DATA 624 PREDICTIVE ANALYTICS - Project 2

Melissa Bowman, Frederick Jones, Shoshana Farber, Gabriel Campos

Last edited May 05, 2024

Libraries

```
library(Amelia)
library(car)
library(caret)
library(corrplot)
library(Cubist)
library(DataExplorer)
library(dplyr)
library(e1071)
library(earth)
library(forcats)
library(forecast)
library(fpp3)
library(gbm)
library(ggplot2)
library(kableExtra)
library(MASS)
library(mice)
library(mlbench)
library(party)
library(randomForest)
library(RANN)
library(RColorBrewer)
library(readxl)
library(rpart)
library(rpart.plot)
library(summarytools)
library(tidyr)
library(VIM)
library(earth)
library(randomForest)
```

Assignment Description

Project #2 (Team) Assignment

This is role playing. I am your new boss. I am in charge of production at ABC Beverage and you are a team of data scientists reporting to me. My leadership has told me that new regulations are requiring us to understand our manufacturing process, the predictive factors and be able to report to them our predictive model of pH.

Please use the historical data set I am providing. Build and report the factors in BOTH a technical and non-technical report. I like to use Word and Excel. Please provide your non-technical report in a business friendly readable document and your predictions in an Excel readable format. The technical report should show clearly the models you tested and how you selected your final approach. Please submit both Rpubs links and .rmd files or other readable formats for technical and non-technical reports. Also submit the excel file showing the prediction of your models for pH.

Data Import

We will first load in the data that is required for this analysis.

```
train_df <- readxl::read_xlsx('Data/StudentData.xlsx')
test_df <- readxl::read_xlsx('Data/StudentData.xlsx')
```

`StudentData.xlsx` is our Training data set.

`StudentEvaluation.xlsx` is our Test data set.

Exporatory Data Analysis

First, we can preview our dataset.

```
glimpse(train_df)
```

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

The dataset consists of 2,571 rows and 33 columns. Most of the variables are numeric, except for the first column indicating `Brand Code`. Our response variable is `PH`.

We can take also take a look at the summary statistics for each of the numeric variables.

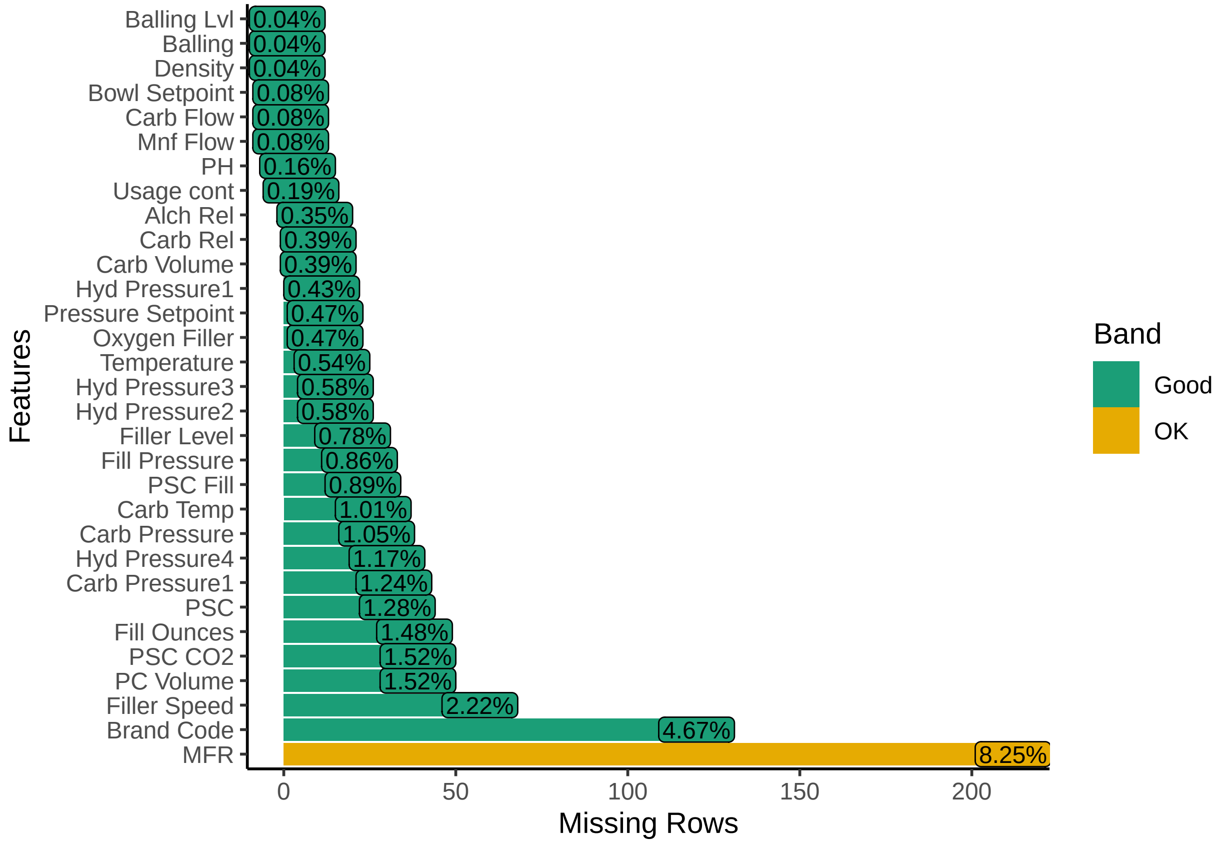
```
summary(train_df)
```

```
##   Brand Code         Carb Volume     Fill Ounces      PC Volume
##  Length:2571        Min.   :5.040   Min.   :23.63   Min.   :0.07933
##  Class :character   1st Qu.:5.293   1st Qu.:23.92   1st Qu.:0.23917
##  Mode  :character   Median :5.347   Median :23.97   Median :0.27133
##                     Mean   :5.370   Mean   :23.97   Mean   :0.27712
##                     3rd Qu.:5.453   3rd Qu.:24.03   3rd Qu.:0.31200
##                     Max.   :5.700   Max.   :24.32   Max.   :0.47800
##                     NA's   :10      NA's   :38      NA's   :39
##  Carb Pressure     Carb Temp          PSC             PSC Fill
##  Min.   :57.00   Min.   :128.6   Min.   :0.00200   Min.   :0.0000
##  1st Qu.:65.60   1st Qu.:138.4   1st Qu.:0.04800   1st Qu.:0.1000
##  Median :68.20   Median :140.8   Median :0.07600   Median :0.1800
##  Mean   :68.19   Mean   :141.1   Mean   :0.08457   Mean   :0.1954
##  3rd Qu.:70.60   3rd Qu.:143.8   3rd Qu.:0.11200   3rd Qu.:0.2600
##  Max.   :79.40   Max.   :154.0   Max.   :0.27000   Max.   :0.6200
##  NA's   :27      NA's   :26      NA's   :33        NA's   :23
##     PSC CO2           Mnf Flow       Carb Pressure1  Fill Pressure
##  Min.   :0.00000   Min.   :-100.20   Min.   :105.6   Min.   :34.60
##  1st Qu.:0.02000   1st Qu.:-100.00   1st Qu.:119.0   1st Qu.:46.00
##  Median :0.04000   Median :  65.20   Median :123.2   Median :46.40
##  Mean   :0.05641   Mean   :  24.57   Mean   :122.6   Mean   :47.92
##  3rd Qu.:0.08000   3rd Qu.: 140.80   3rd Qu.:125.4   3rd Qu.:50.00
##  Max.   :0.24000   Max.   : 229.40   Max.   :140.2   Max.   :60.40
##  NA's   :39        NA's   :2         NA's   :32      NA's   :22
##  Hyd Pressure1   Hyd Pressure2   Hyd Pressure3   Hyd Pressure4
##  Min.   :-0.80   Min.   : 0.00   Min.   :-1.20   Min.   : 52.00
##  1st Qu.: 0.00   1st Qu.: 0.00   1st Qu.: 0.00   1st Qu.: 86.00
##  Median :11.40   Median :28.60   Median :27.60   Median : 96.00
##  Mean   :12.44   Mean   :20.96   Mean   :20.46   Mean   : 96.29
##  3rd Qu.:20.20   3rd Qu.:34.60   3rd Qu.:33.40   3rd Qu.:102.00
##  Max.   :58.00   Max.   :59.40   Max.   :50.00   Max.   :142.00
##  NA's   :11      NA's   :15      NA's   :15      NA's   :30
##   Filler Level    Filler Speed   Temperature      Usage cont      Carb Flow
##  Min.   : 55.8   Min.   : 998   Min.   :63.60   Min.   :12.08   Min.   :  26
##  1st Qu.: 98.3   1st Qu.:3888   1st Qu.:65.20   1st Qu.:18.36   1st Qu.:1144
##  Median :118.4   Median :3982   Median :65.60   Median :21.79   Median :3028
##  Mean   :109.3   Mean   :3687   Mean   :65.97   Mean   :20.99   Mean   :2468
##  3rd Qu.:120.0   3rd Qu.:3998   3rd Qu.:66.40   3rd Qu.:23.75   3rd Qu.:3186
##  Max.   :161.2   Max.   :4030   Max.   :76.20   Max.   :25.90   Max.   :5104
##  NA's   :20      NA's   :57     NA's   :14      NA's   :5       NA's   :2
##     Density           MFR           Balling       Pressure Vacuum
##  Min.   :0.240   Min.   : 31.4   Min.   :-0.170   Min.   :-6.600
##  1st Qu.:0.900   1st Qu.:706.3   1st Qu.: 1.496   1st Qu.:-5.600
##  Median :0.980   Median :724.0   Median : 1.648   Median :-5.400
##  Mean   :1.174   Mean   :704.0   Mean   : 2.198   Mean   :-5.216
##  3rd Qu.:1.620   3rd Qu.:731.0   3rd Qu.: 3.292   3rd Qu.:-5.000
##  Max.   :1.920   Max.   :868.6   Max.   : 4.012   Max.   :-3.600
##  NA's   :1       NA's   :212     NA's   :1
##        PH        Oxygen Filler     Bowl Setpoint   Pressure Setpoint
##  Min.   :7.880   Min.   :0.00240   Min.   : 70.0   Min.   :44.00
##  1st Qu.:8.440   1st Qu.:0.02200   1st Qu.:100.0   1st Qu.:46.00
##  Median :8.540   Median :0.03340   Median :120.0   Median :46.00
##  Mean   :8.546   Mean   :0.04684   Mean   :109.3   Mean   :47.62
##  3rd Qu.:8.680   3rd Qu.:0.06000   3rd Qu.:120.0   3rd Qu.:50.00
##  Max.   :9.360   Max.   :0.40000   Max.   :140.0   Max.   :52.00
##  NA's   :4       NA's   :12        NA's   :2       NA's   :12
##  Air Pressurer      Alch Rel        Carb Rel      Balling Lvl
##  Min.   :140.8   Min.   :5.280   Min.   :4.960   Min.   :0.00
##  1st Qu.:142.2   1st Qu.:6.540   1st Qu.:5.340   1st Qu.:1.38
##  Median :142.6   Median :6.560   Median :5.400   Median :1.48
##  Mean   :142.8   Mean   :6.897   Mean   :5.437   Mean   :2.05
##  3rd Qu.:143.0   3rd Qu.:7.240   3rd Qu.:5.540   3rd Qu.:3.14
##  Max.   :148.2   Max.   :8.620   Max.   :6.060   Max.   :3.66
##                  NA's   :9       NA's   :10      NA's   :1
```

NA Proportions

We can plot the missing values for each column to see what proportion of each variable is missing.

```r
plot_missing(train_df,
             missing_only = T,
             ggtheme = theme_classic(),
             theme_config = list(legend.position = c("right")),
             geom_label_args = list("size" = 3, "label.padding" = unit(0.1, "lines")))
```



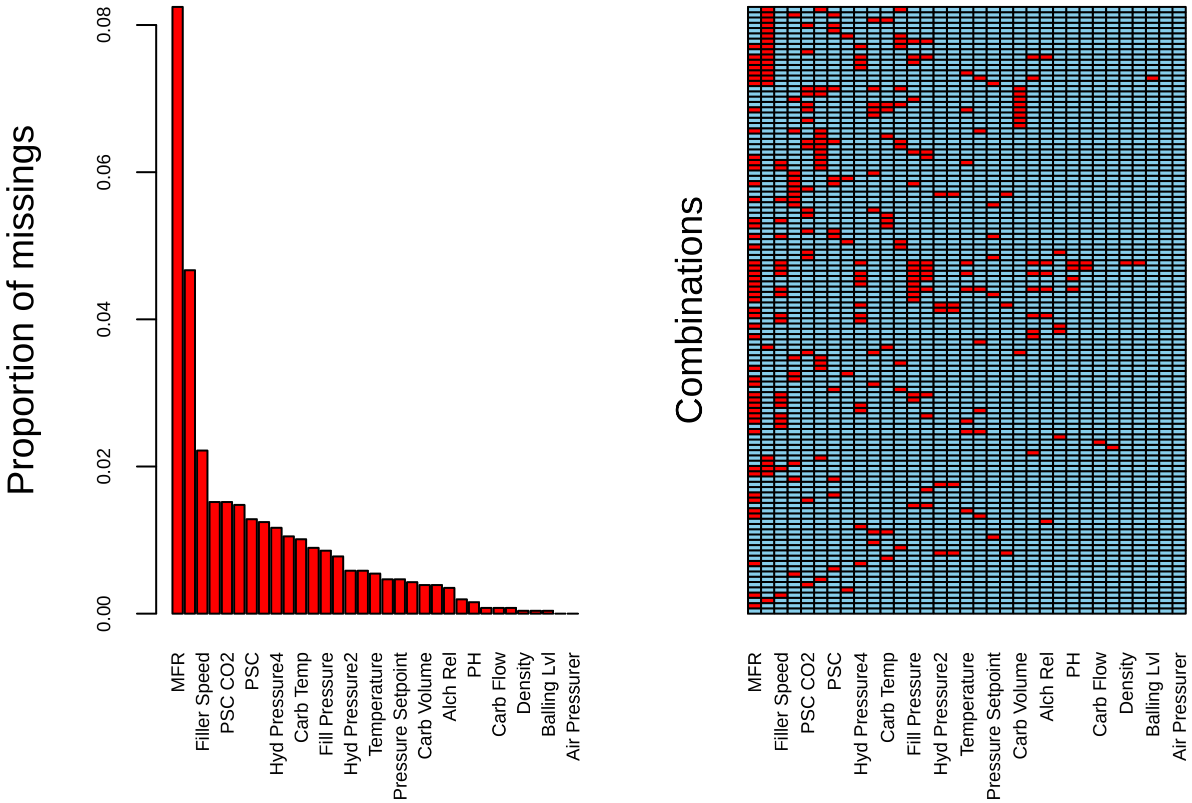
We can see that majority of the variables are missing less than 1% of values. For those that are missing more than 1% of the data, majority still fall below 5%. The variable with the most missing data, and possibly cause for concern, is `MFR`. However, even this is missing only about 8.25% of the data.

```
data.frame(missing = colSums(is.na(train_df))) |>
  filter(missing == 0) |>
  rownames()
```

```
## [1] "Pressure Vacuum" "Air Pressurer"
```

`Pressure Vacuum` and `Air Pressurer` are the only variables not missing any data.

```
VIM::aggr(train_df, numbers=T, sortVars=T, bars = FALSE,
          cex.axis = .6)
```

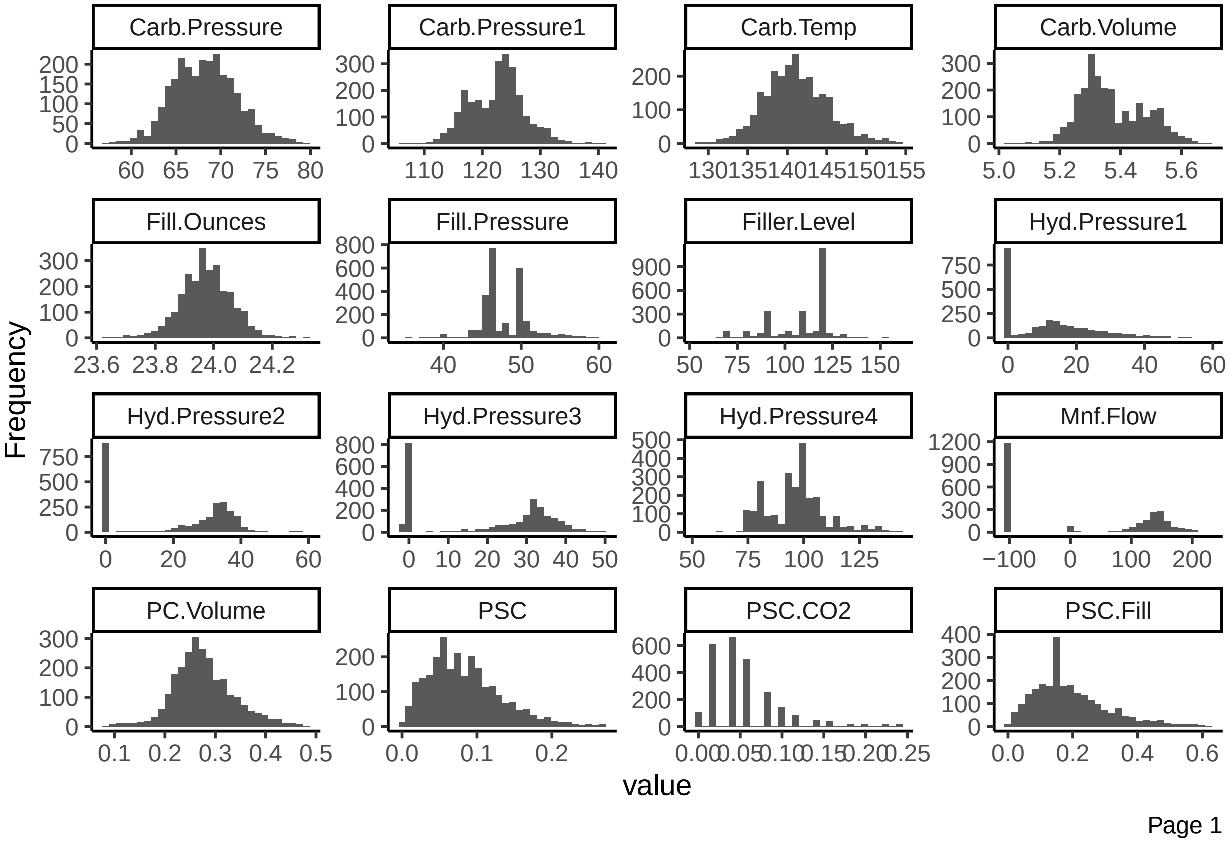


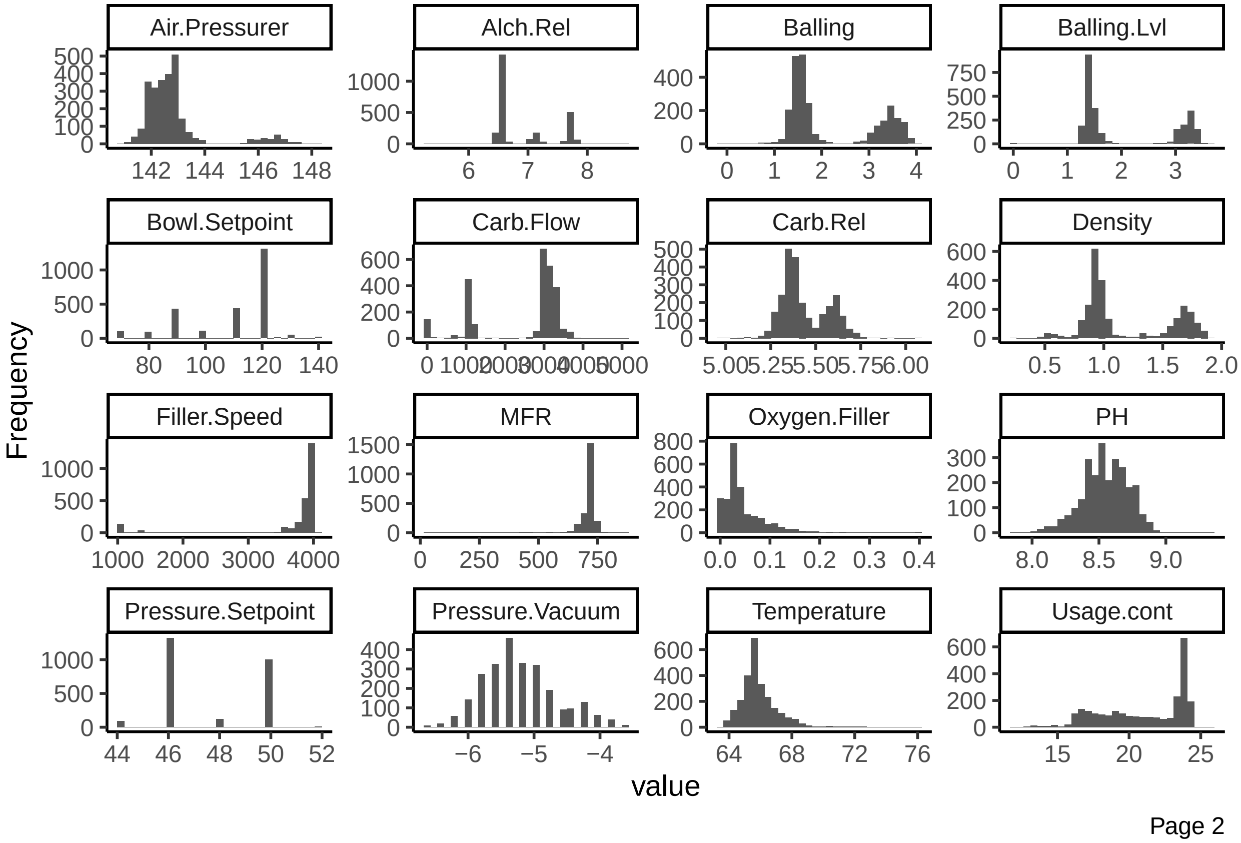
```
##
##  Variables sorted by number of missings:
##           Variable        Count
##                MFR 0.0824581875
##         Brand Code 0.0466744457
##       Filler Speed 0.0221703617
##          PC Volume 0.0151691949
##            PSC CO2 0.0151691949
##        Fill Ounces 0.0147802412
##                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
##    Pressure Vacuum 0.0000000000
##      Air Pressurer 0.0000000000
```

Distributions

We will now take a look at the distributions of the numeric variables.

```
DataExplorer::plot_histogram(train_df, nrow = 4L, ncol = 4L, ggtheme = theme_classic())
```





`Carb Pressure`, `Carb Temp`, `Fill Ounces`, `PC Volume`, and `PH` seem to be relatively normally distributed.

`Hyd Pressure 1`, `PCS`, `PSC CO2`, `PSC Fill`, `Air Pressurer`, `Oxygen Filler`,`Pressure Vacuum`, and `Temperature` all seem to have a right skew.

`Hyd Pressure2`, `Hyd Pressure3`, and `Mnf Flow` all seem to have a left skew, although there are also a fair amount of entries with a value at 0. `Filler Speed` and `MFR` also seem to have a left skew.

Some variables, such as `Balling`, `Balling Lvl`, `Carb Rel`, and `Density` seem to be bimodally distributed.

Initial Findings

- Data consists of 2571 observations with 33 columns
- `Brand Code`:
  - Type character
  - Unordered categorical values
- Predictors:
  - Primarily doubles
  - 4 can be considered integers
  - High range variables:
    i. `Mnf Flow` -100.20 to 220.40
    ii. `Hyd Pressure1` -50.00 to 50.00
    iii. `Hyd Pressure2` -50.00 to 61.40
    iv. `Hyd Pressure3` -50.00 to 49.20
    v. `Hyd Pressure4` 68.00 to 140.00

- About 8% of the values for `MFR` is missing.

- `Brand Code` is missing about 5%
- Filler Speed is missing about 2%
- Remaining Variables have roughly 1% or less missing.

- `Pressure.Vacuum`, `Air.Pressurer` have no NAs
- The Distribution of the variables can be grouped as **left skewed**, **right skewed** and for symmetric we can categorized as **relatively normal**
  - Relatively Normal Distributions:
    - `Carb.Pressure`
    - `Carb.Temp` -`Fill.Ounces`
    - `PC.Volume`
    - `PH`
  - Left-skew Distributions:
    - `Carb.Flow`
    - `Filler.Speed`
    - `Mnf.Flow`
    - `MFR`
    - `Bowl.Setpoint`
    - `Filler.Level`
    - `Hyd.Pressure2`
    - `Hyd.Pressure3` -`Usage.cont`
    - `Carb.Pressure1`
    - `Filler.Speed`
  - Right-skew Distributions:
    - `Pressure.Setpoint`
    - `Fill.Pressure`
    - `Hyd.Pressure1`
    - `Temperature`
    - `Carb.Volume`
    - `PSC`
    - `PSC.CO2`
    - `PSC.Fill`
    - `Balling`
    - `Density`
    - `Hyd.Pressure4`
    - `Air.Pressurer`
    - `Alch.Rel`
    - `Carb.Rel`
    - `Oxygen.Filler`
    - `Balling.Lvl`
    - `Pressure.Vacuum`

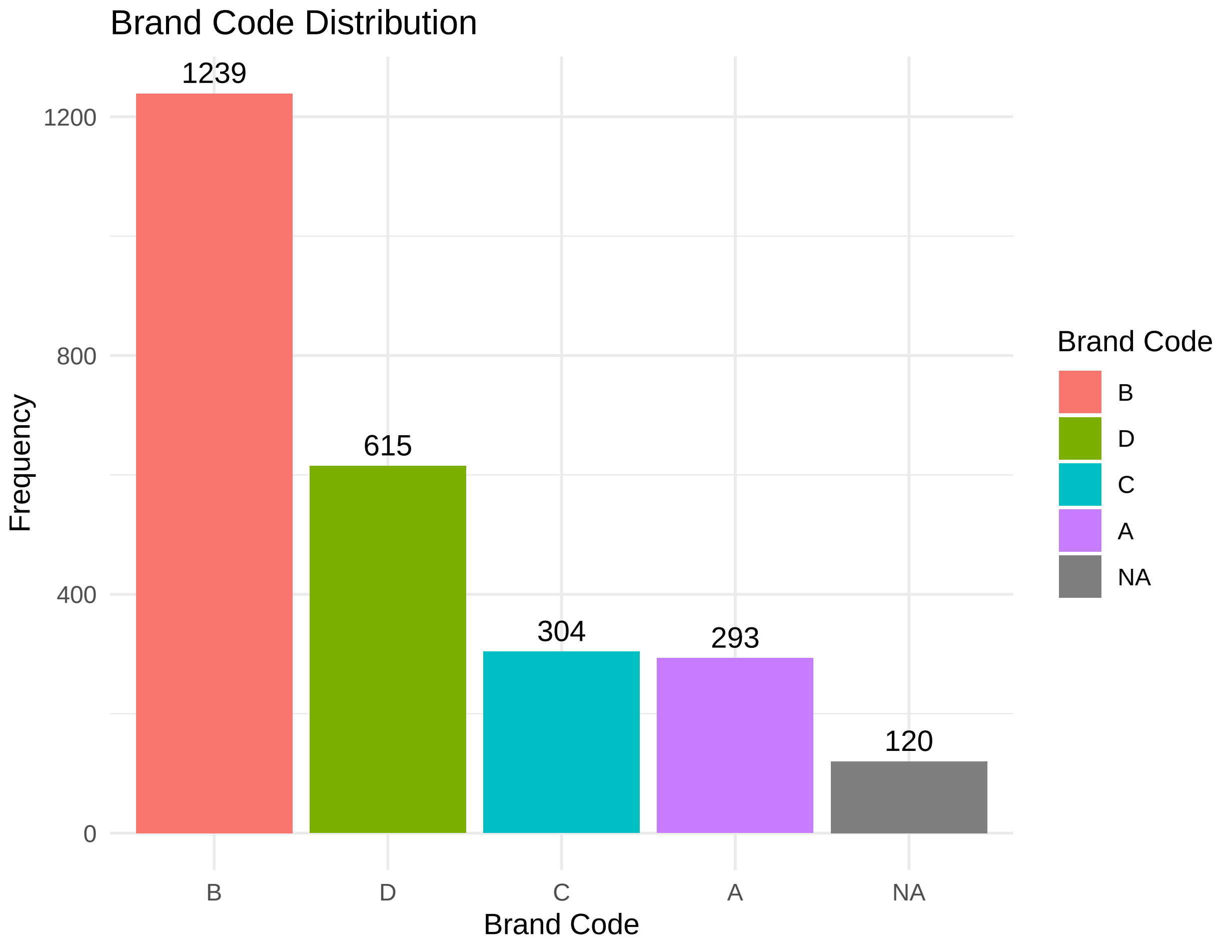
```r
unique(train_df$`Brand Code`)
```

```
## [1] "B" "A" "C" "D" NA
```

Brand Code Distribution

`Brand Code` has 4 categorical values outside of NA (**A,B,C,D**). Let's examine the distribution of these codes.

```
train_df |>
  mutate(`Brand Code` = factor(`Brand Code`, levels = names(sort(table(`Brand Code`), decreasing = TRUE
  ggplot(aes(x = `Brand Code`, fill = `Brand Code`)) +
  geom_bar(stat = "count") +
  geom_text(stat = 'count', aes(label = ..count..), vjust = -0.5, color = "black") +
  labs(title = 'Brand Code Distribution', x = 'Brand Code', y = 'Frequency') +
  theme_minimal()
```



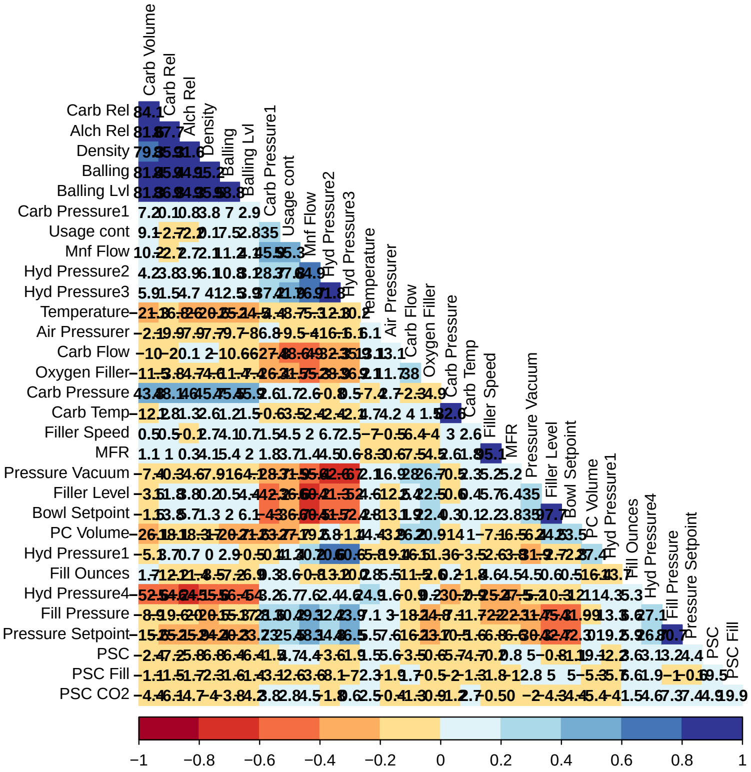
Majority of the entries in the dataset belong to `Brand Code` B. A and C have about the same number of entries. There are 120 missing values for `Brand Code`.

Correlation

First, we can plot a correlation matrix of our predictor variables to see which predictors are correlated with each other.

```
train_numeric_df <- train_df |>
  dplyr::select(where(is.numeric)) |>
  na.omit()

# Calculate correlation matrix
train_numeric_cor <- train_numeric_df |>
```

```
  dplyr::select(-PH) |>
  cor()

# Generate the correlation plot
corrplot(train_numeric_cor,
         method = "color",
         tl.col = "black",
         col = brewer.pal(n = 10,
                          name = "RdYlBu"),
         type = "lower",
         diag=FALSE,
         order = "hclust",
         addCoef.col = "black",
         number.cex = 0.8,
         tl.cex = 0.8,
         cl.cex = 0.8,
         addCoefasPercent = TRUE,
         number.digits = 1)
```


Carb Volume
Carb Rel
Alch Rel
Density
Balling
Balling Lvl
Carb Pressure1
Usage cont
Mnf Flow
Hyd Pressure2
Hyd Pressure3
Temperature
Air Pressurer
Carb Flow
Oxygen Filler
Carb Pressure
Carb Temp
Filler Speed
MFR
Pressure Vacuum
Filler Level
Bowl Setpoint
PC Volume
Hyd Pressure1
Fill Ounces
Hyd Pressure4
Fill Pressure
Pressure Setpoint
PSC
PSC Fill
PSC CO2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1

We can see a few instances of multicollinearity in our predictor variables. `Carb Rel`, `Alch Rel`, `Density`, `Balling` and `Balling Level` are all significantly positively correlated with each other. `Hyd Pressue2` is significantly positively correlated with `Hyd Pressure 3`. Likewise, `Carb Temp` with `Carb Pressure`, `MFR` with `Fill Speed`, `Bowl Setpoint` with `Fill Level`, and `Pressure Setpoint` with `Fill Pressure`.

There are also a number of variables that are highly negatively correlated with each other, such as `Pressure Vacuum` with `Hyd Pressure2` and `Hyd Pressure3`, `Mnf Flow` with `Filler Level` and `Bowl Setpoint`, and `Hyd Pressure4` with `Alch Rel`.

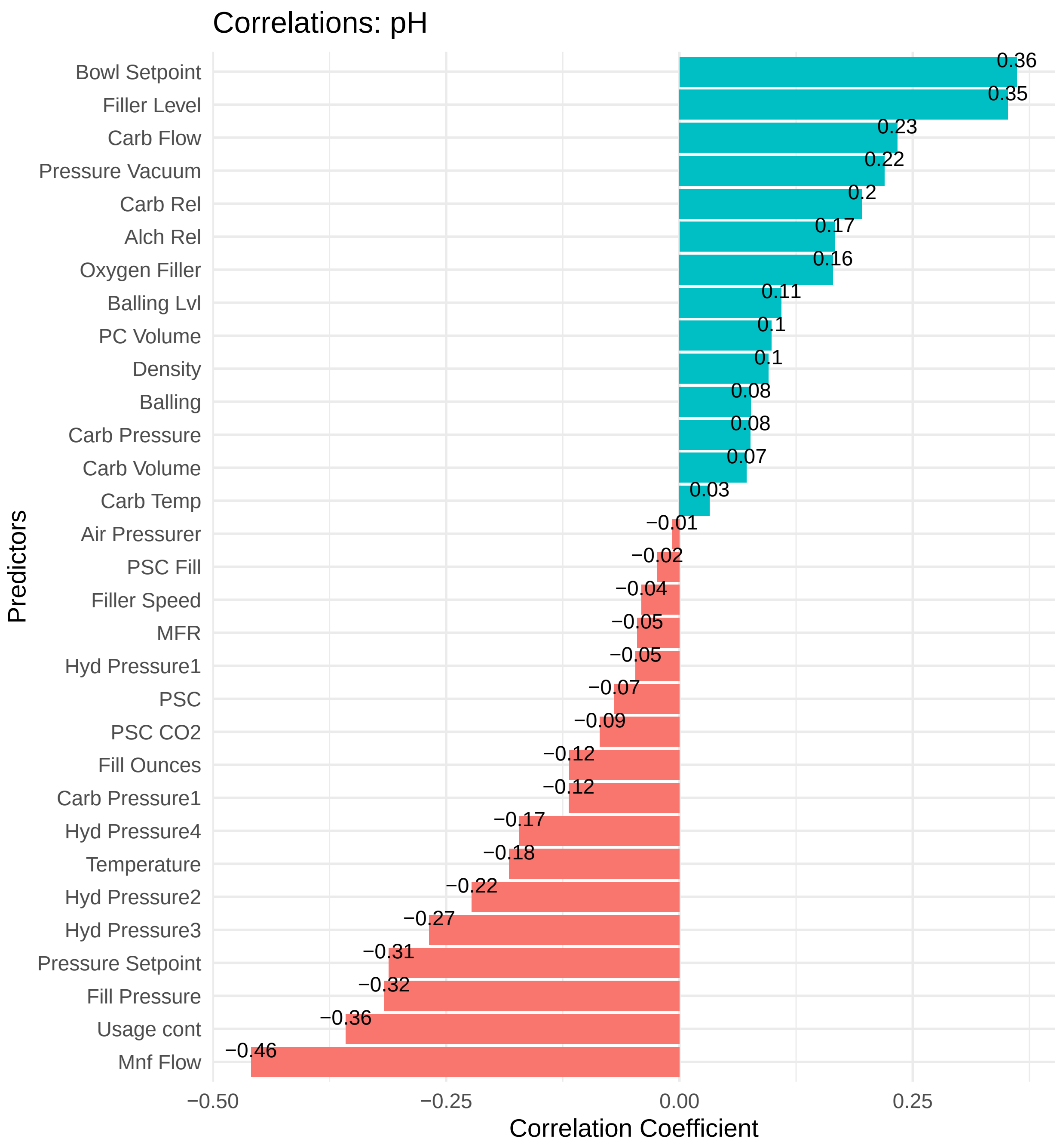
A number of other variables also display moderate correlations with each other, as can be seen from the medium blue and medium red squares in the correlation plot.

We will need to address these multicollinearity issues in our models.

PH

With `PH` being our response variable, assessing `PH`'s correlation with other variables is needed.

```r
train_numeric_df |>
  dplyr::select(-PH) |>  # Exclude 'PH' from predictors if needed
  cor(train_numeric_df$PH) |>  # Calculate correlations with 'PH'
  as.data.frame() |>
  rownames_to_column(var = "Predictor") |>
  filter(Predictor != "PH") |>  # Ensure 'PH' is not included as its own predictor
  mutate(Predictor = fct_reorder(factor(Predictor), V1)) |>  # Reorder factors by correlation for plott
  ggplot(aes(x = Predictor, y = V1, label = round(V1, 2))) +
    geom_col(aes(fill = ifelse(V1 < 0, "negative", "positive"))) +
    geom_text(color = "black", size = 3, vjust = -0.3) +
    coord_flip() +
    labs(title = "Correlations: pH", x = "Predictors", y = "Correlation Coefficient") +
    theme_minimal() +
    theme(legend.position = "none")
```



Individually, there are no variables that are extremely correlated with `PH`. `Mnf Flow` has the largest correlation of about -0.46. The most significantly positively correlated variables with `PH` are `Bowl Setpoint` and `Filler Level`. The most significantly negatively correlated variables, other than `Mnf Flow`, are `Usage cont`, `Fill Pressure`, and `Pressure Setpoint`.

Data Cleanup and Pre-Processing

First, to make it easier to reference our variables, let's make each column name snakecase.

```
names(train_df) <- snakecase::to_snake_case(names(train_df))
names(test_df) <- snakecase::to_snake_case(names(test_df))
```

Now, as ph is our target variable, we will need to remove any rows that do not have a value for this column.

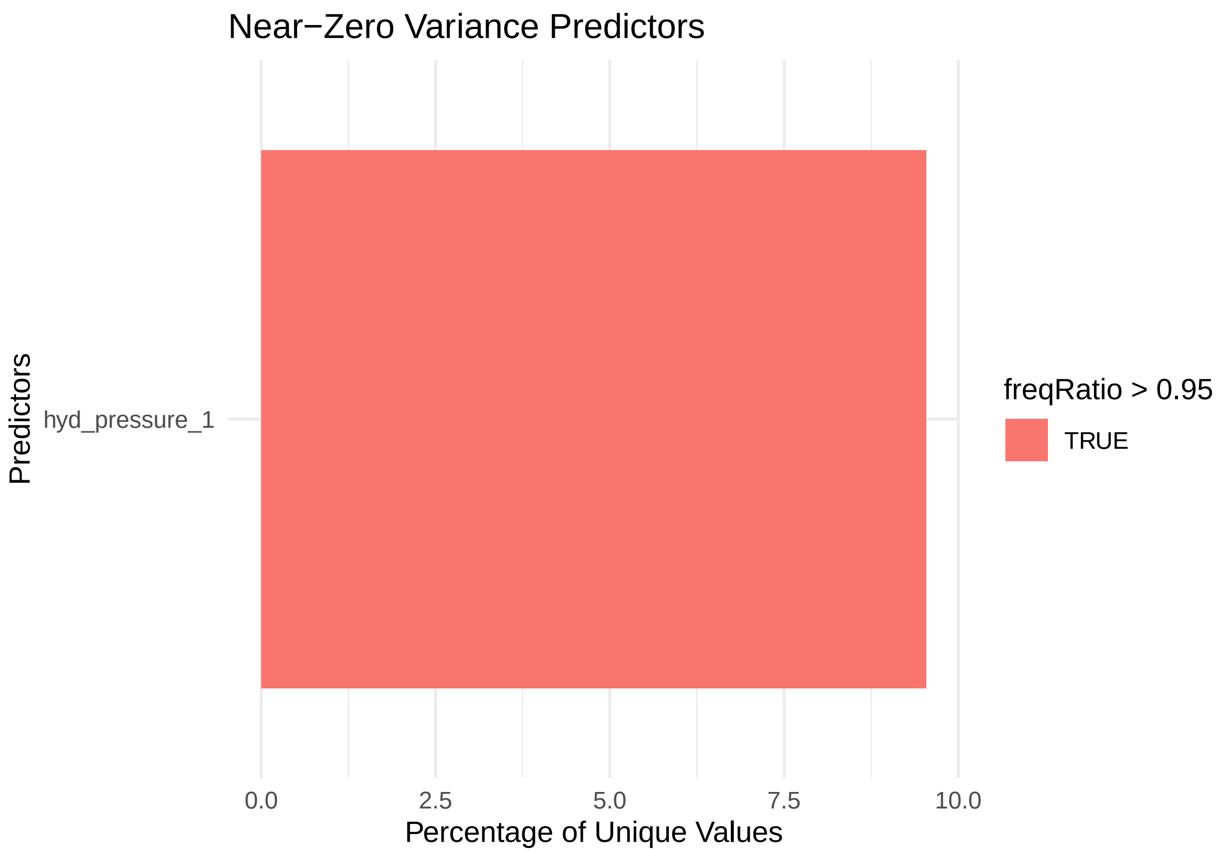
```
train_df <- train_df |>
  filter(!is.na(ph))
```

We will also transform our brand_code variable to categorized factors, replacing any NA value with “Unknown”.

```
train_df <- train_df |>
  dplyr::mutate(brand_code = ifelse(is.na(brand_code), "Unknown", brand_code),
              brand_code = factor(brand_code, levels = c('A','B','C','D','Unknown'), ordered = FALSE)

test_df <- test_df |>
  dplyr::mutate(brand_code = ifelse(is.na(brand_code), "Unknown", brand_code),
              brand_code = factor(brand_code, levels = c('A','B','C','D','Unknown'), ordered = FALSE)
```

We will identify unhelpful columns in the dataset, such as any variables with zero variance or near zero variance.

```
nzv_df <- nearZeroVar(train_df, saveMetrics= TRUE)
nzv_df <- as.data.frame(nzv_df) %>%
  rownames_to_column(var = "Predictor")

nzv_filtered_df <- nzv_df %>%
  filter(nzv == TRUE)

ggplot(nzv_filtered_df, aes(x = Predictor, y = percentUnique, fill = freqRatio > 0.95)) +
  geom_col(position = "dodge") +
  coord_flip() +
  labs(title = "Near-Zero Variance Predictors",
       x = "Predictors",
       y = "Percentage of Unique Values") +
  theme_minimal()
```



```
print(nzv_filtered_df)
```

```
##        Predictor freqRatio percentUnique zeroVar  nzv
## 1 hyd_pressure_1  31.03704      9.544215   FALSE TRUE
```

`hyd_pressure_1` is the only variable with near zero variance. We will not include this variable in our modeling.

Finally, we will pre-process the data for modeling.

The data is in the form of a tibble. For pre-processing using the `preProcess()` function from the `caret` package, we need the data in the form of a dataframe. We will use `as.data.frame()` to do this.

```
train_df <- as.data.frame(train_df)
test_df <- as.data.frame(test_df)
```

We will leverage `caret` package method preProcess to transform data using methods: + knnImpute - nearest neighbor to impute missing data + nzv = remove near-zero values identified above + corr = filters out highly correlated values addressing multicollinearity + center = subtracts the mean of the predictor's data (again from the data in x) from the predictor values + scale = divides by the standard deviation. + BoxCox = normalizes data * Use the `predict` function to process the list variables created with `preProcess()` to recreate the dataframe.

```
#remove pH from the train data set in order to only transform the predictors
train_preprocess_df <- train_df |>
```

```
  dplyr::select(-c(ph))

preProc_ls <- preProcess(train_preprocess_df, method = c("knnImpute", "nzv", "corr", "center", "scale",

train_preProc_df <- predict(preProc_ls, train_preprocess_df)

# rejoin with pH
train_preProc_df$ph <- train_df$ph

#remove pH from the test data set in order to only transform the predictors
test_preprocess_df <- test_df |>
  dplyr::select(-c(ph))

preProc_ls <- preProcess(test_preprocess_df, method = c("knnImpute", "nzv", "corr", "center", "scale",

test_preProc_df <- predict(preProc_ls, test_preprocess_df)

# rejoin with pH
test_preProc_df$ph <- test_df$ph
```

Let's check that no missing values remain.

```
# verify no NAs remain
colSums(is.na(train_preProc_df))
```

```
##        brand_code       carb_volume       fill_ounces         pc_volume
##                 0                 0                 0                 0
##     carb_pressure         carb_temp               psc          psc_fill
##                 0                 0                 0                 0
##          psc_co_2          mnf_flow   carb_pressure_1     fill_pressure
##                 0                 0                 0                 0
##    hyd_pressure_2    hyd_pressure_4       temperature        usage_cont
##                 0                 0                 0                 0
##         carb_flow               mfr   pressure_vacuum     oxygen_filler
##                 0                 0                 0                 0
##     bowl_setpoint pressure_setpoint     air_pressurer          alch_rel
##                 0                 0                 0                 0
##          carb_rel                ph
##                 0                 0
```

Data Partition

We will split the data into an 80:20 training and validation set.

```
set.seed(1234)  # for reproducibility

training_set_df <- createDataPartition(train_preProc_df$ph, p=0.8, list=FALSE)

train <- train_preProc_df[training_set_df,]
eval <- train_preProc_df[-training_set_df,]
```

We will now build several model using the data and we will evaluate each one to determine which is the best model for our data.

Modeling

Linear Model

```r
lm <- lm(ph ~ ., data=train)

summary(lm)
```

```
##
## Call:
## lm(formula = ph ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52114 -0.07953  0.01028  0.08828  0.74107
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)        8.5013168  0.0111835 760.167  < 2e-16 ***
## brand_codeB        0.0745672  0.0170588   4.371 1.30e-05 ***
## brand_codeC       -0.0736206  0.0187874  -3.919 9.20e-05 ***
## brand_codeD        0.0775571  0.0160734   4.825 1.50e-06 ***
## brand_codeUnknown -0.0068023  0.0217928  -0.312 0.754969
## carb_volume       -0.0109571  0.0084396  -1.298 0.194332
## fill_ounces       -0.0064805  0.0032356  -2.003 0.045325 *
## pc_volume         -0.0068601  0.0036763  -1.866 0.062180 .
## carb_pressure     -0.0009003  0.0122829  -0.073 0.941576
## carb_temp          0.0059642  0.0111378   0.535 0.592368
## psc               -0.0037035  0.0032107  -1.153 0.248842
## psc_fill          -0.0045761  0.0031837  -1.437 0.150770
## psc_co_2          -0.0059852  0.0031667  -1.890 0.058899 .
## mnf_flow          -0.0705019  0.0061363 -11.489  < 2e-16 ***
## carb_pressure_1    0.0355911  0.0037543   9.480  < 2e-16 ***
## fill_pressure      0.0160609  0.0043311   3.708 0.000214 ***
## hyd_pressure_2     0.0193080  0.0046854   4.121 3.93e-05 ***
## hyd_pressure_4    -0.0016373  0.0045338  -0.361 0.718034
## temperature       -0.0137708  0.0034844  -3.952 8.01e-05 ***
## usage_cont        -0.0223799  0.0038886  -5.755 9.97e-09 ***
## carb_flow          0.0127671  0.0040203   3.176 0.001517 **
## mfr               -0.0004474  0.0034511  -0.130 0.896873
## pressure_vacuum   -0.0055955  0.0040339  -1.387 0.165563
## oxygen_filler     -0.0082534  0.0042057  -1.962 0.049850 *
## bowl_setpoint      0.0366908  0.0045881   7.997 2.12e-15 ***
## pressure_setpoint -0.0183039  0.0044016  -4.158 3.34e-05 ***
## air_pressurer      0.0008728  0.0032750   0.266 0.789886
## alch_rel           0.0074259  0.0105147   0.706 0.480123
## carb_rel           0.0072810  0.0063010   1.156 0.248008
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1353 on 2026 degrees of freedom
## Multiple R-squared:  0.4023, Adjusted R-squared:  0.394
## F-statistic:  48.7 on 28 and 2026 DF,  p-value: < 2.2e-16
```

The R^2 for this model is 0.394 and there are a number of insignificant variables in the model. Let's use the `step()` function to remove some of the more insignificant variables.

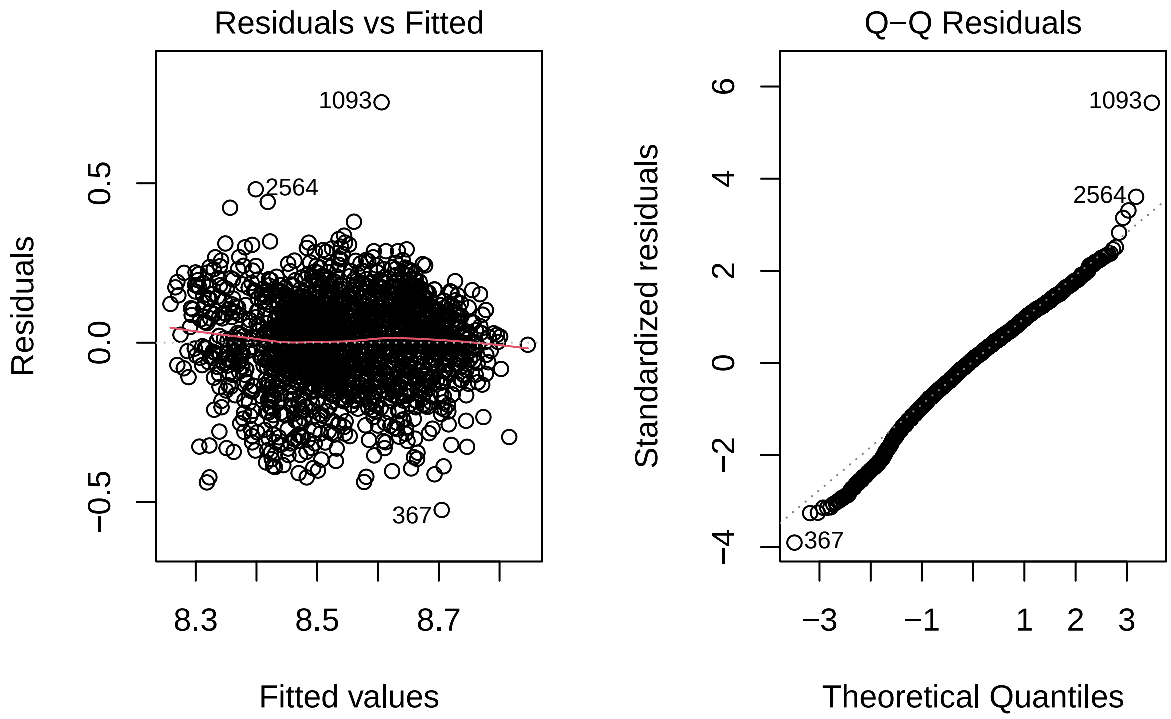
```r
lm_update <- step(lm, direction="both", trace=0)

summary(lm_update)
```

```
##
## Call:
## lm(formula = ph ~ brand_code + carb_volume + fill_ounces + pc_volume +
##     carb_temp + psc_fill + psc_co_2 + mnf_flow + carb_pressure_1 +
##     fill_pressure + hyd_pressure_2 + temperature + usage_cont +
##     carb_flow + pressure_vacuum + oxygen_filler + bowl_setpoint +
##     pressure_setpoint + carb_rel, data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.52485 -0.07939  0.01110  0.08905  0.75411
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        8.504677   0.009706 876.202  < 2e-16 ***
## brand_codeB        0.066065   0.012472   5.297 1.30e-07 ***
## brand_codeC       -0.081771   0.015138  -5.402 7.37e-08 ***
## brand_codeD        0.086058   0.012101   7.112 1.58e-12 ***
## brand_codeUnknown -0.013530   0.019369  -0.699 0.484940
## carb_volume       -0.012031   0.005914  -2.035 0.042029 *
## fill_ounces       -0.006480   0.003217  -2.015 0.044083 *
## pc_volume         -0.007950   0.003510  -2.265 0.023606 *
## carb_temp          0.005226   0.003088   1.692 0.090794 .
## psc_fill          -0.005203   0.003122  -1.667 0.095730 .
## psc_co_2          -0.006242   0.003149  -1.982 0.047598 *
## mnf_flow          -0.070819   0.006093 -11.623  < 2e-16 ***
## carb_pressure_1    0.035682   0.003717   9.599  < 2e-16 ***
## fill_pressure      0.016100   0.004187   3.846 0.000124 ***
## hyd_pressure_2     0.019547   0.004579   4.269 2.06e-05 ***
## temperature       -0.013855   0.003412  -4.060 5.08e-05 ***
## usage_cont        -0.022723   0.003845  -5.909 4.02e-09 ***
## carb_flow          0.013418   0.003892   3.447 0.000577 ***
## pressure_vacuum   -0.005954   0.003943  -1.510 0.131187
## oxygen_filler     -0.008263   0.004164  -1.984 0.047356 *
## bowl_setpoint      0.036745   0.004514   8.140 6.84e-16 ***
## pressure_setpoint -0.018189   0.004356  -4.175 3.10e-05 ***
## carb_rel           0.008844   0.005929   1.492 0.135945
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1351 on 2032 degrees of freedom
## Multiple R-squared:  0.4017, Adjusted R-squared:  0.3952
## F-statistic: 62.01 on 22 and 2032 DF,  p-value: < 2.2e-16
```

The R^2 value increased slightly to about 0.395.

Let's check the diagnostic plots for this model.

```r
par(mfrow = c(1,2))
plot(lm_update, which = c(1,2))
```



From the residuals vs fitted plot, there does not seem to be any heteroscedasticity, so constant variance is fulfilled. From the QQ-plot, the residuals seem relatively normally distributed although they diverge from the normal line toward the lower end.

Let's evaluate how this model performs on the evaluation data.

```r
lm_pred <- predict(lm_update, eval)
(lm_metrics <- postResample(lm_pred, eval$ph))
```

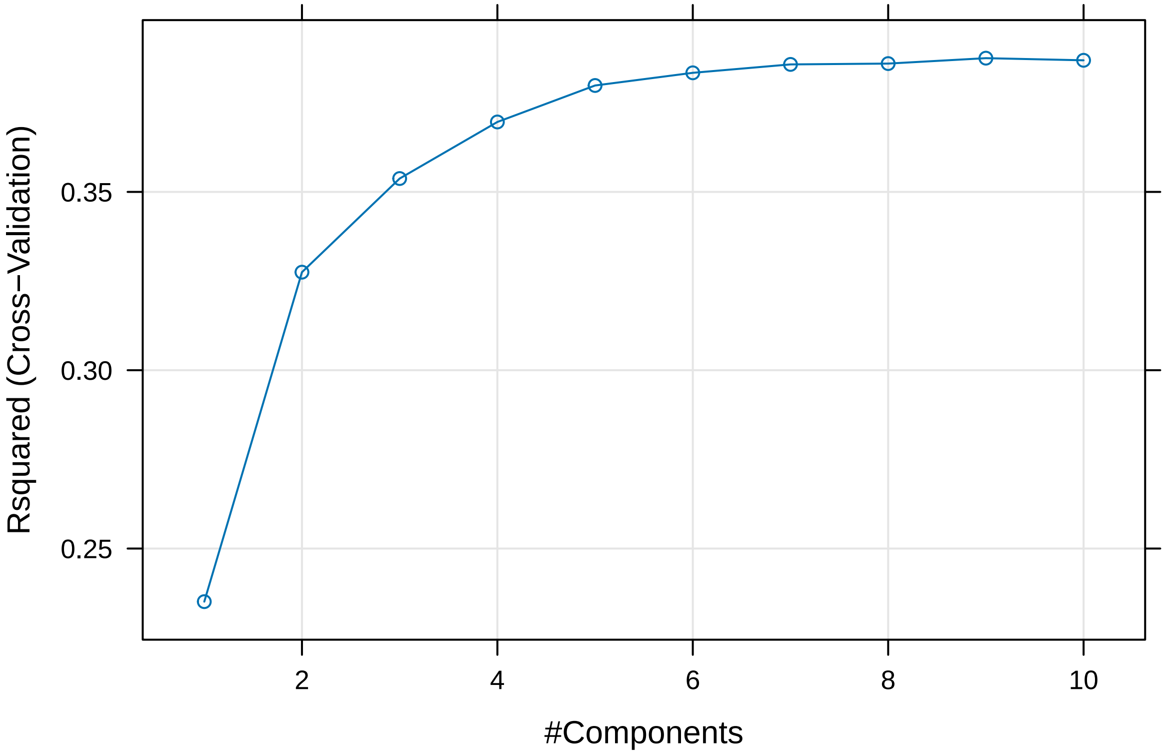
```
##      RMSE  Rsquared       MAE 
## 0.1305888 0.3934348 0.1043181
```

The evaluation set has an $RMSE$ of 0.13 and an R^2 of 0.39.

PLS Model

```r
set.seed(2341)

# generate model
```

```r
pls_model <- train(ph ~ .,
                   data=train,
                   method='pls',
                   metric='Rsquared',
                   tuneLength=10,
                   trControl=trainControl(method = "cv",  number = 10))

plot(pls_model)
```



```r
pls_model
```

```
## Partial Least Squares
##
## 2055 samples
##   25 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1849, 1850, 1849, 1849, 1850, 1850, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     0.1521396  0.2351118  0.1195055
##    2     0.1426937  0.3274893  0.1115541
##    3     0.1399277  0.3537450  0.1087935
##    4     0.1382565  0.3696090  0.1080814
##    5     0.1370713  0.3798388  0.1066452
##    6     0.1367173  0.3833915  0.1064445
##    7     0.1364496  0.3857474  0.1060137
##    8     0.1364197  0.3859793  0.1059609
##    9     0.1362762  0.3875012  0.1058786
##   10     0.1363538  0.3868979  0.1059767
##
## Rsquared was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 9.
```

The optimal number of components for the PLS model was 9, with a corresponding R^2 of about 0.39.

Let's take a look at the most important variables for the PLS model.

```
plot(varImp(pls_model))
```

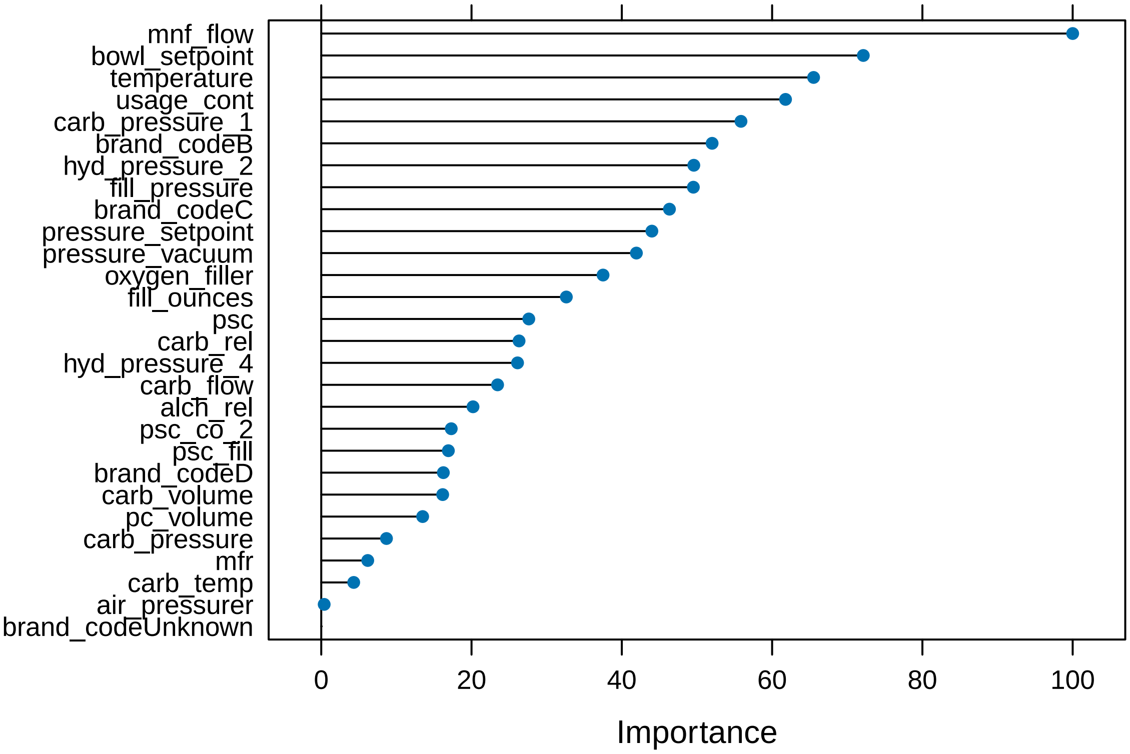
```
## Warning: package 'pls' was built under R version 4.3.3
```

```
##
## Attaching package: 'pls'
```

```
## The following object is masked from 'package:corrplot':
##
##     corrplot
```

```
## The following object is masked from 'package:caret':
##
##     R2
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```



The most important variable is `mnf_flow`.

Let's evaluate how this model performs on the evaluation data.

```
# evaluate model metrics
pls_pred <- predict(pls_model, eval)
(pls_metrics <- postResample(pls_pred, eval$ph))
```

```
##      RMSE  Rsquared       MAE
## 0.1297584 0.4008291 0.1034359
```

The evaluation set for the PLS model has a slightly improved R^2 of 0.40.

KNN Model

```
set.seed(613)

knn_model <- train(ph ~ .,
                   data = train,
                   method = "knn",
                   tuneLength = 10)
knn_model
```

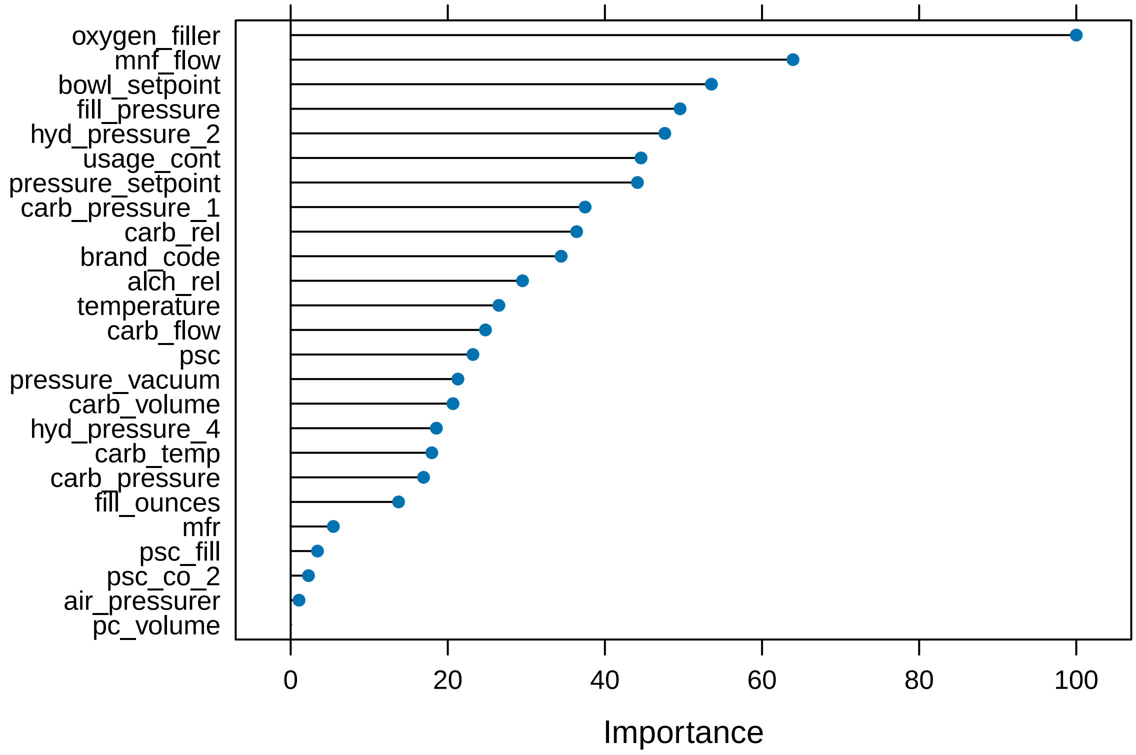
```
## k-Nearest Neighbors
```

```
##
## 2055 samples
##   25 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 2055, 2055, 2055, 2055, 2055, 2055, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    5  0.1390984  0.3987875  0.10224238
##    7  0.1356105  0.4178307  0.10063952
##    9  0.1340402  0.4277480  0.10008633
##   11  0.1334694  0.4317359  0.09995445
##   13  0.1334360  0.4320796  0.10011994
##   15  0.1337366  0.4297865  0.10053509
##   17  0.1341251  0.4271288  0.10093821
##   19  0.1342850  0.4264191  0.10118382
##   21  0.1345970  0.4241648  0.10158822
##   23  0.1350066  0.4208244  0.10207108
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 13.
```

The optimal k was 13, with a corresponding R^2 value of 0.43. This is improved over both the linear and PLS models.

Let's take a look at the most important variables for this model.

```
plot(varImp(knn_model))
```



For the KNN model, `oxygen_filler` is the most important variable and `mnf_filler` is the second most important variable.

Let's evaluate how this model performs on the evaluation data.

```
knn_pred <- predict(knn_model, eval)
(knn_metrics <- postResample(knn_pred, eval$ph))
```

```
##       RMSE   Rsquared        MAE
## 0.12001314 0.49682031 0.09114698
```

The KNN model performs much better than the linear and PLS models, with an R^2 of about 0.5 on the evaluation set.

MARS Model

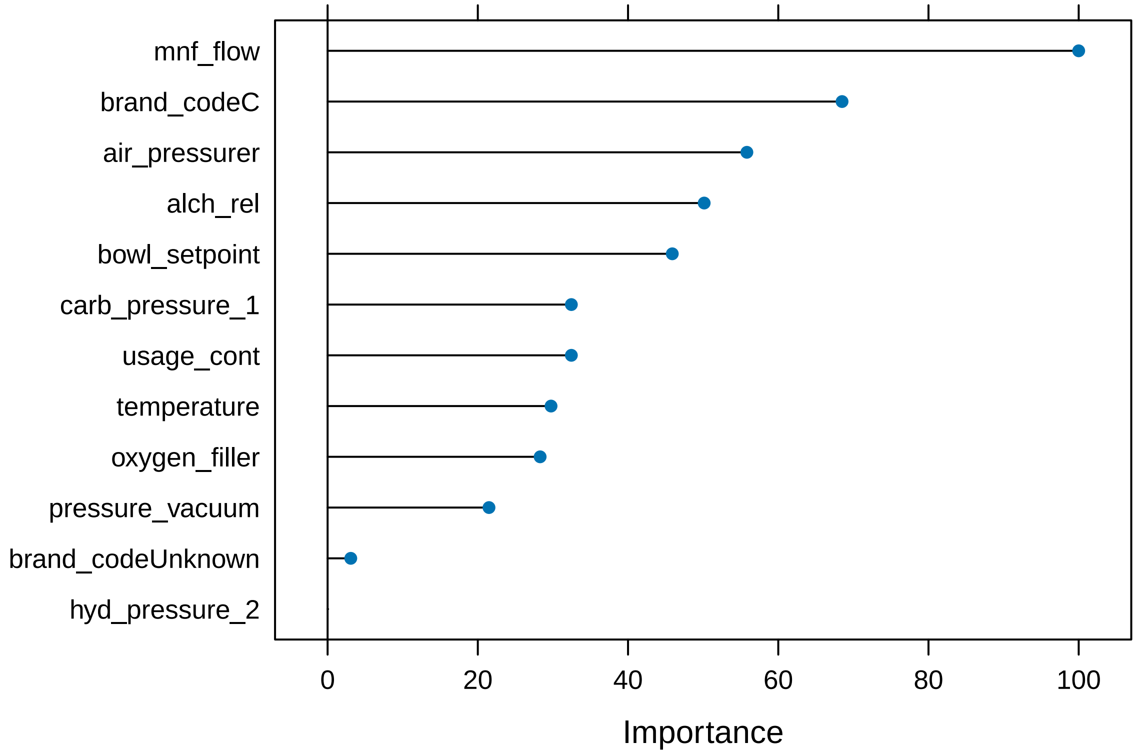
```
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)

set.seed(613)

mars_model <- train(ph ~ .,
                    data = train,
                    method = "earth",
                    tuneGrid = marsGrid,
                    trControl = trainControl(method = "cv"))

mars_model
```

```
## Multivariate Adaptive Regression Spline
##
## 2055 samples
##   25 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1848, 1849, 1849, 1850, 1849, 1851, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE       Rsquared   MAE
##   1        2      0.1535873  0.2212785  0.11968101
##   1        3      0.1448777  0.3064319  0.11317776
##   1        4      0.1434915  0.3189151  0.11184578
##   1        5      0.1426340  0.3273142  0.11082134
##   1        6      0.1412078  0.3398698  0.10922470
##   1        7      0.1395662  0.3560322  0.10767575
##   1        8      0.1377089  0.3727176  0.10627053
##   1        9      0.1367601  0.3811219  0.10573257
##   1       10      0.1351608  0.3950965  0.10476198
##   1       11      0.1347769  0.3985716  0.10416362
##   1       12      0.1334250  0.4105160  0.10311047
##   1       13      0.1345734  0.4024726  0.10345874
##   1       14      0.1342337  0.4051460  0.10315766
##   1       15      0.1346415  0.4016102  0.10311742
##   1       16      0.1346262  0.4022586  0.10291964
##   1       17      0.1345258  0.4033978  0.10279534
##   1       18      0.1342011  0.4060808  0.10271496
##   1       19      0.1337828  0.4096137  0.10260958
##   1       20      0.1335053  0.4120417  0.10239299
##   1       21      0.1334387  0.4125922  0.10238966
##   1       22      0.1333905  0.4134484  0.10225990
##   1       23      0.1335464  0.4124154  0.10222786
##   1       24      0.1334183  0.4136389  0.10211975
##   1       25      0.1334758  0.4137009  0.10181612
##   1       26      0.1337060  0.4119887  0.10198165
##   1       27      0.1333982  0.4145935  0.10164635
##   1       28      0.1330357  0.4175293  0.10140635
##   1       29      0.1328101  0.4191835  0.10122453
##   1       30      0.1324069  0.4227461  0.10096221
##   1       31      0.1321572  0.4246428  0.10070312
##   1       32      0.1323947  0.4229480  0.10099467
##   1       33      0.1322344  0.4242455  0.10090254
##   1       34      0.1321323  0.4250261  0.10102388
##   1       35      0.1344771  0.4145926  0.10136173
##   1       36      0.1344011  0.4149405  0.10131512
##   1       37      0.1344278  0.4148383  0.10128010
##   1       38      0.1343354  0.4155427  0.10128834
##   2        2      0.1530944  0.2265794  0.11898679
##   2        3      0.1462250  0.2942891  0.11394808
##   2        4      0.1440784  0.3153016  0.11204350
##   2        5      0.1421114  0.3337770  0.11086995
##   2        6      0.1407611  0.3467498  0.10956907
##   2        7      0.1394805  0.3577464  0.10850651
##   2        8      0.1376499  0.3747515  0.10665540
##   2        9      0.1359799  0.3889583  0.10497439
##   2       10      0.1347076  0.4004098  0.10405077
##   2       11      0.1338458  0.4083320  0.10339104
##   2       12      0.1327088  0.4178045  0.10271164
##   2       13      0.1321305  0.4225395  0.10189380
##   2       14      0.1313585  0.4290817  0.10100493
##   2       15      0.1304205  0.4371764  0.09989178
##   2       16      0.1294247  0.4452763  0.09910754
##   2       17      0.1283584  0.4541348  0.09793470
##   2       18      0.1285692  0.4529985  0.09836378
##   2       19      0.1278478  0.4597029  0.09769820
##   2       20      0.1276282  0.4614473  0.09735486
##   2       21      0.1276065  0.4617771  0.09712116
##   2       22      0.1275896  0.4622986  0.09733336
##   2       23      0.1271724  0.4656356  0.09694150
##   2       24      0.1272758  0.4650817  0.09709139
##   2       25      0.1274095  0.4643671  0.09731865
##   2       26      0.1272583  0.4658037  0.09704997
##   2       27      0.1269681  0.4683432  0.09672071
##   2       28      0.1269133  0.4688745  0.09657392
##   2       29      0.1268942  0.4691397  0.09647809
##   2       30      0.1267669  0.4702629  0.09639925
##   2       31      0.1285202  0.4622434  0.09659245
##   2       32      0.1288632  0.4612861  0.09678814
##   2       33      0.1293031  0.4596399  0.09686214
##   2       34      0.1297509  0.4583308  0.09692435
##   2       35      0.1295061  0.4590350  0.09693595
##   2       36      0.1293917  0.4598075  0.09684188
##   2       37      0.1293917  0.4598075  0.09684188
##   2       38      0.1293917  0.4598075  0.09684188
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 30 and degree = 2.
```

The MARS model is optimal at nprune = 30 and degree = 2. The R^2 at this iteration is 0.47 which is not improved from the KNN model.

Let's take a look at the most important variables for this model.

```
plot(varImp(mars_model))
```



This model has fewer important variables than the PLS and the KNN models. Like with the PLS model, `mnf_flow` is the most important variable.

Let's evaluate how this model performs on the evaluation data.

```
mars_pred <- predict(mars_model, eval)
(mars_metrics <- postResample(mars_pred, eval$ph))
```

```
##      RMSE  Rsquared       MAE
## 0.1184498 0.5038378 0.0910705
```

The evaluation set has an R^2 of 0.5, slightly improved over the KNN model.

SVM Model

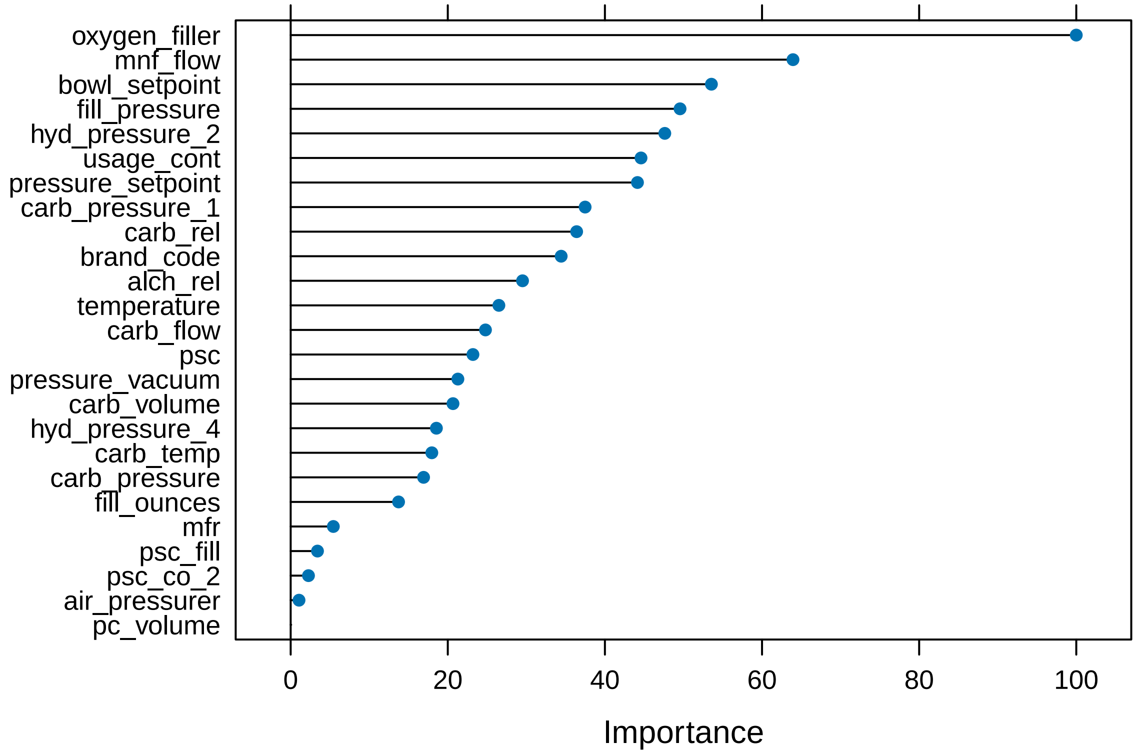
```
set.seed(613)

svm_model <- train(ph ~ .,
                   data = train,
                   method = "svmRadial",
                   tuneLength = 14,
                   trControl = trainControl(method = "cv"))

svm_model
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 2055 samples
##   25 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1848, 1849, 1849, 1850, 1849, 1851, ...
## Resampling results across tuning parameters:
##
##   C        RMSE       Rsquared   MAE
##      0.25  0.1275992  0.4675130  0.09465438
##      0.50  0.1244643  0.4916487  0.09175646
##      1.00  0.1215873  0.5138680  0.08937335
##      2.00  0.1194783  0.5294227  0.08777912
##      4.00  0.1184050  0.5380039  0.08703835
##      8.00  0.1190210  0.5366143  0.08821722
##     16.00  0.1220096  0.5225359  0.09048471
##     32.00  0.1268173  0.5010522  0.09380972
##     64.00  0.1338263  0.4709833  0.09924286
##    128.00  0.1416055  0.4371453  0.10477848
##    256.00  0.1489243  0.4090902  0.10985776
##    512.00  0.1534970  0.3915444  0.11337869
##   1024.00  0.1535865  0.3916381  0.11343416
##   2048.00  0.1535865  0.3916381  0.11343416
##
## Tuning parameter 'sigma' was held constant at a value of 0.0240063
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.0240063 and C = 4.
```

The optimal model has a sigma of about 0.024 and C = 4. The R^2 for this model is about 0.54.

Let's take a look at the most important variables for this model.

```
plot(varImp(svm_model))
```



This model has the same important variables as the MARS model in the same order.

Let's evaluate how this model performs on the evaluation data.

```
svm_pred <- predict(svm_model, eval)
(svm_metrics <- postResample(svm_pred, eval$ph))
```

```
##      RMSE  Rsquared       MAE
## 0.1098260 0.5760762 0.0823950
```

This model is much improved from the previous models, with an R^2 of about 0.58 for the evaluation set.

Random Forest Model

```
set.seed(613)

rf_model <- randomForest(ph ~ .,
                         data = train,
                         importance = TRUE,
                         ntree = 1000)

rf_model
```

```
##
```

```
## Call:
##  randomForest(formula = ph ~ ., data = train, importance = TRUE,      ntree = 1000)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 0.01051914
##                     % Var explained: 65.14
```

```
rf_model
```

```
##
## Call:
##  randomForest(formula = ph ~ ., data = train, importance = TRUE,      ntree = 1000)
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 8
##
##           Mean of squared residuals: 0.01051914
##                     % Var explained: 65.14
```

The model explains 65% of the variability, much improved from our previous models.

Let's take a look at the most important variables for this model.

```
varImp(rf_model) |>
  arrange(desc(Overall)) |>
  knitr::kable()
```

	Overall
brand_code	76.3123199
mnf_flow	57.7016188
pressure_vacuum	52.9055158
usage_cont	50.0934986
oxygen_filler	49.3531816
temperature	42.4104849
alch_rel	41.9715716
air_pressurer	41.5575295
carb_rel	41.1594852
bowl_setpoint	38.8790579
carb_flow	35.6962596
carb_pressure_1	31.7826384
carb_volume	27.5636344
mfr	26.9018896
hyd_pressure_2	21.5836876
pc_volume	21.2972441
fill_pressure	19.8120330
pressure_setpoint	19.6711754
hyd_pressure_4	17.5602769
fill_ounces	7.6233517
carb_pressure	6.5679198
carb_temp	3.9926770

	Overall
psc_co_2	2.0111185
psc_fill	0.2927448
psc	-1.1513903

`brand_code` is the most important variable for this model and `mnf_flow` is the second most important.

Let's evaluate how this model performs on the evaluation data.

```
rf_pred <- predict(rf_model, eval)
(rf_metrics <- postResample(rf_pred, eval$ph))
```

```
##       RMSE   Rsquared        MAE
## 0.09531386 0.68860455 0.07130942
```

This model performs the best from all the previous models. The R^2 for the evaluation set is 0.69.

Let's take a look at all the metrics together.

```
rbind(lm_metrics, pls_metrics, knn_metrics, mars_metrics, svm_metrics, rf_metrics) |>
  knitr::kable()
```

	RMSE	Rsquared	MAE
lm_metrics	0.1305888	0.3934348	0.1043181
pls_metrics	0.1297584	0.4008291	0.1034359
knn_metrics	0.1200131	0.4968203	0.0911470
mars_metrics	0.1184498	0.5038378	0.0910705
svm_metrics	0.1098260	0.5760762	0.0823950
rf_metrics	0.0953139	0.6886045	0.0713094

We can clearly see that the random forest model has the highest prediction accuracy when it comes to the evaluation set, with an R^2 of about 69%.