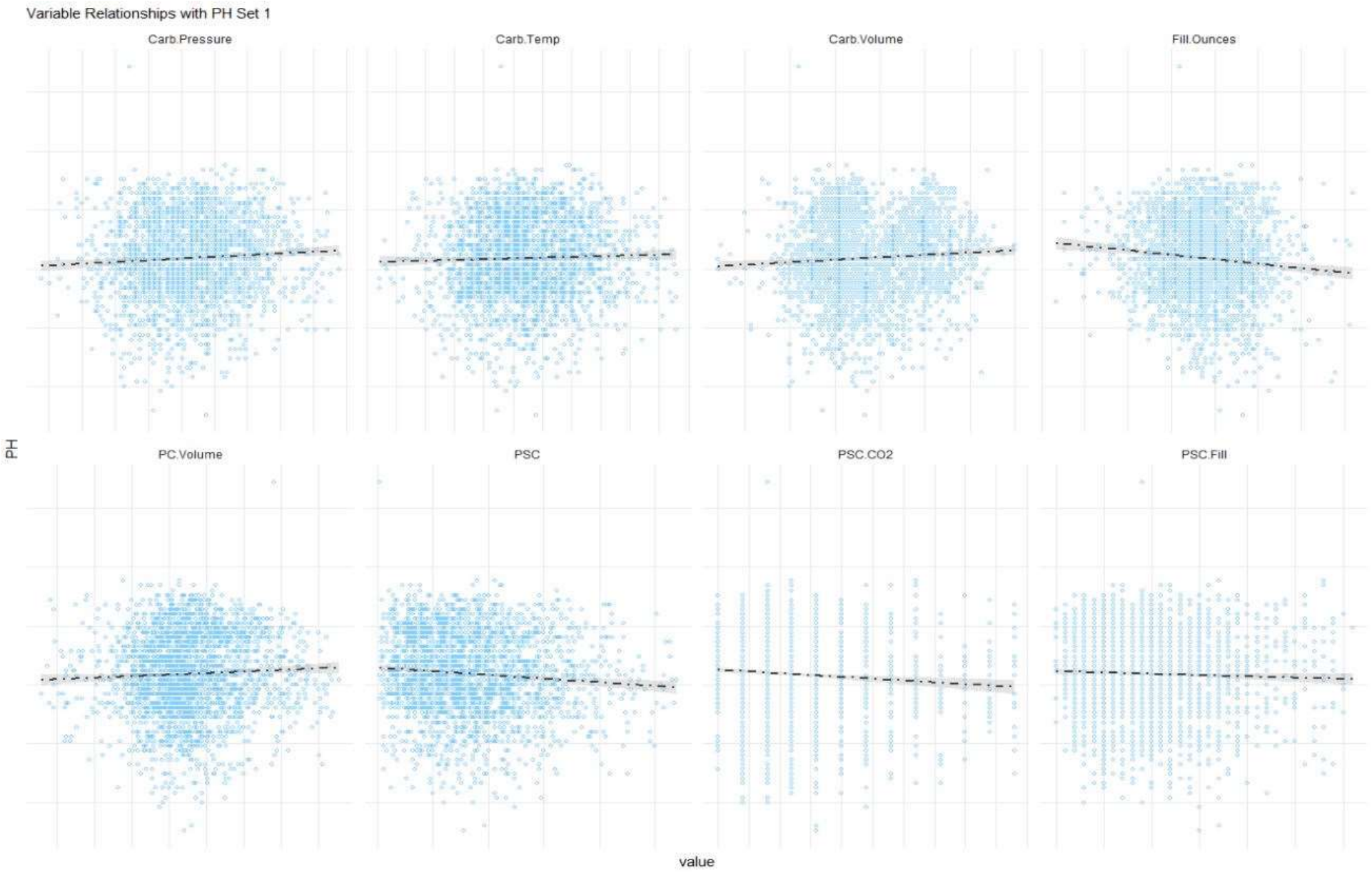

Interestingly, the pattern appears to fit an almost perfect sinusoidal curve, as demonstrated by the yellow line which is a closely fit locally weighted scatterplot smoothing (LOESS). For contrast, the black line attempts to fit a linear trend to the data. The shade of blue of each point indicates PH level and clustering. The lighter the point, the more alkaline and more solitary the measurement while darker points are more acidic and grouped together. A good example of the lightest point is the lone outlier plotted well above a pH of 9 just past position 1000.

A greater number of blue points (pH measurements) fall onto and around the yellow line than the black linear line and the pH shades do not show any big clusters or groups of pH values. From this we can be confident in two principles about this data. First, that the data measurements are random with no obvious pH clusters and secondly, that the data's collection process was not random, it was more sinusoidal.

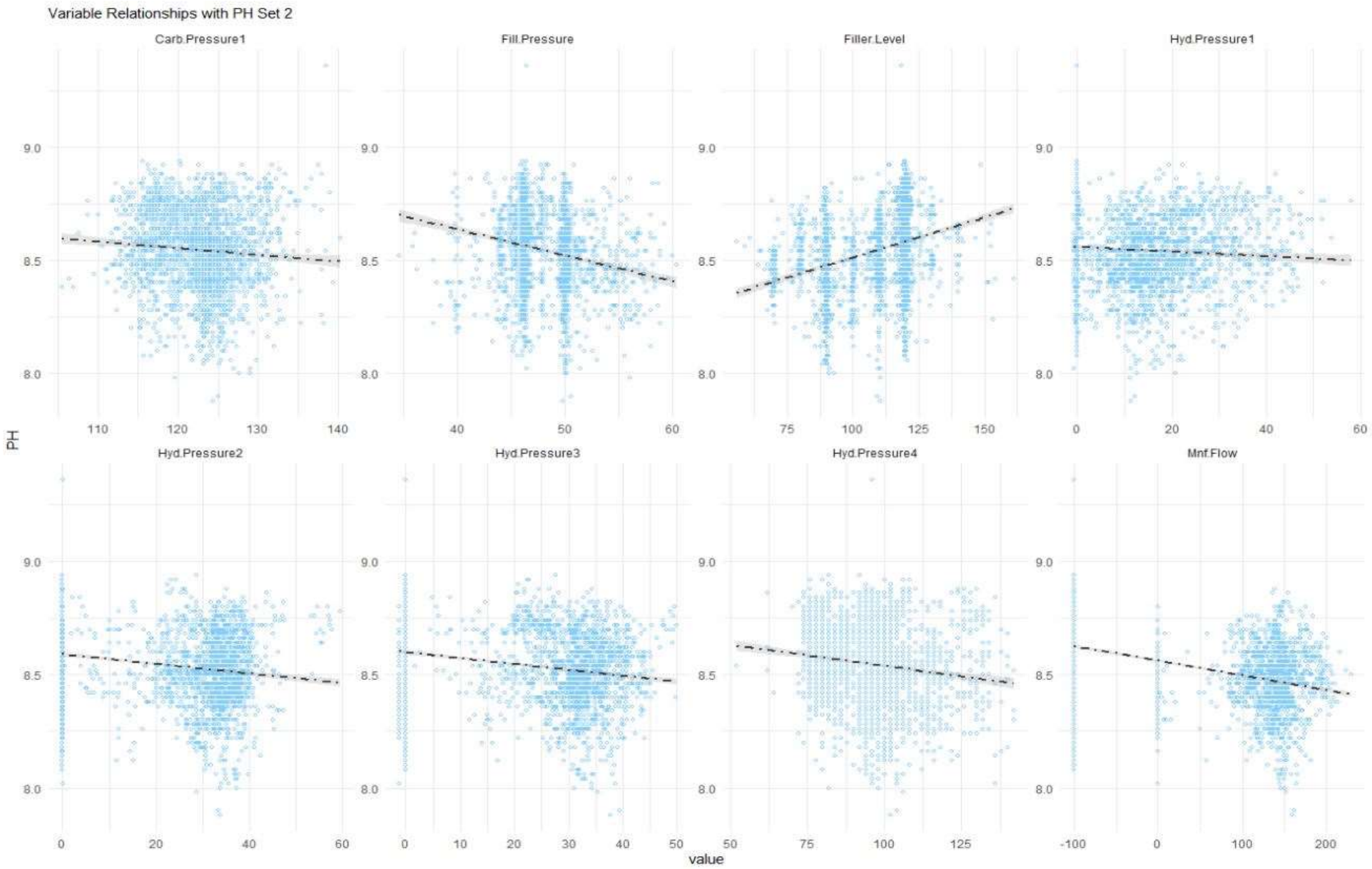
From this perspective, it is likely that there is a give and take of pH in each beverage. They are likely bounded by pH levels and the fluctuations in each beverage were accurately captured by the machine or device measuring pH. At this stage, we can also estimate the bounds as 99.84% of the measurements fall between 8.0 and 9.0 while over 75% are between 8.25 and 8.75. This should not have a tremendous effect on all models, but it would certainly affect any business decisions related to our predictions.

We continue to visualize the relationship of our target 'PH,' with the variables. For ease of viewing, we plot these variables in 4 sets each given a linear black line of best fit using 'PH,' as the response.

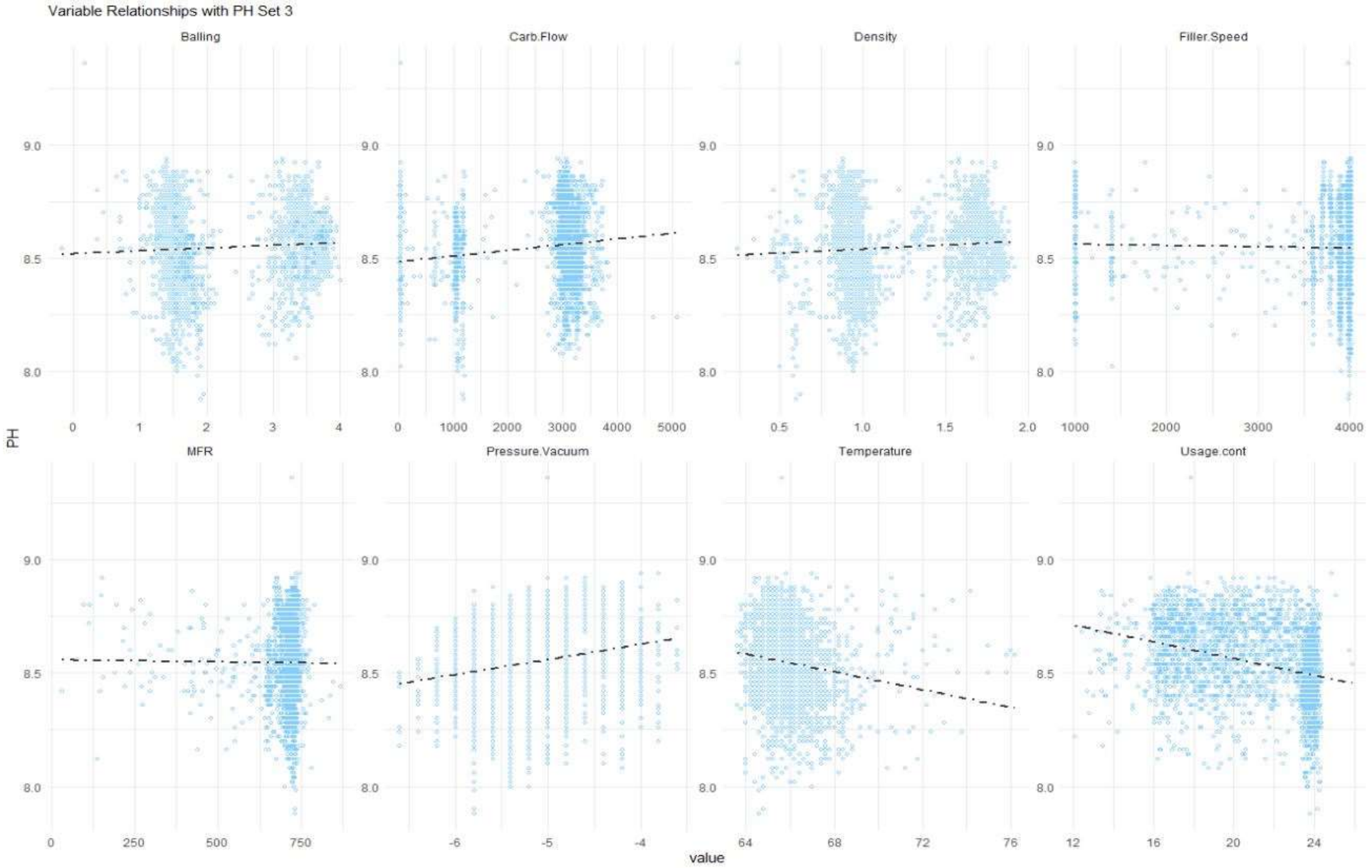
```r
# 1st set of variables - excluding brand code (categorical)
ph[c(2:9,26)] %>%
  gather(variable, value, -PH) %>%
  ggplot(., aes(value, PH)) +
  labs(
    subtitle =
      "Variable Relationships with Yield Set 1") +
  geom_point(fill = "white",
             size=1,
             shape=1,
             color="light sky blue") +
  geom_smooth(formula = y~x,
              method = "lm",
              size=1,
              se = TRUE,
              color = "grey24",
              linetype = "dotdash",
              alpha=0.25) +
  facet_wrap(~variable,
             scales ="free",
             ncol = 4) +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_blank(),
        axis.ticks = element_blank())
```


Variable Relationships with PH Set 1
Carb.Pressure
Carb.Temp
Carb.Volume
Fill.Ounces
PC.Volume
PSC
PSC.CO2
PSC.Fill
PH
value

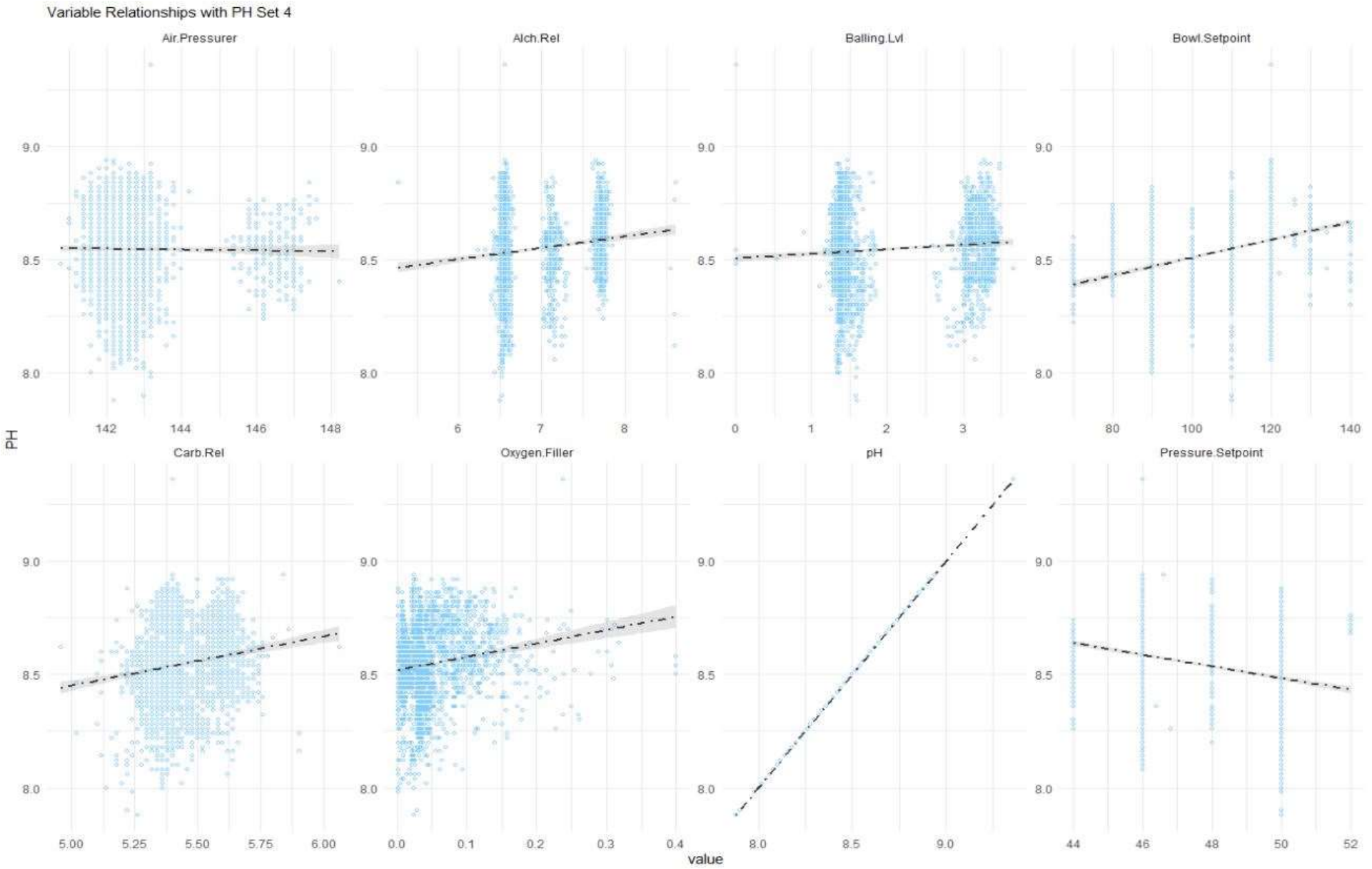
```r
# 2nd set of variables
ph[c(10:17,26)] %>%
  gather(variable, value, -PH) %>%
  ggplot(., aes(value, PH)) +
  labs(
    subtitle =
      "Variable Relationships with Yield Set 2") +
  geom_point(fill = "white",
             size=1,
             shape=1,
             color="light sky blue") +
  geom_smooth(formula = y~x,
              method = "lm",
              size=1,
              se = TRUE,
              color = "grey24",
              lty = "dotdash",
              alpha=0.25) +
  facet_wrap(~variable,
             scales ="free",
             ncol = 4)
```


Variable Relationships with PH Set 2
Carb.Pressure1
Fill.Pressure
Filler.Level
Hyd.Pressure1
Hyd.Pressure2
Hyd.Pressure3
Hyd.Pressure4
Mnf.Flow
PH
value
9.0
8.5
8.0
110
120
130
140
40
50
60
75
100
125
150
0
20
40
60
0
10
20
30
40
50
50
75
100
125
-100
0
100
200

```r
# 3rd set of variables
ph[c(18:26)] %>%
  gather(variable, value, -PH) %>%
  ggplot(., aes(value, PH)) +
  labs(
    subtitle =
      "Variable Relationships with Yield Set 3") +
  geom_point(fill = "white",
             size=1,
             shape=1,
             color="light sky blue") +
  geom_smooth(formula = y~x,
              method = "lm",
              size=1,
              se = TRUE,
              color = "grey24",
              linetype = "dotdash",
              alpha=0.25,
              fill="white") +
  facet_wrap(~variable,
             scales ="free",
             ncol = 4)
```


Variable Relationships with PH Set 3
Balling
Carb.Flow
Density
Filler.Speed
MFR
Pressure.Vacuum
Temperature
Usage.cont
PH
value
9.0
8.5
8.0
0
1
2
3
4
1000
2000
3000
4000
5000
0.5
1.0
1.5
2.0
250
500
750
-6
-5
-4
64
68
72
76
12
16
20
24

```r
# last 8 variables
ph[c(27:33,26)] %>%
  mutate(pH = PH) %>%
  gather(variable, value, -PH) %>%
  ggplot(., aes(value, PH)) +
  labs(
    subtitle =
      "Variable Relationships with Yield Set 4") +
  labs(subtitle = ) +
  geom_point(fill = "white",
             size=1,
             shape=1,
             color="light sky blue") +
  geom_smooth(formula = y~x,
              method = "lm",
              size=1,
              se = TRUE,
              color = "grey24",
              linetype = "dotdash",
              alpha=0.25) +
  facet_wrap(~variable,
             scales ="free",
             ncol = 4)
```


Variable Relationships with PH Set 4
Air.Pressurer
Alch.Rel
Balling.Lvl
Bowl.Setpoint
Carb.Rel
Oxygen.Filler
pH
Pressure.Setpoint
PH
value

None of the variables appear to follow a linear relationship with 'PH.' If they were, it would look similar to the plot of 'PH,' with itself. Most variables are continuous. Excluding a handful of seemingly discrete numerical values such as 'Pressure.Setpoint,' 'Bowl.Setpoint,' and 'Alch.Rel,' there is an inherent randomness to all relationships with 'PH.' We repeat the visualization process to create histograms of each variable to review their distributions.

```
ph %>%
  gather(variable, value) %>%
  ggplot(., aes(value)) +
  ggtitle("Linearity of Values") +
  geom_histogram(stat="count",
                 binwidth = 10,
                 fill = "white",
                 color="light sky blue") +
  facet_wrap(~variable,
             scales ="free",
             ncol = 4) +
  theme(axis.text.x = element_blank(),
        axis.text.y = element_blank(),
        plot.title = element_blank())
```

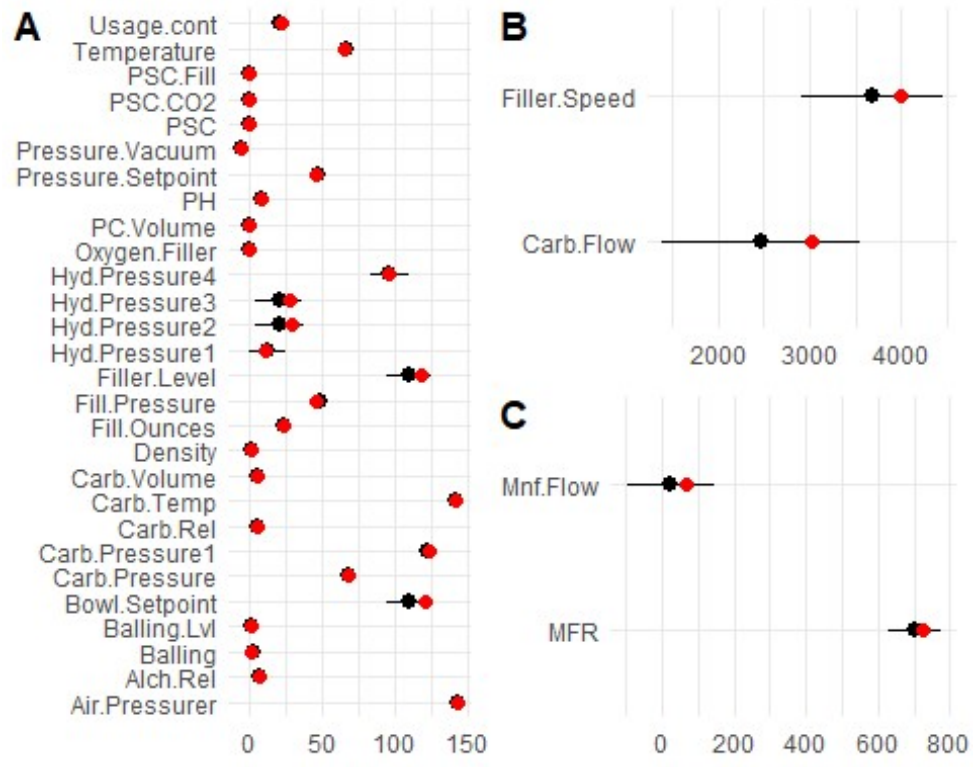

Air.Pressurer
Alch.Rel
Balling
Balling.Lvl
Bowl.Setpoint
Carb.Flow
Carb.Pressure
Carb.Pressure1
Carb.Rel
Carb.Temp
Carb.Volume
Density
Fill.Ounces
Fill.Pressure
Filler.Level
Filler.Speed
Hyd.Pressure1
Hyd.Pressure2
Hyd.Pressure3
Hyd.Pressure4
i..Brand.Code
MFR
Mnf.Flow
Oxygen.Filler
PC.Volume
PH
Pressure.Setpoint
Pressure.Vacuum
PSC
PSC.CO2
PSC.Fill
Temperature
Usage.cont

At a binwidth of 10, 'PH' appears normally distributed. However, many variables have two modes (biomodal) and are skewed. Temperature is skewed right due to a suspected outlier. We will need to fix this before modeling.

Additionally, the discrete numeric values formed by 'Pressure.Setpoint,' and 'Bowl.Setpoint,' in our scatterplots are confirmed. They can be spotted easily this way since they look more like our categorical variable "Brand.Code" than a continuous variable like 'Temperature.' Given these results, we must consider skew more closely. We use a point range function and visual to explore further.

```
# Function to calculate and set pointrange
xysdu <- function(x) {
   m <- mean(x)
   ymin <- m - sd(x)
   ymax <- m + sd(x)
   return(c(y = m, ymin = ymin, ymax = ymax))
}
# Full picture point range
ptrng.full <- ph %>%
  dplyr::select(where(is.numeric)) %>%
  gather(variable, value) %>%
  ggplot(aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color=
"black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None", axis.title.x = element_blank(), axis.title.
y = element_blank())
# Similar point ranges (smaller)
ptrng.small <- ph %>%
  dplyr::select(where(is.numeric)) %>%
  dplyr::select(-MFR, -Filler.Speed, -Carb.Flow, -Mnf.Flow) %>%
  gather(variable, value) %>%
  ggplot(aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color=
"black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None", axis.title.x = element_blank(), axis.title.
y = element_blank())
# Similar point ranges (Medium)
ptrng.med <- ph %>%
  dplyr::select(where(is.numeric)) %>%
  dplyr::select(MFR, Mnf.Flow) %>%
  gather(variable, value) %>%
  ggplot(aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color=
"black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None", axis.title.x = element_blank(), axis.title.
y = element_blank())
# Similar point ranges (larger)
```

```
ptrng.lrg <- ph %>%
  dplyr::select(where(is.numeric)) %>%
  dplyr::select(Filler.Speed, Carb.Flow) %>%
  gather(variable, value) %>%
  ggplot(aes(variable, value)) + coord_flip() +
  stat_summary(fun.data=xysdu, geom = "Pointrange", shape=16, size=.5, color=
"black") +
  stat_summary(fun.y=median, geom="point", shape=16, size=2, color="red") +
  theme(legend.position = "None", axis.title.x = element_blank(), axis.title.
y = element_blank())
ggarrange(ptrng.small,
          ggarrange(ptrng.lrg, ptrng.med, ncol = 1, labels = c("B", "C")), nr
ow = 1, labels = "A"
          )
```




The red dot on each variable range is its median while the black dot is its mean. The more of the black dot we see, the more skewed that distribution is. Here, we label three parts A, B, and C to denote a corresponding group of variables on similar scales with centers of their distribution closer together than other groups. It is best to simply think of these labels as A being the small-scale group, B showing the large-scale group, and C as a middle or medium scale group.

Using a point range visualization like this lets us check the magnitude of outliers by exploiting the difference in robustness between median and mean. It also helps us pick out exactly which variables are best for modeling by sighting variables with many points that influence their distribution more than other variables. Unfortunately, we are still not sure if

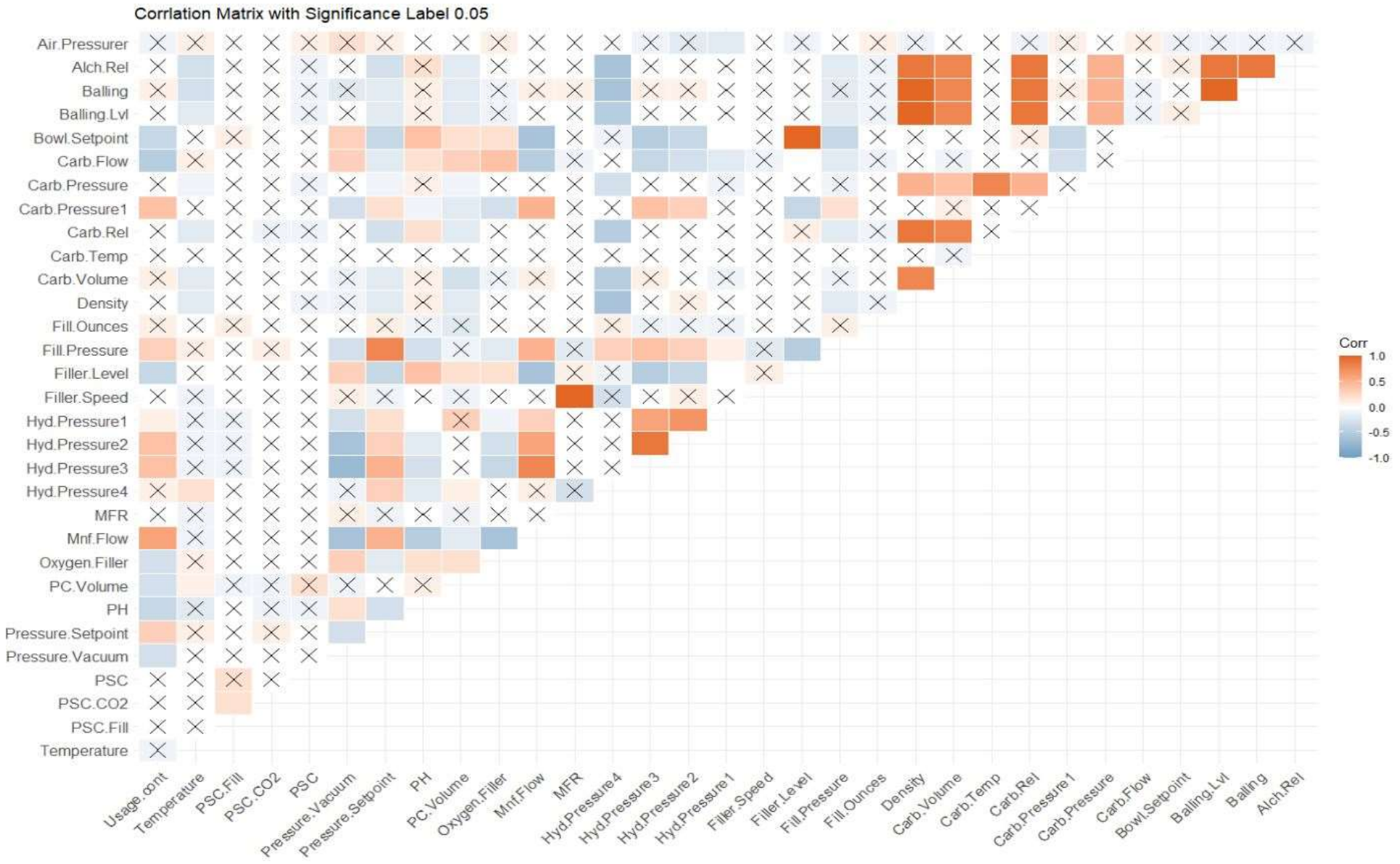
any of the seemingly outlier points were intentional so a fair bit of intuition is used to identify what should and should not be influencing the distribution.

When considering variable relationships, we also need to consider their correlations. In this final correlation plot we examine not only the variable's correlation with 'PH' but all variables' correlations with one another.

```
# Correlation Plot
order <- ph %>%
  dplyr::select(where(is.numeric)) %>%
  arrange() %>%
  gather(variable, value) %>%
  pivot_wider(id_cols = variable, names_from = variable, values_from = value)
%>%
  unnest()
order <- order[,order(colnames(order),decreasing=TRUE)]
cors <- cor(order, use = "complete.obs")
p.mat <- ggcorrplot::cor_pmat(cors, sig.level = 0.05)
sum(p.mat > .05)

## [1] 670

ggcorrplot(cors, "square", "lower",
           colors = c("#6D9EC1", "white", "#E46726"),
           outline.color = "white",
           digits = 1,
           p.mat=p.mat,
           hc.order = FALSE,
           hc.method = "complete") +
  coord_flip() +
  labs(title = "Corrlation Matrix with Significance Label 0.05")
```


Corrlation Matrix with Significance Label 0.05
Air.Pressurer
Alch.Rel
Balling
Balling.Lvl
Bowl.Setpoint
Carb.Flow
Carb.Pressure
Carb.Pressure1
Carb.Rel
Carb.Temp
Carb.Volume
Density
Fill.Ounces
Fill.Pressure
Filler.Level
Filler.Speed
Hyd.Pressure1
Hyd.Pressure2
Hyd.Pressure3
Hyd.Pressure4
MFR
Mnf.Flow
Oxygen.Filler
PC.Volume
PH
Pressure.Setpoint
Pressure.Vacuum
PSC
PSC.CO2
PSC.Fill
Temperature
Usage.cont
Temperature
PSC.Fill
PSC.CO2
PSC
Pressure.Vacuum
Pressure.Setpoint
PH
PC.Volume
Oxygen.Filler
Mnf.Flow
MFR
Hyd.Pressure4
Hyd.Pressure3
Hyd.Pressure2
Hyd.Pressure1
Filler.Speed
Filler.Level
Fill.Pressure
Fill.Ounces
Density
Carb.Volume
Carb.Temp
Carb.Rel
Carb.Pressure1
Carb.Pressure
Carb.Flow
Bowl.Setpoint
Balling.Lvl
Balling
Alch.Rel
Corr
1.0
0.5
0.0
-0.5
-1.0

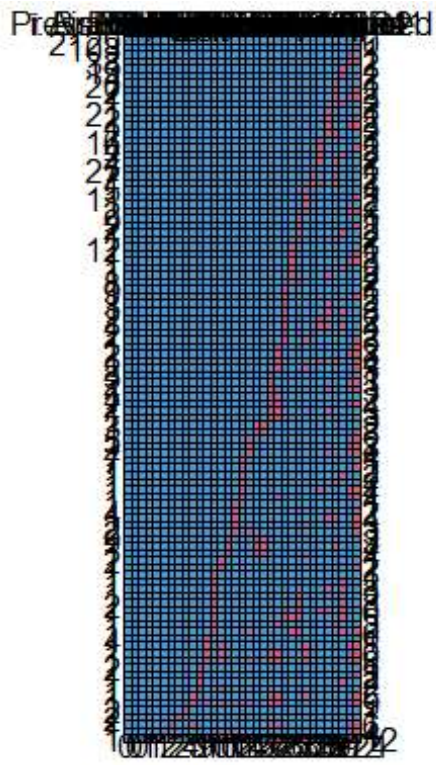
Although computed and shown for reference, we ignore insignificantly correlated variables with p-values greater than or equal to 0.05. Those we ignore have an 'X' drawn through its box. Variables directly measuring pressure and flow exhibit more negative control over 'PH' shown in darker blue. The only significant exception to this is 'Carb.Flow.' Meanwhile, 'Bowl.Setpoint,' 'Filler.Level,' 'Oxygen.Filler,' and 'Usage.Count' are the only remaining significantly correlated variables.

## Data Preparation

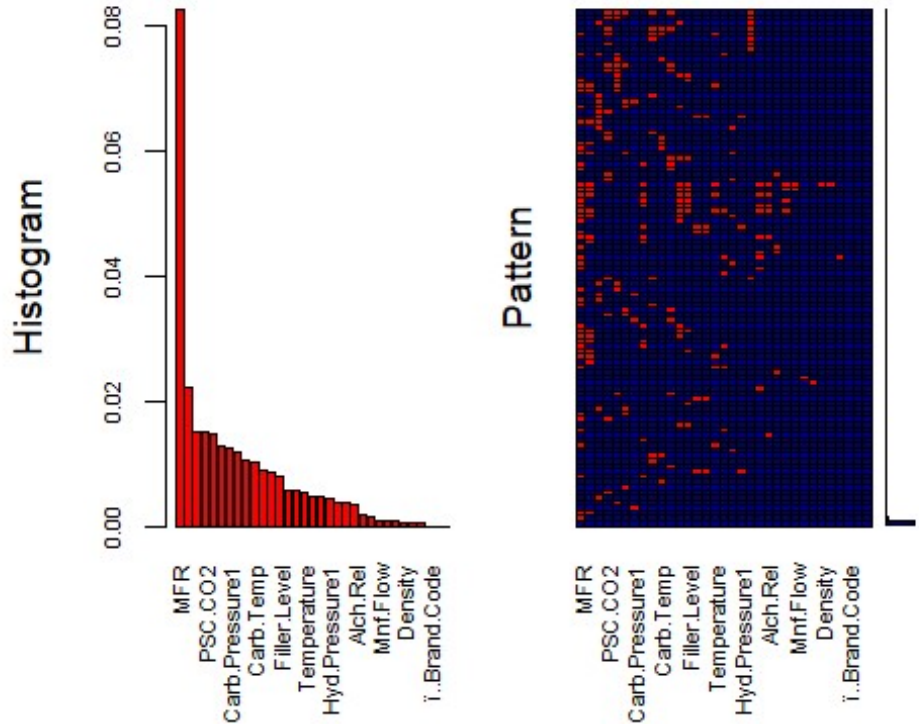
### Imputation

Before imputing, we check for patterns in the missing data. According to the mice method used in the md.pattern() function, there is no clear pattern. The errors are randomly dispersed. However, since we discovered some natural eb and flow in the collection process we double check with our own eyes.

```
# Check for missing data patterns with mice
na.pattern.mice <- md.pattern(ph) # No clear pattern
```



```
# Double check with hist and plot
aggr(ph, col=c('navyblue','red'),
                numbers=TRUE, sortVars=TRUE,
                labels=names(ph), cex.axis=.7,
                gap=3, ylab=c("Histogram","Pattern"))
```




```
##
##  Variables sorted by number of missings:
##           Variable        Count
##                MFR 0.0824581875
##       Filler.Speed 0.0221703617
##          PC.Volume 0.0151691949
##            PSC.CO2 0.0151691949
##         Fill.Ounces 0.0147802412
##                PSC 0.0128354726
##     Carb.Pressure1 0.0124465189
##      Hyd.Pressure4 0.0116686114
##      Carb.Pressure 0.0105017503
##          Carb.Temp 0.0101127966
##           PSC.Fill 0.0089459354
##      Fill.Pressure 0.0085569817
##       Filler.Level 0.0077790743
##      Hyd.Pressure2 0.0058343057
##      Hyd.Pressure3 0.0058343057
##        Temperature 0.0054453520
##      Oxygen.Filler 0.0046674446
##  Pressure.Setpoint 0.0046674446
##      Hyd.Pressure1 0.0042784909
##        Carb.Volume 0.0038895371
##           Carb.Rel 0.0038895371
##           Alch.Rel 0.0035005834
##         Usage.cont 0.0019447686
##                PH 0.0015558149
```

```
##           Mnf.Flow 0.0007779074
##          Carb.Flow 0.0007779074
##      Bowl.Setpoint 0.0007779074
##            Density 0.0003889537
##            Balling 0.0003889537
##        Balling.Lvl 0.0003889537
##      ï..Brand.Code 0.0000000000
##    Pressure.Vacuum 0.0000000000
##      Air.Pressurer 0.0000000000

# Small amount missing - simple median will do
obs.missing.perc <-
  (sum(ph.desc$obs) / (33*2571)*100)
# replaces NA with median (given a removal of missing values in calculation)
for (i in colnames(ph)) {
  ph[[i]][is.na(ph[[i]])] <- median(ph[[i]], na.rm=TRUE)
}
# Confirm none are missing
sum(is.na(ph))

## [1] 0
```

The histogram distributes the missing data as expected from our inferential statistics. The lion's share of missing values (8.2%) is concentrated under variable 'MFR' dwarfing the next variables by comparison. Our pattern plot also randomly spreads red pixelated rectangles on a dark blue background, indicating completely random missing values. Since the quantity of missing values is quite small (less than 1%) we perform a simple imputation by the median of each variable using a for loop.

## Outlier Extraction

To extract and remove outliers, we first identify which variables contain the outliers. Variable 'Brand.Code' is categorical and has no missing values so we exclude it by selecting all numeric variables (which includes everything else). We then identify and replace those outliers using a formula with upper and lower bounds to extract nonoutlier data, leaving the rest as outliers. Thus, we conveniently quantify these outliers as greater than or less than 1.5 times each variable's interquartile range.

```
# select numeric variables
ph.numerics <- ph %>%
  dplyr::select(where(is.numeric))
# remove outliers based on IQR
for (i in colnames(ph.numerics)) {
  iqr <- IQR(ph.numerics[[i]])
  q <- quantile(ph.numerics[[i]], probs = c(0.25, 0.75), na.rm = FALSE)
  qupper <- q[2]+1.5*iqr
  qlower <- q[1]+1.5*iqr
  outlier_free <- subset(ph.numerics, ph.numerics[[i]] > (q[1] - 1.5*iqr) & p
h.numerics[[i]] < (q[2]+1.5*iqr) )
}
```

```
ph.numerics <- outlier_free
# join outlier free numerics with categorical
Brand.Code <- ph$ï..Brand.Code
df <- cbind(Brand.Code, ph.numerics)
df.summary <- summary(df)
```

After identifying and replacing outliers by looping through each variable in the data, we join the outlier free numeric data with the excluded 'Brand.Code' variable. This retains the data types as read in and is complete with a binding function for data frames. We then check for missing data in 'Brand.Code' to ensure it all transferred appropriately and review the results to see that very few outliers were present.

## Transformations

Although the data did not exhibit linear trends, nor fit a typically gaussian pattern in its isolation, we consider transformations that would best normalize each variable's distribution. Using a function in the bestNormalize package we store the chosen transformations of each variable in a data frame named 'best.norms' along with corresponding metrics to transform them.

```
# Produce recommended transformations
df.nums <- df %>%
  dplyr::select(where(is.numeric))
best.norms <- df.nums[1:11,1:10]
for (i in colnames(df.nums)) {
  best.norms[[i]] <- bestNormalize(df.nums[[i]],
                                   allow_orderNorm = FALSE,
                                   out_of_sample =FALSE)
}
best.norms$Carb.Volume$chosen_transform

## Standardized Yeo-Johnson Transformation with 2571 nonmissing obs.:
##  Estimated statistics:
##  - lambda = -4.99994
##  - mean (before standardization) = 0.1999832
##  - sd (before standardization) = 1.568536e-06
```

From this data frame, we can call upon the desired variables to know which transformation is best at a specific time. The ideal transformation to normalize the 'Carb.Volume' variable is shown in the code as an example. These will be used in conjunction with model importance when building models.

## Split and Selections

We split the data set giving 70% to training and 30% to testing data frames. Using a forward and backward traveling stepwise regression that considered Akaike information criterion (AIC) to evaluate model fit, we determined the best variables. Quite intuitively, any coefficient in the model that had a p-value lower than 0.05 was used. This indicated a significant coefficient to us and was the basis for our selection-based models.

```r
# Split 70-30 training/test
set.seed(1102)
index <- createDataPartition(
    df$PH, p = .7,
    list = FALSE, times = 1)
train <- df[index,]
test <- df[-index,]
# Create train x and y
trainx <- train %>%
  dplyr::select(-PH)
trainy <- train$PH
testx <- test %>%
  dplyr::select(-PH)
testy <- test$PH

# StepAIC Model Selectors
df.selected <- df %>%
  dplyr::select(Brand.Code,
                Carb.Volume,
                Fill.Ounces,
                PC.Volume,
                Carb.Temp,
                PSC,
                PSC.Fill,
                PSC.CO2,
                PH,
                Mnf.Flow,
                Carb.Pressure1,
                Fill.Pressure,
                Hyd.Pressure2,
                Hyd.Pressure3,
                Filler.Level,
                Temperature,
                Usage.cont,
                Carb.Flow,
                Density,
                Balling,
                Pressure.Vacuum,
                Oxygen.Filler,
                Bowl.Setpoint,
                Pressure.Setpoint,
                Alch.Rel,
                Balling.Lvl)
set.seed(1102)
index <- createDataPartition(
    df.selected$PH, p = .7,
    list = FALSE, times = 1)
train.selected <- df.selected[index,]
test.selected <- df.selected[-index,]
# Create selected variation of train x and y
```

```
trainx.selected <- train.selected %>%
  dplyr::select(-PH)
trainy.selected <- train.selected$PH
testx.selected <- test.selected %>%
  dplyr::select(-PH)
testy.selected <- test.selected$PH
```

## Model Building

Given a clean data set and options of using transformed and down selected data, we can begin to build models for predicting our target variable, 'PH.' There are 17 models, so we organize them into three categories: parametric, nonparametric, and specialized. We refrain from explaining why the models are selected until the 'model selection' section but include importance values for each predictor used in the models.

For a quick reference, in parametric models the parameters used to generate predictions are fixed. Whereas nonparametric models tend to change parameters with the size of data. To further generalize, if the model has a predetermined outcome between predictor and response, then it is parametric (Ex: linear, multiple regression). All others are nonparametric. Specialized models can be either type but use the down-selected data with a slightly different cleaning process to adapt to their chosen transformations as mentioned previously.

### Parametric Models

We begin with a stepwise regression model that travels forwards and backwards through the parameters to select the best ones. It uses Akaike information criterion (AIC) to evaluate model fit. This model was used to generate the previously selected parameters in our 'Select and Split' section.

```
model.pm <- lm(PH~.,train)
pm <- stepAIC(model.pm,
              trace = F,
              direction = "both")
p <- summary(pm)$call
pm <- lm(p[2], df)
pmPred <- predict(model.pm, newdata = test)
pm_test <- data.frame(
  postResample(pred = pmPred, obs = test$PH))
summary(pm)

##
## Call:
## lm(formula = p[2], data = df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52505 -0.07555  0.00974  0.08777  0.75646
```

```
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)         1.047e+01  8.180e-01  12.795  < 2e-16 ***
## Brand.CodeA        -3.429e-03  2.445e-02  -0.140  0.88847
## Brand.CodeB         7.337e-02  1.356e-02   5.412 6.80e-08 ***
## Brand.CodeC        -7.203e-02  1.477e-02  -4.877 1.14e-06 ***
## Brand.CodeD         5.176e-02  2.770e-02   1.869  0.06176 .
## Carb.Volume        -6.370e-02  4.546e-02  -1.401  0.16123
## Fill.Ounces        -7.976e-02  3.178e-02  -2.510  0.01213 *
## Carb.Temp           1.023e-03  6.715e-04   1.523  0.12785
## PSC                -1.418e-01  5.428e-02  -2.612  0.00906 **
## Mnf.Flow           -7.100e-04  4.519e-05 -15.711  < 2e-16 ***
## Carb.Pressure1      6.678e-03  6.832e-04   9.774  < 2e-16 ***
## Fill.Pressure       2.691e-03  1.178e-03   2.284  0.02246 *
## Hyd.Pressure2      -8.118e-04  4.676e-04  -1.736  0.08265 .
## Hyd.Pressure3       2.853e-03  5.667e-04   5.035 5.11e-07 ***
## Hyd.Pressure4      -1.881e-04  2.831e-04  -0.664  0.50643
## Filler.Level       -8.855e-04  4.979e-04  -1.778  0.07547 .
## Temperature        -1.421e-02  2.229e-03  -6.372 2.20e-10 ***
## Usage.cont         -6.488e-03  1.114e-03  -5.825 6.42e-09 ***
## Carb.Flow           9.474e-06  3.295e-06   2.875  0.00407 **
## Density            -1.251e-01  2.768e-02  -4.518 6.52e-06 ***
## Balling            -6.365e-02  2.180e-02  -2.920  0.00353 **
## Pressure.Vacuum    -1.760e-02  6.979e-03  -2.522  0.01174 *
## Oxygen.Filler      -2.835e-01  6.738e-02  -4.207 2.68e-05 ***
## Bowl.Setpoint       2.957e-03  5.174e-04   5.715 1.23e-08 ***
## Pressure.Setpoint  -8.948e-03  1.912e-03  -4.680 3.02e-06 ***
## Alch.Rel            6.630e-02  2.066e-02   3.209  0.00135 **
## Balling.Lvl         1.022e-01  2.054e-02   4.979 6.83e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1327 on 2544 degrees of freedom
## Multiple R-squared:  0.4134, Adjusted R-squared:  0.4074
## F-statistic: 68.95 on 26 and 2544 DF,  p-value: < 2.2e-16
```

```
caret::varImp(model.pm) %>%
  dplyr::arrange(desc(Overall))
```

```
##                      Overall
## Mnf.Flow          12.90552066
## Carb.Pressure1     8.46144560
## Temperature        5.91895482
## Usage.cont         5.28392577
## Brand.CodeB        5.06446945
## Bowl.Setpoint      4.63529384
## Balling.Lvl        4.58220778
## Oxygen.Filler      4.27226520
## Hyd.Pressure3      4.15673945
```

```
## Pressure.Setpoint  4.11096992
## Balling            2.96562686
## Density            2.75035127
## Pressure.Vacuum    2.63457991
## Brand.CodeC        2.55784872
## Alch.Rel           2.41598768
## Fill.Pressure      2.41161929
## Carb.Flow          2.29517315
## Fill.Ounces        1.92700494
## Carb.Temp          1.63740336
## PSC                1.57143255
## Hyd.Pressure4      1.53898269
## Hyd.Pressure2      1.52357509
## Carb.Volume        1.49035900
## Filler.Level       1.47081618
## Brand.CodeD        1.32956678
## PSC.Fill           1.24950798
## Filler.Speed       0.98706190
## Air.Pressurer      0.90918827
## Carb.Pressure      0.90346164
## PSC.CO2            0.86633297
## PC.Volume          0.55040489
## MFR                0.40486454
## Carb.Rel           0.35254386
## Hyd.Pressure1      0.09522975
## Brand.CodeA        0.05736799
```

With the stepwise regression complete, we try another method colloquially known as the 'kitchen sink' or KS model. This will throw everything but the kitchen sink into the model as parameters and return significance values to us in the form of p-values. Performing this model helps to evaluate the selection of the previous model.

```
model.ks <- lm(PH~., train)
ksPred <- predict(model.ks, newdata = test)
ks_test <- data.frame(
  postResample(pred = ksPred, obs = test$PH))
summary(model.ks)
```

```
##
## Call:
## lm(formula = PH ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52964 -0.07675  0.00804  0.08685  0.77027
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.071e+01  1.117e+00   9.585  < 2e-16 ***
## Brand.CodeA       -1.705e-03  2.972e-02  -0.057  0.95426
```

```
## Brand.CodeB        8.759e-02  1.730e-02   5.064 4.52e-07 ***
## Brand.CodeC       -4.790e-02  1.872e-02  -2.558  0.01062 *
## Brand.CodeD        4.526e-02  3.404e-02   1.330  0.18383
## Carb.Volume       -9.996e-02  6.707e-02  -1.490  0.13631
## Fill.Ounces       -7.456e-02  3.869e-02  -1.927  0.05414 .
## PC.Volume         -3.482e-02  6.325e-02  -0.550  0.58211
## Carb.Pressure     -2.222e-03  2.459e-03  -0.903  0.36640
## Carb.Temp          3.247e-03  1.983e-03   1.637  0.10172
## PSC               -1.097e-01  6.978e-02  -1.571  0.11626
## PSC.Fill          -3.461e-02  2.770e-02  -1.250  0.21165
## PSC.CO2           -6.503e-02  7.506e-02  -0.866  0.38643
## Mnf.Flow          -7.201e-04  5.580e-05 -12.906  < 2e-16 ***
## Carb.Pressure1     7.063e-03  8.347e-04   8.461  < 2e-16 ***
## Fill.Pressure      3.490e-03  1.447e-03   2.412  0.01598 *
## Hyd.Pressure1      4.179e-05  4.388e-04   0.095  0.92414
## Hyd.Pressure2     -9.955e-04  6.534e-04  -1.524  0.12779
## Hyd.Pressure3      2.938e-03  7.069e-04   4.157 3.38e-05 ***
## Hyd.Pressure4     -5.453e-04  3.543e-04  -1.539  0.12399
## Filler.Level      -8.596e-04  5.844e-04  -1.471  0.14152
## Filler.Speed      -6.050e-06  6.129e-06  -0.987  0.32375
## Temperature       -1.598e-02  2.700e-03  -5.919 3.88e-09 ***
## Usage.cont        -7.221e-03  1.367e-03  -5.284 1.42e-07 ***
## Carb.Flow          9.879e-06  4.304e-06   2.295  0.02184 *
## Density           -9.350e-02  3.400e-02  -2.750  0.00601 **
## MFR                2.131e-05  5.264e-05   0.405  0.68563
## Balling           -8.026e-02  2.706e-02  -2.966  0.00306 **
## Pressure.Vacuum   -2.293e-02  8.703e-03  -2.635  0.00850 **
## Oxygen.Filler     -3.533e-01  8.271e-02  -4.272 2.04e-05 ***
## Bowl.Setpoint      2.861e-03  6.173e-04   4.635 3.83e-06 ***
## Pressure.Setpoint -9.619e-03  2.340e-03  -4.111 4.12e-05 ***
## Air.Pressurer     -2.563e-03  2.819e-03  -0.909  0.36337
## Alch.Rel           6.718e-02  2.781e-02   2.416  0.01579 *
## Carb.Rel           1.964e-02  5.571e-02   0.353  0.72447
## Balling.Lvl        1.169e-01  2.551e-02   4.582 4.92e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1332 on 1765 degrees of freedom
## Multiple R-squared:  0.4119, Adjusted R-squared:  0.4002
## F-statistic: 35.32 on 35 and 1765 DF,  p-value: < 2.2e-16

caret::varImp(model.ks) %>%
  dplyr::arrange(desc(Overall))

##                      Overall
## Mnf.Flow          12.90552066
## Carb.Pressure1     8.46144560
## Temperature        5.91895482
## Usage.cont         5.28392577
## Brand.CodeB        5.06446945
```

```
## Bowl.Setpoint      4.63529384
## Balling.Lvl        4.58220778
## Oxygen.Filler      4.27226520
## Hyd.Pressure3      4.15673945
## Pressure.Setpoint  4.11096992
## Balling            2.96562686
## Density            2.75035127
## Pressure.Vacuum    2.63457991
## Brand.CodeC        2.55784872
## Alch.Rel           2.41598768
## Fill.Pressure      2.41161929
## Carb.Flow          2.29517315
## Fill.Ounces        1.92700494
## Carb.Temp          1.63740336
## PSC                1.57143255
## Hyd.Pressure4      1.53898269
## Hyd.Pressure2      1.52357509
## Carb.Volume        1.49035900
## Filler.Level       1.47081618
## Brand.CodeD        1.32956678
## PSC.Fill           1.24950798
## Filler.Speed       0.98706190
## Air.Pressurer      0.90918827
## Carb.Pressure      0.90346164
## PSC.CO2            0.86633297
## PC.Volume          0.55040489
## MFR                0.40486454
## Carb.Rel           0.35254386
## Hyd.Pressure1      0.09522975
## Brand.CodeA        0.05736799
```

We begin with a stepwise regression model that travels forwards and backwards through the parameters to select the best ones. It uses Akaike information criterion (AIC) to evaluate model fit. This model was used to generate the previously selected parameters in our 'Select and Split' section.

```
trctrl<- trainControl(method="repeatedcv",
                      number=3,
                      repeats=2)
model.rr <- caret::train(PH~., data=train,
                      method="ridge",
                      trControl=trctrl)
rrPred <- predict(model.rr, newdata = test)
rr_test <-data.frame(
  postResample(pred = rrPred,
               obs = test$PH))
model.rr

## Ridge Regression
##
```

```
## 1801 samples
##   32 predictor
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 2 times)
## Summary of sample sizes: 1200, 1201, 1201, 1200, 1200, 1202, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE       Rsquared   MAE
##   0e+00   0.1359779  0.3786719  0.1048789
##   1e-04   0.1359618  0.3787656  0.1048805
##   1e-01   0.1364600  0.3722408  0.1068864
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 1e-04.
```

```
caret::varImp(model.rr)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 32)
##
##                   Overall
## Oxygen.Filler      100.00
## Filler.Level        79.70
## Balling             65.91
## Mnf.Flow            62.57
## Filler.Speed        61.87
## Hyd.Pressure3       53.20
## Bowl.Setpoint       52.97
## Hyd.Pressure2       52.40
## Hyd.Pressure1       51.54
## Fill.Pressure       50.10
## Usage.cont          44.60
## Pressure.Setpoint   44.60
## Carb.Pressure1      43.33
## Balling.Lvl         33.79
## Carb.Rel            33.55
## Density             31.96
## Brand.Code          30.22
## Carb.Flow           27.01
## Temperature         25.59
## PSC                 25.49
```

With the stepwise regression complete, we try another method colloquially known as the 'kitchen sink' or KS model. This will throw everything but the kitchen sink into the model as parameters and return significance values to us in the form of p-values. Performing this model helps to evaluate the selection of the previous model.

```
model.glmb <- train(x = trainx,
                    y = trainy,
                    method = "glmboost",
                    preProcess = c("center", "scale"),
                    tuneLength = 10)
glmbPred <- predict(model.glmb, newdata = test)
glmb_test <- data.frame(
  postResample(pred = glmbPred, obs = test$PH))
model.glmb
```

```
## Boosted Generalized Linear Model
##
## 1801 samples
##   32 predictor
##
## Pre-processing: centered (31), scaled (31), ignore (1)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1801, 1801, 1801, 1801, 1801, 1801, ...
## Resampling results across tuning parameters:
##
##   mstop  RMSE       Rsquared   MAE
##    50    0.1438774  0.3222270  0.1137794
##   100    0.1404775  0.3453766  0.1110714
##   150    0.1390293  0.3556490  0.1097430
##   200    0.1381310  0.3621603  0.1088404
##   250    0.1375588  0.3662475  0.1082176
##   300    0.1371918  0.3688031  0.1077793
##   350    0.1369452  0.3705247  0.1074655
##   400    0.1367734  0.3717465  0.1072303
##   450    0.1366391  0.3727334  0.1070371
##   500    0.1365224  0.3736747  0.1068703
##
## Tuning parameter 'prune' was held constant at a value of no
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were mstop = 500 and prune = no.
```

```
caret::varImp(model.glmb)
```

```
## glmboost variable importance
##
##   only 20 most important variables shown (out of 36)
##
##                   Overall
## Brand.CodeC       100.000
## Mnf.Flow           74.113
## Brand.CodeB        36.937
## Bowl.Setpoint      28.245
## Brand.CodeD        27.377
## Carb.Pressure1     27.198
## Brand.CodeA        26.197
```

```
## Hyd.Pressure3      23.468
## Usage.cont         21.754
## Temperature        20.672
## Pressure.Setpoint  12.951
## Alch.Rel           11.730
## Oxygen.Filler       8.287
## Fill.Ounces         6.770
## Fill.Pressure       5.947
## PSC                 5.661
## Carb.Temp           4.879
## Carb.Volume         4.867
## Density             4.691
## Balling.Lvl         4.433
```

A ridge regression is performed next. While this model was not expected to be the best model, it set up a framework to understand how well the standard error of the predictors influences our response. Since ridge regression acts to limit standard error, we would expect that if the model turned out to be one of the best, there would be significant standard error in our predictors. From this model we also refine the level of importance we could assign to each predictor.

```
model.pls <- train(x = trainx,
                   y = trainy,
                   method = "pls",
                   preProcess = c("center", "scale"),
                   tuneLength = 10)
plsPred <- predict(model.pls, newdata = test)
pls_test <- data.frame(
  postResample(pred = plsPred, obs = test$PH))
model.pls
```

```
## Partial Least Squares
##
## 1801 samples
##   32 predictor
##
## Pre-processing: centered (31), scaled (31), ignore (1)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1801, 1801, 1801, 1801, 1801, 1801, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     0.1543979  0.2075986  0.1226774
##    2     0.1457227  0.2935975  0.1146515
##    3     0.1429901  0.3202799  0.1127463
##    4     0.1413624  0.3359289  0.1107825
##    5     0.1387176  0.3606534  0.1077492
##    6     0.1376388  0.3708057  0.1073084
##    7     0.1369509  0.3770225  0.1063951
##    8     0.1368718  0.3778315  0.1063153
```

```
##    9     0.1368008  0.3785119  0.1060149
##   10     0.1368091  0.3785562  0.1058738
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 9.

caret::varImp(model.pls)

## pls variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                    Overall
## Mnf.Flow           100.00
## Temperature         64.89
## Usage.cont          63.02
## Bowl.Setpoint       54.69
## Hyd.Pressure3       49.75
## Brand.CodeB         48.03
## Filler.Level        44.25
## Pressure.Setpoint   43.27
## Hyd.Pressure2       41.58
## Brand.CodeC         39.63
## Carb.Pressure1      39.41
## Fill.Pressure       38.53
## Pressure.Vacuum     30.48
## PSC                 28.42
## Carb.Flow           28.27
## Hyd.Pressure4       27.91
## Fill.Ounces         27.54
## Oxygen.Filler       27.35
## Density             21.56
## Balling.Lvl         21.46
```

From previous plots, we can infer that the data is not linear and thus linear models are likely moot. The next model is called generalized linear modeling (GLM) and we give a boost (GLMB). In it, we accept that fact but attempt to form a link between predictors and response. This is fulfilled by a link function that reviews many potential distributions to assess which has the best fit. The boost is present to support the model in its search for the best link. Results provide another perspective on our predictor relationships so that hopefully, we can pick the best ones.

## Nonparametric Models

For the final distinctly parametric model on nonspecialized data, we have a partial least squares (PLS) regression. With this, the dimensions of the model are reduced to focus the model on the least harmful set of predictors. This subset of predictors could be used to enhance other models as well. We give close attention to level of importance of each predictor in these models as they will inform the models we think will perform best.

```
marsGrid <- expand.grid(.degree=1:2,
                        .nprune=2:10)
model.mar <- train(x = trainx,
                   y = trainy,
                   method = "earth",
                   preProcess = c("center", "scale"),
                   tuneGrid = marsGrid)
marPred <- predict(model.mar, newdata = test)
mar_test <- data.frame(
  postResample(pred = marPred, obs = test$PH))
model.mar
```

```
## Multivariate Adaptive Regression Spline
##
## 1801 samples
##   32 predictor
##
## Pre-processing: centered (31), scaled (31), ignore (1)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1801, 1801, 1801, 1801, 1801, 1801, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE       Rsquared   MAE
##   1        2      0.1532229  0.2187284  0.1199211
##   1        3      0.1466596  0.2840057  0.1147312
##   1        4      0.1447724  0.3022666  0.1126522
##   1        5      0.1434345  0.3149132  0.1115407
##   1        6      0.1420530  0.3282894  0.1101166
##   1        7      0.1402804  0.3452835  0.1085698
##   1        8      0.1389846  0.3573650  0.1071394
##   1        9      0.1380323  0.3661995  0.1061904
##   1       10      0.1366826  0.3785272  0.1050132
##   2        2      0.1533283  0.2174148  0.1198591
##   2        3      0.1475293  0.2753429  0.1152093
##   2        4      0.1460076  0.2910286  0.1132334
##   2        5      0.1451231  0.3014407  0.1121105
##   2        6      0.1443553  0.3108761  0.1106718
##   2        7      0.1431601  0.3244186  0.1089932
##   2        8      0.1426790  0.3309937  0.1080873
##   2        9      0.1407302  0.3482979  0.1067805
##   2       10      0.1403491  0.3529197  0.1057961
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 10 and degree = 1.
```

```
caret::varImp(model.mar)
```

```
## earth variable importance
##
##                 Overall
```

```
## Mnf.Flow        100.000
## Brand.CodeC      61.415
## Pressure.Vacuum  38.055
## Usage.cont       37.324
## Temperature      37.324
## Balling          22.914
## Alch.Rel         17.472
## Carb.Pressure1    8.656
## Bowl.Setpoint     0.000
```

A multivariate adaptive regression spline (MARS) models nonlinearity among predictors and their outcome. This is partly why it was selected for our first parametric model. It also works well at discovering patterns in interactions better than linear regression models by using hinge functions. It is best to think of these as drawing a straight line through points with slight kinks when the data changes direction. These kinks can make a MARS model highly accurate in prediction. We center and scale the data prior to processing.

```
ctl <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3)
mtry <- sqrt(ncol(train))
tunegrid <- expand.grid(.mtry=mtry)
model.rf <- train(PH~.,
                  data=train,
                  method='rf',
                  tuneGrid=tunegrid,
                  preProcess = c("center", "scale"),
                  trControl=ctl)
rfPred <- predict(model.rf, newdata = test)
rf_test <- data.frame(
  postResample(pred = rfPred, obs = test$PH))
model.rf
```

```
## Random Forest
##
## 1801 samples
##   32 predictor
##
## Pre-processing: centered (35), scaled (35)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1622, 1621, 1619, 1621, 1621, 1619, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1050206  0.6552789  0.07749893
##
## Tuning parameter 'mtry' was held constant at a value of 5.744563
```

```
caret::varImp(model.rf)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                 Overall
## Mnf.Flow         100.00
## Usage.cont        60.32
## Filler.Level      44.23
## Bowl.Setpoint     34.54
## Temperature       34.40
## Oxygen.Filler     34.03
## Carb.Rel          32.22
## Brand.CodeC       30.01
## Balling.Lvl       29.66
## Pressure.Vacuum   28.53
## Balling           25.97
## Alch.Rel          24.45
## Carb.Pressure1    24.37
## Carb.Flow         22.54
## Filler.Speed      22.44
## Air.Pressurer     21.14
## Density           21.13
## Fill.Pressure     20.48
## Hyd.Pressure3     20.02
## Carb.Volume       16.92
```

Another common nonparametric model generally good at prediction is the random forest (RF) model. This kind of model is built on decision trees that can merge limbs to form more accurate predictions than simple regression. It also adds some degree of randomness while growing its trees to improve performance. Here, again the data is centered and scaled since many of the predictors were not close to the same magnitude and some were far from one another's average in their distributions.

```
# Neural net may take several minutes to run
nnet_grid <- expand.grid(.decay =
                           c(0, 0.01, .1),
                         .size = c(1:10),
                         .bag = FALSE)
nnet_maxnwts <- 5 * ncol(train) + 5 + 1
model.nnet <- train(
  PH ~ ., data = train, method = "avNNet",
  center = TRUE,
  scale = TRUE,
  tuneGrid = nnet_grid,
  trControl = trainControl(method = "cv"),
  linout = TRUE,
  trace = FALSE,
  MaxNWts = nnet_maxnwts,
  maxit = 500
)
```

```
nnetPred <- predict(model.nnet, newdata = test)
nnet_test <- data.frame(
  postResample(pred = nnetPred, obs = test$PH))
model.nnet

## Model Averaged Neural Network
##
## 1801 samples
##   32 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1622, 1621, 1621, 1622, 1621, 1621, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE       Rsquared    MAE
##   0.00    1    0.1717531  0.01138139  0.13782752
##   0.00    2    0.1710501  0.04241839  0.13715615
##   0.00    3    0.1702651  0.05393671  0.13642940
##   0.00    4    0.1710365  0.02714215  0.13729792
##   0.00    5          NaN         NaN         NaN
##   0.00    6          NaN         NaN         NaN
##   0.00    7          NaN         NaN         NaN
##   0.00    8          NaN         NaN         NaN
##   0.00    9          NaN         NaN         NaN
##   0.00   10          NaN         NaN         NaN
##   0.01    1    0.1425242  0.39555388  0.11204711
##   0.01    2    0.1381662  0.36730544  0.10415747
##   0.01    3    0.1401005  0.38522521  0.10231663
##   0.01    4    0.1342702  0.40157581  0.09957351
##   0.01    5          NaN         NaN         NaN
##   0.01    6          NaN         NaN         NaN
##   0.01    7          NaN         NaN         NaN
##   0.01    8          NaN         NaN         NaN
##   0.01    9          NaN         NaN         NaN
##   0.01   10          NaN         NaN         NaN
##   0.10    1    0.1388339  0.37985203  0.10869935
##   0.10    2    0.1331443  0.40524911  0.10211626
##   0.10    3    0.1302313  0.42826566  0.09872251
##   0.10    4    0.1311139  0.42790598  0.09774633
##   0.10    5          NaN         NaN         NaN
##   0.10    6          NaN         NaN         NaN
##   0.10    7          NaN         NaN         NaN
##   0.10    8          NaN         NaN         NaN
##   0.10    9          NaN         NaN         NaN
##   0.10   10          NaN         NaN         NaN
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
```

```
## The final values used for the model were size = 3, decay = 0.1 and bag = F
ALSE.

caret::varImp(model.nnet)

## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 32)
##
##                    Overall
## Oxygen.Filler       100.00
## Filler.Level         79.70
## Balling              65.91
## Mnf.Flow             62.57
## Filler.Speed         61.87
## Hyd.Pressure3        53.20
## Bowl.Setpoint        52.97
## Hyd.Pressure2        52.40
## Hyd.Pressure1        51.54
## Fill.Pressure        50.10
## Usage.cont           44.60
## Pressure.Setpoint    44.60
## Carb.Pressure1       43.33
## Balling.Lvl          33.79
## Carb.Rel             33.55
## Density              31.96
## Brand.Code           30.22
## Carb.Flow            27.01
## Temperature          25.59
## PSC                  25.49
```

Inspired by the human brain, neural networks (NNET) have become popular for discovering patterns in data. It works by training or adjusting an input with a weighted value based on the performance of previous inputs. When the output is correct, additional weight is given to that input. For this reason, neural networks a principally limited by the number of neurons available for inputs and the time required to train them. Many problems benefit from the use of NNETs because the algorithm can produce highly accurate results as long as the data going into in is capable.

```
model.cf <- train(PH~.,
                  data=train,
                  method="cforest",
                  trControl=trctrl,
                  preProcess = c("center", "scale"),
                  tuneLength =2)
cfPred <- predict(model.cf, newdata = test)
cf_test <- data.frame(postResample(pred = cfPred, obs = test$PH))
model.cf
```

```
## Conditional Inference Random Forest
##
## 1801 samples
##   32 predictor
##
## Pre-processing: centered (35), scaled (35)
## Resampling: Cross-Validated (3 fold, repeated 2 times)
## Summary of sample sizes: 1200, 1201, 1201, 1201, 1201, 1200, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##    2    0.1383423  0.4268640  0.10883377
##   35    0.1104986  0.5930356  0.08160401
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 35.
```

```
caret::varImp(model.cf)
```

```
## cforest variable importance
##
##   only 20 most important variables shown (out of 35)
##
##                 Overall
## Mnf.Flow        100.000
## Brand.CodeC      20.473
## Brand.CodeD      18.028
## Air.Pressurer    11.641
## Usage.cont       10.664
## Pressure.Vacuum   9.037
## Bowl.Setpoint     8.089
## Brand.CodeB       7.602
## Oxygen.Filler     7.546
## Carb.Rel          4.138
## Hyd.Pressure3     3.916
## Temperature       3.265
## Carb.Pressure1    3.260
## Alch.Rel          3.222
## Carb.Flow         3.090
## Balling.Lvl       2.153
## Carb.Volume       2.133
## Density           1.968
## Filler.Level      1.794
## Fill.Pressure     1.621
```

Conditional forests (CF) are popular as well since they can generally handle smaller amounts of data with little significance better than other nonparametric and tree-based methods. Because of the ability of this model to conditionally build trees with data inputs complex interactions and predictors that share highly, potentially overly correlated

relationships are not an issue for the model. We center and scale here again in hopes of improving accuracy. Through this model we expect to have the best performance overall.

Support Vector

```
svm_grid <-  expand.grid(C = c(1,1000))
model.svm <- train(PH~., data = train,
                   method = 'svmRadialCost',
                   trControl = trctrl,
                   tuneGrid = svm_grid)
svmPred <- predict(model.svm, newdata = test)
svm_test <- data.frame(
  postResample(pred = svmPred, obs = test$PH))
model.svm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1801 samples
##   32 predictor
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 2 times)
## Summary of sample sizes: 1201, 1201, 1200, 1200, 1201, 1201, ...
## Resampling results across tuning parameters:
##
##   C     RMSE       Rsquared   MAE
##      1  0.1248444  0.4759631  0.09273919
##   1000  0.1515189  0.3805857  0.11270597
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 1.
```

```
caret::varImp(model.svm)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 32)
##
##                   Overall
## Oxygen.Filler      100.00
## Filler.Level        79.70
## Balling             65.91
## Mnf.Flow            62.57
## Filler.Speed        61.87
## Hyd.Pressure3       53.20
## Bowl.Setpoint       52.97
## Hyd.Pressure2       52.40
## Hyd.Pressure1       51.54
## Fill.Pressure       50.10
## Usage.cont          44.60
## Pressure.Setpoint   44.60
```

```
## Carb.Pressure1      43.33
## Balling.Lvl         33.79
## Carb.Rel            33.55
## Density             31.96
## Brand.Code          30.22
## Carb.Flow           27.01
## Temperature         25.59
## PSC                 25.49
```

Support vector machines (SVM) are particularly great at working in higher dimensions and detecting outliers. While we are uncertain if any predictor-response relationships exist in higher dimensions, it would be best practice to include the option. This supervised learning method can specify whether dimensionality should be a greater consideration since we already know outliers are no longer an issue.

```
model.knn <- preProcess(train, "knnImpute")
model.knn <- train(
  PH ~ ., data = train,
  method = "knn",
  center = TRUE,
  scale = TRUE,
  trControl = trainControl("cv", number = 10),
  tuneLength = 25
)
knnPred <- predict(model.knn, newdata = test)
knn_test <- data.frame(
  postResample(pred = knnPred, obs = test$PH))
model.knn
```

```
## k-Nearest Neighbors
##
## 1801 samples
##   32 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1621, 1619, 1622, 1622, 1619, 1621, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    5  0.1365543  0.3847445  0.1007528
##    7  0.1370810  0.3742582  0.1031809
##    9  0.1374473  0.3687643  0.1050557
##   11  0.1372849  0.3667824  0.1054488
##   13  0.1385088  0.3543693  0.1068265
##   15  0.1390987  0.3477235  0.1077093
##   17  0.1405252  0.3345124  0.1090354
##   19  0.1413955  0.3255549  0.1096964
##   21  0.1420692  0.3186610  0.1103664
##   23  0.1424281  0.3148531  0.1107553
```

```
##   25  0.1432065  0.3075185  0.1115795
##   27  0.1437513  0.3024491  0.1121990
##   29  0.1442905  0.2973534  0.1130440
##   31  0.1451483  0.2887273  0.1138900
##   33  0.1455937  0.2843154  0.1142210
##   35  0.1463210  0.2771883  0.1148668
##   37  0.1467295  0.2733518  0.1154371
##   39  0.1471055  0.2697105  0.1158210
##   41  0.1475226  0.2656709  0.1163044
##   43  0.1480806  0.2601238  0.1168119
##   45  0.1485197  0.2560251  0.1172089
##   47  0.1490975  0.2503300  0.1177612
##   49  0.1495323  0.2462947  0.1180793
##   51  0.1499724  0.2417183  0.1184579
##   53  0.1503598  0.2376845  0.1187488
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 5.

caret::varImp(model.knn)

## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 32)
##
##                   Overall
## Oxygen.Filler      100.00
## Filler.Level        79.70
## Balling             65.91
## Mnf.Flow            62.57
## Filler.Speed        61.87
## Hyd.Pressure3       53.20
## Bowl.Setpoint       52.97
## Hyd.Pressure2       52.40
## Hyd.Pressure1       51.54
## Fill.Pressure       50.10
## Usage.cont          44.60
## Pressure.Setpoint   44.60
## Carb.Pressure1      43.33
## Balling.Lvl         33.79
## Carb.Rel            33.55
## Density             31.96
## Brand.Code          30.22
## Carb.Flow           27.01
## Temperature         25.59
## PSC                 25.49
```

For our last nonparametric model, we consider a k-nearest neighbor (KNN). It is one of the simplest algorithms and its main purpose is to classify clusters of sample points into groups for prediction. It is likely not the best model to use in our randomly scattered 'PH'

measurements but there is a chance we are interpreting this data wrong. Having this may prove useful in widening the net of potential models capable of making the best predictions.

## Specialists

This next set of models builds on the most important predictors and models of the previous parametric and nonparametric models. They are repeated model types executed with new, specialized data to each model. Our main takeaway is the down-selection of the data by stepwise regression from the first model. The remaining models added information to confirm our assumptions about predictor importance and provide estimates of which models would perform best. We begin with another KS model.

```
model.ks.sel <- lm(PH~., train.selected)
ksPred.sel <- predict(model.ks.sel, newdata = test.selected)
ks_sel_test <- data.frame(
  postResample(pred = ksPred.sel, obs = test.selected$PH))
summary(model.ks.sel)

##
## Call:
## lm(formula = PH ~ ., data = train.selected)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52630 -0.07659  0.00976  0.08744  0.77580
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        1.061e+01  9.740e-01  10.898  < 2e-16 ***
## Brand.CodeA       -9.272e-03  2.899e-02  -0.320  0.74915
## Brand.CodeB        8.428e-02  1.710e-02   4.928 9.07e-07 ***
## Brand.CodeC       -5.254e-02  1.850e-02  -2.840  0.00456 **
## Brand.CodeD        4.577e-02  3.393e-02   1.349  0.17747
## Carb.Volume       -1.291e-01  5.362e-02  -2.408  0.01615 *
## Fill.Ounces       -7.582e-02  3.823e-02  -1.983  0.04753 *
## PC.Volume         -3.165e-02  6.126e-02  -0.517  0.60543
## Carb.Temp          1.611e-03  8.208e-04   1.962  0.04988 *
## PSC               -1.160e-01  6.954e-02  -1.668  0.09551 .
## PSC.Fill          -3.246e-02  2.764e-02  -1.174  0.24039
## PSC.CO2           -6.483e-02  7.499e-02  -0.865  0.38740
## Mnf.Flow          -7.216e-04  5.511e-05 -13.095  < 2e-16 ***
## Carb.Pressure1     7.085e-03  8.201e-04   8.638  < 2e-16 ***
## Fill.Pressure      3.447e-03  1.396e-03   2.468  0.01366 *
## Hyd.Pressure2     -7.845e-04  5.716e-04  -1.372  0.17011
## Hyd.Pressure3      2.774e-03  6.937e-04   3.998 6.64e-05 ***
## Filler.Level      -9.603e-04  5.794e-04  -1.657  0.09761 .
## Temperature       -1.565e-02  2.630e-03  -5.950 3.23e-09 ***
## Usage.cont        -7.288e-03  1.353e-03  -5.388 8.06e-08 ***
## Carb.Flow          9.548e-06  3.846e-06   2.483  0.01313 *
```

```
## Density           -9.481e-02  3.368e-02  -2.815  0.00493 **
## Balling           -7.698e-02  2.543e-02  -3.027  0.00251 **
## Pressure.Vacuum   -2.256e-02  8.124e-03  -2.777  0.00554 **
## Oxygen.Filler     -3.483e-01  8.205e-02  -4.245 2.30e-05 ***
## Bowl.Setpoint      3.014e-03  6.083e-04   4.955 7.91e-07 ***
## Pressure.Setpoint -9.621e-03  2.300e-03  -4.184 3.01e-05 ***
## Alch.Rel           6.868e-02  2.732e-02   2.513  0.01204 *
## Balling.Lvl        1.178e-01  2.417e-02   4.877 1.18e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1331 on 1772 degrees of freedom
## Multiple R-squared:  0.4103, Adjusted R-squared:  0.401
## F-statistic: 44.03 on 28 and 1772 DF,  p-value: < 2.2e-16

caret::varImp(model.ks.sel)

##                     Overall
## Brand.CodeA        0.3198163
## Brand.CodeB        4.9282884
## Brand.CodeC        2.8402459
## Brand.CodeD        1.3491159
## Carb.Volume        2.4078595
## Fill.Ounces        1.9828942
## PC.Volume          0.5166947
## Carb.Temp          1.9623127
## PSC                1.6678931
## PSC.Fill           1.1744066
## PSC.CO2            0.8645558
## Mnf.Flow          13.0949735
## Carb.Pressure1     8.6384521
## Fill.Pressure      2.4684597
## Hyd.Pressure2      1.3724224
## Hyd.Pressure3      3.9983452
## Filler.Level       1.6574302
## Temperature        5.9495447
## Usage.cont         5.3883338
## Carb.Flow          2.4828455
## Density            2.8151855
## Balling            3.0270473
## Pressure.Vacuum    2.7773353
## Oxygen.Filler      4.2446403
## Bowl.Setpoint      4.9553497
## Pressure.Setpoint  4.1835099
## Alch.Rel           2.5134355
## Balling.Lvl        4.8765171
```

It came as no surprise that similar results occurred akin to that of the original kitchen sink model in the parametric section. However, this model appears to perform slightly better,

having a higher Rsquared value in its summary. We consider the ridge regression once more as well.

```
trctrl <- trainControl(method="repeatedcv",
                       number=3,
                       repeats=2)
model.rr.sel<- caret::train(PH~.,
                            data=train.selected,
                       method="ridge",
                       trControl=trctrl)
rrPred.sel <- predict(model.rr.sel, newdata = test.selected)
rr_sel_test <-data.frame(
  postResample(pred = rrPred.sel,
               obs = test.selected$PH))
model.rr.sel
```

```
## Ridge Regression
##
## 1801 samples
##   25 predictor
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 2 times)
## Summary of sample sizes: 1200, 1201, 1201, 1202, 1199, 1201, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE       Rsquared   MAE
##   0e+00   0.1349880  0.3844382  0.1042221
##   1e-04   0.1349832  0.3844708  0.1042246
##   1e-01   0.1359174  0.3756516  0.1061663
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 1e-04.
```

```
caret::varImp(model.rr.sel)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 25)
##
##                   Overall
## Oxygen.Filler      100.00
## Filler.Level        79.21
## Balling             65.10
## Mnf.Flow            61.68
## Hyd.Pressure3       52.08
## Bowl.Setpoint       51.85
## Hyd.Pressure2       51.27
## Fill.Pressure       48.91
## Usage.cont          43.29
## Pressure.Setpoint   43.28
```

```
## Carb.Pressure1      41.98
## Balling.Lvl         32.21
## Density             30.34
## Brand.Code          28.55
## Carb.Flow           25.27
## Temperature         23.82
## PSC                 23.72
## Pressure.Vacuum     23.36
## Alch.Rel            23.32
## Carb.Volume         22.10
```

Here again, we have made an improvement in the Rsquared term from our previous ridge regression model. This is good news since it likely means a higher RMSE as well. We continue modeling with another partial least squares' regression.

```
model.pls.sel <- train(x = trainx.selected,
                       y = trainy.selected,
                       method = "pls",
                       preProcess =
                         c("center", "scale"),
                       tuneLength = 10)
plsPred.sel <- predict(model.pls.sel, newdata = test.selected)
pls_sel_test <- data.frame(
  postResample(pred = plsPred.sel, obs = test.selected$PH))
model.pls.sel
```

```
## Partial Least Squares
##
## 1801 samples
##   25 predictor
##
## Pre-processing: centered (24), scaled (24), ignore (1)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1801, 1801, 1801, 1801, 1801, 1801, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     0.1531197  0.2109954  0.1215692
##    2     0.1444699  0.2970746  0.1133239
##    3     0.1416260  0.3245609  0.1105196
##    4     0.1388956  0.3503317  0.1082919
##    5     0.1366153  0.3715969  0.1062550
##    6     0.1359202  0.3779680  0.1057347
##    7     0.1355031  0.3818086  0.1052153
##    8     0.1352068  0.3844582  0.1048873
##    9     0.1350818  0.3855997  0.1045372
##   10     0.1351862  0.3848038  0.1046012
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 9.
```

```
caret::varImp(model.pls.sel)
```

```
## pls variable importance
##
##   only 20 most important variables shown (out of 28)
##
##                   Overall
## Mnf.Flow           100.00
## Temperature         73.65
## Usage.cont          61.95
## Hyd.Pressure3       52.93
## Bowl.Setpoint       51.91
## Brand.CodeB         47.66
## Carb.Pressure1      46.76
## Hyd.Pressure2       44.93
## Brand.CodeC         42.38
## Pressure.Setpoint   41.82
## Fill.Pressure       39.94
## Filler.Level        37.69
## Alch.Rel            31.27
## Pressure.Vacuum     30.62
## PSC                 29.91
## Carb.Flow           29.31
## Fill.Ounces         28.51
## Oxygen.Filler       27.01
## Brand.CodeD         15.79
## PSC.Fill            13.40
```

In this model we improve yet again from the original, but worse than the ridge regression and ks models based on Rsquared values. Ideally, we want capture as much variation as possible in the predictions and it is clear through these three models so far that the selected data improve general model performance.

```
ctl <- trainControl(method='repeatedcv',
                        number=10,
                        repeats=3)
mtry <- sqrt(ncol(train.selected))
tunegrid <- expand.grid(.mtry=mtry)
model.rf.sel <- train(PH~.,
                  data=train.selected,
                  method='rf',
                  tuneGrid=tunegrid,
                  preProcess = c("center", "scale"),
                  trControl=ctl)
rfPred.sel <- predict(model.rf.sel, newdata = test.selected)
rf_sel_test <- data.frame(
  postResample(pred = rfPred.sel, obs = test.selected$PH))
model.rf.sel
```

```
## Random Forest
##
## 1801 samples
##   25 predictor
##
## Pre-processing: centered (28), scaled (28)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1621, 1621, 1620, 1621, 1621, 1622, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1074159  0.6335104  0.0798771
##
## Tuning parameter 'mtry' was held constant at a value of 5.09902
```

```
caret::varImp(model.rf.sel)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 28)
##
##                 Overall
## Mnf.Flow         100.00
## Usage.cont        67.47
## Filler.Level      57.55
## Oxygen.Filler     45.93
## Temperature       44.21
## Bowl.Setpoint     42.83
## Balling.Lvl       38.28
## Pressure.Vacuum   37.02
## Carb.Pressure1    32.89
## Balling           32.67
## Brand.CodeC       32.21
## Carb.Flow         32.11
## Alch.Rel          32.11
## Density           27.35
## Fill.Pressure     27.08
## Hyd.Pressure3     25.68
## Carb.Volume       24.64
## PC.Volume         22.88
## Hyd.Pressure2     22.11
## Fill.Ounces       19.33
```

The random forest model appeared to have done best of all models at capturing variations in predictors with the 'PH' response. For testing purposes, we recreated this data set with the selected data although we expect the overall Rsquared for this model to be slightly lower since it is a subset of the original data, and the random forest model is not sensitive to more data but is sensitive to less input.

```
model.cf.sel <- train(PH~.,
                      data=train.selected,
                      method="cforest",
                      trControl=trctrl,
                      preProcess = c("center", "scale"),
                      tuneLength =2)
cfPred.sel <- predict(model.cf.sel, newdata = test.selected)
cf_sel_test <- data.frame(postResample(pred = cfPred.sel, obs = test.selected
$PH))
model.cf.sel
```

```
## Conditional Inference Random Forest
##
## 1801 samples
##   25 predictor
##
## Pre-processing: centered (28), scaled (28)
## Resampling: Cross-Validated (3 fold, repeated 2 times)
## Summary of sample sizes: 1201, 1199, 1202, 1200, 1200, 1202, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##    2    0.1367577  0.4295738  0.10772846
##   28    0.1128230  0.5726778  0.08473082
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 28.
```

```
caret::varImp(model.cf.sel)
```

```
## cforest variable importance
##
##   only 20 most important variables shown (out of 28)
##
##                   Overall
## Mnf.Flow          100.000
## Brand.CodeC        21.612
## Brand.CodeD        18.837
## Usage.cont         16.280
## Pressure.Vacuum    12.246
## Bowl.Setpoint       9.882
## Oxygen.Filler       9.046
## Brand.CodeB         8.326
## Temperature         5.193
## Hyd.Pressure3       4.870
## Carb.Flow           3.842
## Alch.Rel            3.795
## Carb.Pressure1      3.553
## Balling.Lvl         2.841
## Hyd.Pressure2       2.750
```

```
## Fill.Pressure       2.707
## Density             2.652
## Carb.Volume         2.531
## Filler.Level        2.327
## Pressure.Setpoint   2.143
```

A close second, the conditional forest model has an Rsquared just under the traditional random forest model. Unfortunately, both the conditional and random forest models perform worse with the down selected data. This was to be expected but still worth the effort to test it since they are quite close to one another. We finish off with a support vector model which again, did not perform as well as the original. This seems to be the case for all nonparametric models.

```
svm_grid <-  expand.grid(C = c(1,1000))
model.svm.sel <- train(PH~., data = train.selected,
                  method = 'svmRadialCost',
                  trControl = trctrl,
                  tuneGrid = svm_grid)
svmPred.sel <- predict(model.svm.sel, newdata = test.selected)
svm_sel_test <- data.frame(
  postResample(pred = svmPred.sel, obs = test.selected$PH))
model.svm.sel
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1801 samples
##   25 predictor
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 2 times)
## Summary of sample sizes: 1201, 1201, 1200, 1202, 1199, 1201, ...
## Resampling results across tuning parameters:
##
##   C     RMSE       Rsquared   MAE
##      1  0.1258353  0.4687413  0.09362054
##   1000  0.1708021  0.3035524  0.12625596
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 1.
```

```
caret::varImp(model.svm.sel)
```

```
## loess r-squared variable importance
##
##   only 20 most important variables shown (out of 25)
##
##                   Overall
## Oxygen.Filler      100.00
## Filler.Level        79.21
## Balling             65.10
```

```
## Mnf.Flow            61.68
## Hyd.Pressure3       52.08
## Bowl.Setpoint       51.85
## Hyd.Pressure2       51.27
## Fill.Pressure       48.91
## Usage.cont          43.29
## Pressure.Setpoint   43.28
## Carb.Pressure1      41.98
## Balling.Lvl         32.21
## Density             30.34
## Brand.Code          28.55
## Carb.Flow           25.27
## Temperature         23.82
## PSC                 23.72
## Pressure.Vacuum     23.36
## Alch.Rel            23.32
## Carb.Volume         22.10
```

## Model Selection

Before selecting the best model, we decide on which summary statistics will produce the most accurate results in a real-world setting. We also review the importance of several predictors, make predictions on our test set, and visualize the results.

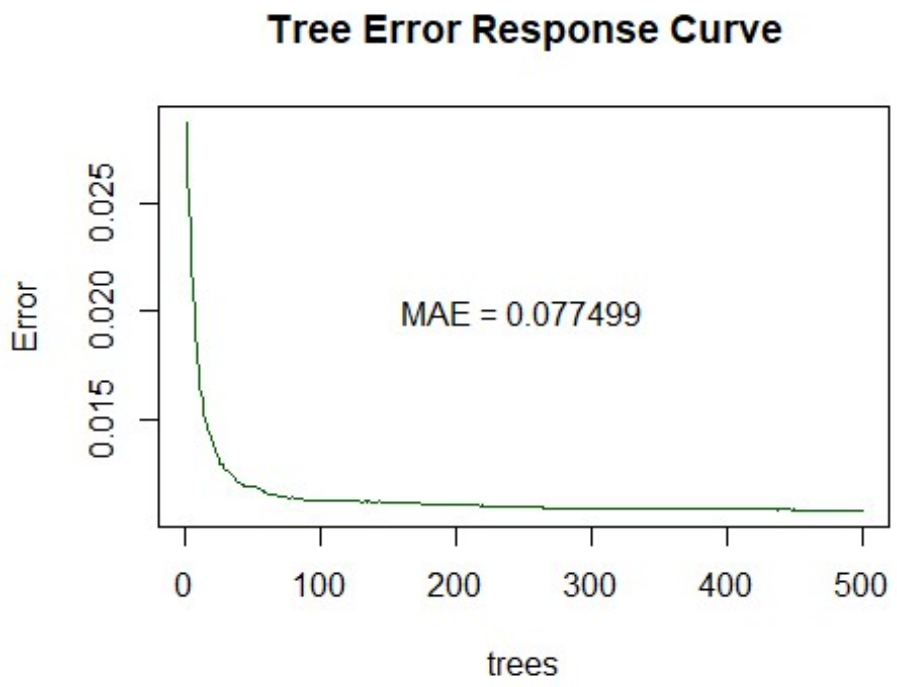
### Evaluation Criteria

We focus on three main metrics to evaluate the models. They are the root mean squared error (RMSE), the coefficient of determination (R squared), and mean absolute error (MAE) in each model. Of these, MAE is the most important to us in evaluation. This is because it finds the absolute difference in forecast values from actual values. For our simulated training and testing data, this should prove most useful in identifying the most accurate model. However, if other metrics (RMSE and R squared) are particularly strong or weak, then slightly lower MAE may mean less overall. We consider these metrics holistically to get the full picture.

### Selection and Prediction

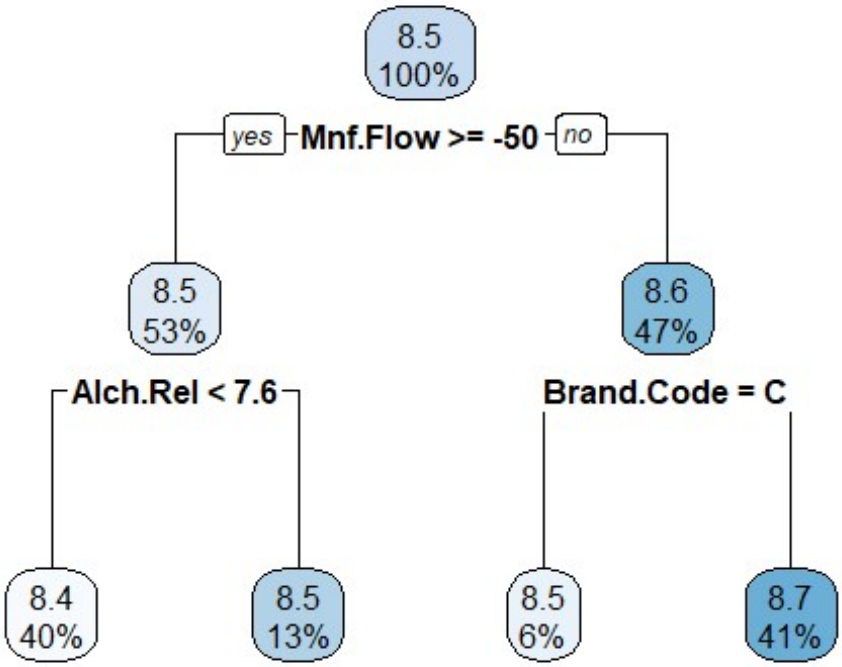
Predictions are made with the test set that contains 30% of the original data. The function postResample() from the caret package was used during model creation and evaluation to assess each model when building them. This led us to assume that the random forest would perform best. This plot of the tree-error response curve demonstrates why.

```
rf.MAE <- round(model.rf$results$MAE, 6)
plot(model.rf$finalModel, main="Tree Error Response Curve", col = "darkgreen"
)
text(xy.coords(250, 0.02),
     labels = paste("MAE =",rf.MAE))
```




As the number of trees increase the errors made by the model decreased on an exponential scale. Although we are aware there are variations in the dark green error curve, they are hardly visible because of how small the changes are. It took less than 50 trees to produce an error response below 0.012 and it steadily moved closer to zero with additional trees.

```
decisiontree <- train(x = trainx,
                      y = trainy,
                      method = "rpart",
                      tuneLength = 7,
                      control = rpart.control(maxdepth=2))
rpart.plot(decisiontree$finalModel)
```




Decision trees dendrograms can also provide meaningful information about the final model's predictors selection process. Our model contains 500 trees and produces highly accurate results within the first 50. While it would be great to know what this tree looked like, it would not be visible on this page. Instead, we create an exemplary decision tree of the same with only the first few decisions.

In this example we visualize to comprehend how our random forest model functionally makes decisions. Each bubble is a node corresponding to a predictor. Nodes are arranged in descending order of importance. In this case the most important predictor happens to be 'Mnf.Flow' which is why it is placed at the top of the model. The model also computes selector criterion at each node to descend the predictors in order of importance. The ideal quantity of decisions are then quantified by the model to minimize errors. This dendrogram only contains 7 nodes. Our final random forest model does not have the same nodes but shares the same process.

After making our judgement on the random forest model we then intentionally recreated new models on down selected data to in six separate attempts improve model MAE. Our intent was to beat it. However, the original nonparametric random forest model remained the primary source of prediction due to it having the lowest MAE and second RMSE and Rsquared. It was used to generate our predictions.

## Conclusion

After cleaning approximately 2,571 observations in each of the 33 variables of this food and beverage manufacturing company, 17 models were generated to find that the best model was the nonparametric random forest. It has the lowest MAE which we favorited as the best predictor since it quantifies error of the model by directly measuring the absolute difference between expected and actual observations. It also had the second highest RMSE and Rsquared values of all models.

### Model Results

The results of all 17 models are summarized by RMSE, Rsquared, and MAE in the follow table. Model names are abbreviated for quick referencing and any model with an " * " next to it is a specialized form of that model.

```
model.results <- data.frame(t(pm_test)) %>%
  mutate("Model" = "PM") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Stepwise Model Results
model.results.pm <- data.frame(t(pm_test)) %>%
  mutate("Model" = "STEP") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Kitchen Sink Model Results
model.results.ks <- data.frame(t(ks_test)) %>%
  mutate("Model" = "KS") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Ridge Regression Model Results
model.results.rr <- data.frame(t(rr_test)) %>%
  mutate("Model" = "RR") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Boosted Generalized Linear Model Results
model.results.glmb <- data.frame(t(glmb_test)) %>%
  mutate("Model" = "GLMB") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Partial Least Squares Model Results
model.results.pls <- data.frame(t(pls_test)) %>%
  mutate("Model" = "PLS") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Multivariate Adaptive Splines Model Results
model.results.mar <- data.frame(t(mar_test)) %>%
  mutate("Model" = "MAR") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Random Forest Model Results
model.results.rf <- data.frame(t(rf_test)) %>%
  mutate("Model" = "RF") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Neural Net Model Results
model.results.nnet <- data.frame(t(nnet_test)) %>%
  mutate("Model" = "NNET") %>%
```

```
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Conditional Forest Model Results
model.results.cf <- data.frame(t(cf_test)) %>%
  mutate("Model" = "CF") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Support Vector Machine Model Results
model.results.svm <- data.frame(t(svm_test)) %>%
  mutate("Model" = "SVM") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# K-Nearest Neighbor Model Results
model.results.knn <- data.frame(t(knn_test)) %>%
  mutate("Model" = "KNN") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Specialized Kitchen Sink Model Results
model.results.ks.sel <- data.frame(
  t(ks_sel_test)) %>%
  mutate("Model" = "KS*") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Specialized Ridge Regression Model Results
model.results.rr.sel <- data.frame(
  t(rr_sel_test)) %>%
  mutate("Model" = "RR*") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Specialized Ridge Regression Model Results
model.results.pls.sel <- data.frame(
  t(pls_sel_test)) %>%
  mutate("Model" = "PLS*") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Specialized Random Forest Model Results
model.results.rf.sel <- data.frame(
  t(rf_sel_test)) %>%
  mutate("Model" = "RF*") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Specialized Conditional Forest Model Results
model.results.cf.sel <- data.frame(
  t(cf_sel_test)) %>%
  mutate("Model" = "CF*") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Specialized Support Vector Model Results
model.results.svm.sel <- data.frame(
  t(svm_sel_test)) %>%
  mutate("Model" = "SVM*") %>%
  dplyr::select(Model, RMSE, Rsquared, MAE)
# Combined Tabulated Model Results
model.results <- rbind(
  model.results.pm,
  model.results.ks,
  model.results.rr,
  model.results.glmb,
  model.results.pls,
```

```
  model.results.mar,
  model.results.rf,
  model.results.nnet,
  model.results.cf,
  model.results.svm,
  model.results.knn,
  model.results.ks.sel,
  model.results.rr.sel,
  model.results.pls.sel,
  model.results.rf.sel,
  model.results.cf.sel,
  model.results.svm.sel
)
model.results <- model.results %>%
  dplyr::mutate(across(where(is.numeric),
                       round, 3)) %>%
  dplyr::arrange(MAE) %>%
  dplyr::mutate(Rank = row_number()) %>%
  dplyr::select(Rank, Model, RMSE, Rsquared, MAE)
flextable(model.results) %>%
  theme_vanilla() %>%
  set_table_properties(layout = "autofit")
```

| Rank | Model | RMSE | Rsquared | MAE |
|---|---|---|---|---|
| 1 | RF | 0.101 | 0.697 | 0.075 |
| 2 | CF | 0.100 | 0.675 | 0.075 |
| 3 | RF* | 0.103 | 0.679 | 0.077 |
| 4 | CF* | 0.103 | 0.657 | 0.078 |
| 5 | SVM | 0.116 | 0.558 | 0.084 |
| 6 | SVM* | 0.117 | 0.545 | 0.086 |
| 7 | NNET | 0.147 | 0.335 | 0.099 |
| 8 | KNN | 0.136 | 0.393 | 0.100 |
| 9 | MAR | 0.130 | 0.438 | 0.101 |
| 10 | STEP | 0.133 | 0.415 | 0.103 |
| 11 | KS | 0.133 | 0.415 | 0.103 |
| 12 | RR | 0.133 | 0.415 | 0.103 |
| 13 | KS* | 0.132 | 0.418 | 0.103 |
| 14 | RR* | 0.132 | 0.418 | 0.103 |

| Rank | Model | RMSE | Rsquared | MAE |
|---|---|---|---|---|
| 15 | PLS* | 0.132 | 0.417 | 0.103 |
| 16 | GLMB | 0.133 | 0.412 | 0.104 |
| 17 | PLS | 0.133 | 0.412 | 0.104 |

## Discussion

When reviewing the most important predictors to each model but especially the winning random forest model, there is a discrepancy. For example, a table has been compiled with the first 9 predictors from the most important stepwise AIC regression selection. They are ordered.

```
# Calculate Importance by Predictor
# Rearrange in descending order
model.imps <- data.frame()
pm.imps <- caret::varImp(model.pm) %>%
  dplyr::arrange(desc(Overall))
ks.imps <- caret::varImp(model.ks) %>%
  dplyr::arrange(desc(Overall))
rr.imps <- caret::varImp(model.rr)
glmb.imps <- caret::varImp(model.glmb)
pls.imps <- caret::varImp(model.pls)
mar.imps <- caret::varImp(model.mar)
rf.imps <- caret::varImp(model.rf)
nnet.imps <- caret::varImp(model.nnet)
cf.imps <- caret::varImp(model.cf)
svm.imps <- caret::varImp(model.svm)
knn.imps <- caret::varImp(model.knn)
ks.sel.imps <- caret::varImp(model.ks.sel) %>%
  dplyr::arrange(desc(Overall))
rr.sel.imps <- caret::varImp(model.rr.sel)
pls.sel.imps <- caret::varImp(model.pls.sel)
rf.sel.imps <- caret::varImp(model.rf.sel)
cf.sel.imps <- caret::varImp(model.cf.sel)
svm.sel.imps <- caret::varImp(model.svm.sel)
# Combine and Display Importance by Model
model.imps <- pm.imps
model.imps <- model.imps %>%
  data.frame() %>%
  tibble::rownames_to_column("Predictor") %>%
  dplyr::rename(STEP = Overall) %>%
  dplyr::arrange(desc(STEP)) %>%
  dplyr::slice_head(n = 9) %>%
  cbind(ks.imps[1:9,]) %>%
  cbind(rr.imps$importance[1:9,]) %>%
  cbind(glmb.imps$importance[1:9,]) %>%
```

```
  cbind(pls.imps$importance[1:9,]) %>%
  cbind(mar.imps$importance[1:9,]) %>%
  cbind(rf.imps$importance[1:9,]) %>%
  cbind(nnet.imps$importance[1:9,]) %>%
  cbind(cf.imps$importance[1:9,]) %>%
  cbind(svm.imps$importance[1:9,]) %>%
  cbind(knn.imps$importance[1:9,]) %>%
  cbind(ks.sel.imps[1:9,]) %>%
  cbind(rr.sel.imps$importance[1:9,]) %>%
  cbind(pls.sel.imps$importance[1:9,]) %>%
  cbind(rf.sel.imps$importance[1:9,]) %>%
  cbind(cf.sel.imps$importance[1:9,]) %>%
  cbind(svm.sel.imps$importance[1:9,]) %>%
  dplyr::rename(KS='ks.imps[1:9, ]',
      RR='rr.imps$importance[1:9, ]',
      GLMB='glmb.imps$importance[1:9, ]',
      PLS='pls.imps$importance[1:9, ]',
      MARS='mar.imps$importance[1:9, ]',
      RF='rf.imps$importance[1:9, ]',
      NNET='nnet.imps$importance[1:9, ]',
      CF='cf.imps$importance[1:9, ]',
      SVM='svm.imps$importance[1:9, ]',
      KNN='knn.imps$importance[1:9, ]',
      'KS*'='ks.sel.imps[1:9, ]',
      'RR*'='rr.sel.imps$importance[1:9, ]',
      'PLS*'='pls.sel.imps$importance[1:9, ]',
      'RF*'='rf.sel.imps$importance[1:9, ]',
      'CF*'='cf.sel.imps$importance[1:9, ]',
      'SVM*'='svm.sel.imps$importance[1:9, ]',
)
model.imps <- model.imps %>%
  t() %>%
  data.frame()
names(model.imps) <- as.matrix(model.imps[1,])
model.imps <- model.imps[-1,]
model.imps <- model.imps %>%
  tibble::rownames_to_column("Model")
flextable::flextable(model.imps) %>%
  theme_vanilla() %>%
  set_table_properties(layout = "autofit")
```

| Model | Mnf.Flow | Carb.Pressure1 | Temperature | Usage.cont | Brand.CodeB | Bowl.Setpoint | Balling.Lvl | Oxygen.Filler | Hyd.Pressure3 |
|---|---|---|---|---|---|---|---|---|---|
| STEP | 12.905521 | 8.461446 | 5.918955 | 5.283926 | 5.064469 | 4.635294 | 4.582208 | 4.272265 | 4.156739 |
| KS | 12.905521 | 8.461446 | 5.918955 | 5.283926 | 5.064469 | 4.635294 | 4.582208 | 4.272265 | 4.156739 |
| RR | 30.215924 | 23.912472 | 15.910720 | 2.324998 | 16.656413 | 16.965786 | 25.492481 | 7.920395 | 2.922509 |
| GLMB | 26.196829 | 36.936978 | 100.000000 | 27.377351 | 4.866748 | 6.770186 | 4.878596 | 5.661394 | 1.605650 |
| PLS | 10.739848 | 48.026903 | 39.625361 | 11.769748 | 13.390892 | 27.542211 | 8.344082 | 8.383881 | 7.859082 |
| MARS | 100.000000 | 61.415463 | 38.055402 | 37.324107 | 37.324107 | 22.913963 | 17.471943 | 8.656008 | 0.000000 |
| RF | 0.000000 | 8.269060 | 30.007180 | 3.492107 | 16.916638 | 13.279216 | 15.695956 | 9.584583 | 9.118037 |
| NNET | 30.215924 | 23.912472 | 15.910720 | 2.324998 | 16.656413 | 16.965786 | 25.492481 | 7.920395 | 2.922509 |
| CF | 1.48717953 | 7.61232635 | 20.87706950 | 18.34570554 | 2.27495376 | 0.38726487 | 0.72451081 | 0.09792447 | 0.04845622 |
| SVM | 30.215924 | 23.912472 | 15.910720 | 2.324998 | 16.656413 | 16.965786 | 25.492481 | 7.920395 | 2.922509 |
| KNN | 30.215924 | 23.912472 | 15.910720 | 2.324998 | 16.656413 | 16.965786 | 25.492481 | 7.920395 | 2.922509 |
| KS* | 13.094974 | 8.638452 | 5.949545 | 5.388334 | 4.955350 | 4.928288 | 4.876517 | 4.244640 | 4.183510 |
| RR* | 28.5548257 | 22.1013297 | 13.9091083 | 0.0000000 | 14.9892893 | 23.7189483 | 5.7285864 | 0.6117337 | 61.6770967 |
| PLS* | 11.478586 | 47.664766 | 42.382832 | 15.788191 | 1.201595 | 28.508944 | 5.517669 | 9.157070 | 29.914377 |
| RF* | 0.000000 | 9.926528 | 32.213397 | 3.757714 | 24.640449 | 19.327192 | 22.882223 | 14.546633 | 18.543651 |
| CF* | 1.5770554 | 8.2968753 | 22.0349855 | 19.0589877 | 2.3425089 | 0.6573852 | 0.8267325 | 0.0000000 | 0.1036405 |
| SVM* | 28.5548257 | 22.1013297 | 13.9091083 | 0.0000000 | 14.9892893 | 23.7189483 | 5.7285864 | 0.6117337 | 61.6770967 |

The predictor deemed most important in our stepwise regression ‘Mnf.Flow’ turned out to have no importance in the random forest. In fact, the only variable that showed promise from our selection in the random forest model was ‘Temperature’ with an importance value of 28.6. This may be caused by the formula unique to the model type since MARS, GLMB, and SVM share a similar adaptive regression capability. Future studies should explore this further to exploit the benefits of specific predictor-response relationships if there are any to be found.

Lastly, in future studies it would also be wise to consider alternative models, reduce the outlier leniency by shrinking the IQR multiplier, and perhaps create new features from existing predictors. While our model provides an impressive amount of accuracy it could be improved. New features could include ratios of pressure and temperature, for example, since they are known to have a clear linear relationship in nature. Adjustments such as these would also provide deeper insight into the eb and flow pattern of the randomly distributed ‘PH’ measurements in the manufacturing process, which, is the main challenge we face when making predictions.