Project #2
ABC Beverage PH Levels

DATA624 SPRING 2018

GROUP 2

BRIAN KREIS

BIN LIN

CHRIS MARTIN

ASHER MEYERS

Contents

ABC Beverage: PH Predictive Factors

Executive Summary / Non-Technical Report

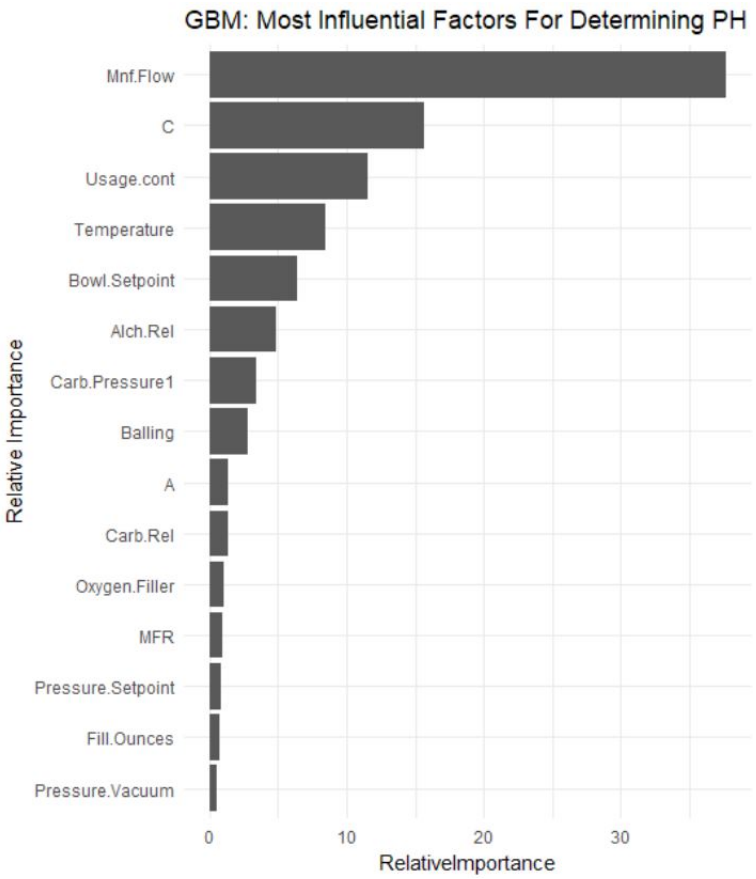
As data scientists for ABC Beverage, our task is to discover the predictive factors of a batch's PH levels and create a predictive model.

Our project team has obtained a data set containing 2,571 observations with each observation representing a batch of beverages that were produced. Variables included in the data set provide information on the manufacturing process that resulted in the creation of each batch. The data contains 32 predictor variables, as well as a training and test set.

Our team used this to build multiple models on the training data and run analysis to determine which model performed best. After performing a thorough comparison of several types of predictive models, we determined that a Random Forest model has the best predictive ability; the gradient boosting method was used to determine the most influential factors in determining PH levels. The top 5 are:

1. Mnt.Flow
2. Brand C
3. Usage.cont
4. Temperature
5. Bowl.Setpoint

The average error (Root Mean Squared Error, or RMSE, in technical parlance) is approximately 0.092, or approximately 1% of the average reported PH, on data set aside for validation of the model.


GBM: Most Influential Factors For Determining PH
Mnf.Flow
C
Usage.cont
Temperature
Bowl.Setpoint
Alch.Rel
Carb.Pressure1
Balling
A
Carb.Rel
Oxygen.Filler
MFR
Pressure.Setpoint
Fill.Ounces
Pressure.Vacuum
Relative Importance
0
10
20
30
RelativeImportance

Technical Report (with R Code)

The models and analysis used a variety of packages which will need to be installed and loaded:

```
library(psych)
library(ggplot2)
library(ggthemes)
library(dplyr)
library(DataExplorer)
library(knitr)
library(GGally)
library(missForest)
library(Hmisc)
library(earth)
library(caret)
library(RandomForest)
library(missingForest)
library(gbm)
```

For reproducibility of the results, the data was loaded to and accessed from a Github repository.

```
#Data to build models
data1 <- read.csv("https://raw.githubusercontent.com/624-
Group2/Project-2/master/StudentData.csv", header=TRUE, sep=",")

#Data to make predictions on
toPred <- read.csv("https://raw.githubusercontent.com/624-
Group2/Project-2/master/StudentEvaluation-%20TO%20PREDICT.csv")
```

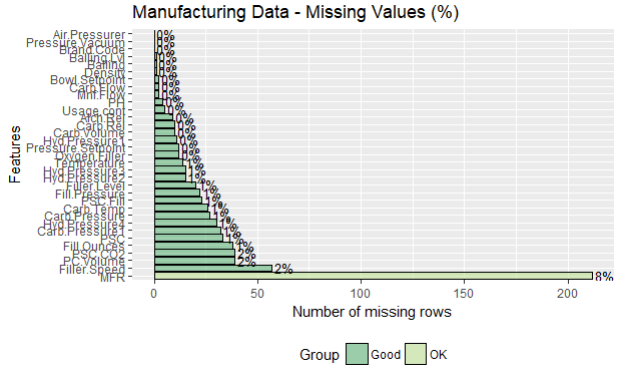
Data Exploration and Statistic Measures

Data exploration and statistical measures were used to understand the data to determine how to process the dataset for modelling.

Missing and Zero Values

Plotting the data as done below shows that the data has a number of features with missing values. Other methods of exploration used included counting the missing values by feature, understanding the distribution of missing values by Brand Code, and counting the missing values by Brand Code.

```
plot_missing(data1, title="Manufacturing Data - Missing Values
(%)")
```




```
kable(sapply(data1, FUN = function(x) sum(is.na(x))))
```

```
data1 %>%
  group_by(Brand.Code) %>%
  summarise_all(funs(sum(is.na(.))))

table(data1$Brand.Code)
```

Presumably, because we have identified NA values across various brands, both named and unnamed, we would expect that the NA values are not informative. In other words, if information for a particular brand is usually recorded, we would expect failure to record information to be the result of an error and not a typical process change for a particular brand. As our knowledge of the production process itself is limited, we will rely on the subject matter expertise to let us know if this presumption is incorrect.

MFR stands out as a variable with a significant number of missing values; however, the percentage of missing values is still low enough where imputation is not unreasonable. We will likely find it necessary to drop the few observations that have an NA value for the response variable.

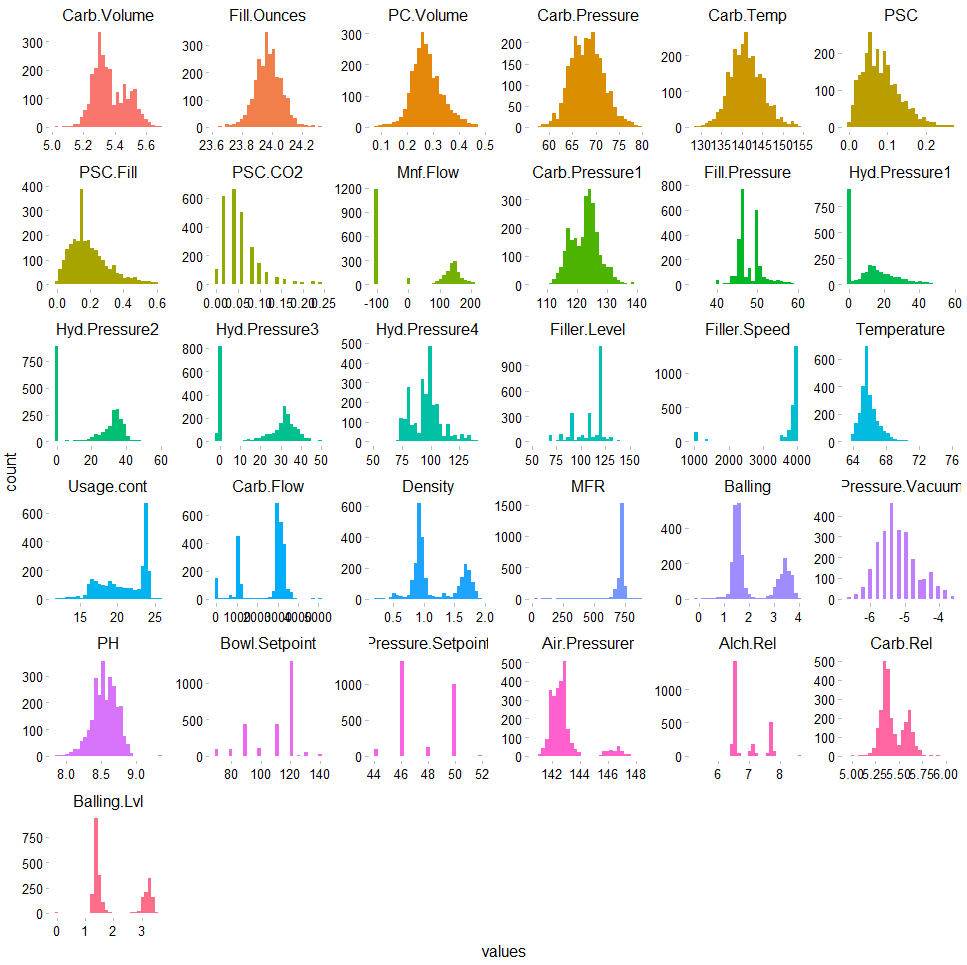
Descriptive Statistics and Data Exploration

Descriptive statistics were performed for all predictor and response variables to explore the data. The 'describe' package helps in this by automatically calculating descriptive statistics for each feature and outputting metadata on each such as the number missing, number of distinct values in the feature, etc.

```
#Use Describe Package to calculate Descriptive Statistic
(CC_des <- describe(data1, na.rm=TRUE, interp=FALSE, skew=TRUE,
ranges=TRUE, trim=.1, type=3, check=TRUE, fast=FALSE,
quant=c(.1,.25,.75,.90), IQR=TRUE))
```

We have significant skewness in several variables, and, depending on our choice of regression technique may require transformation.

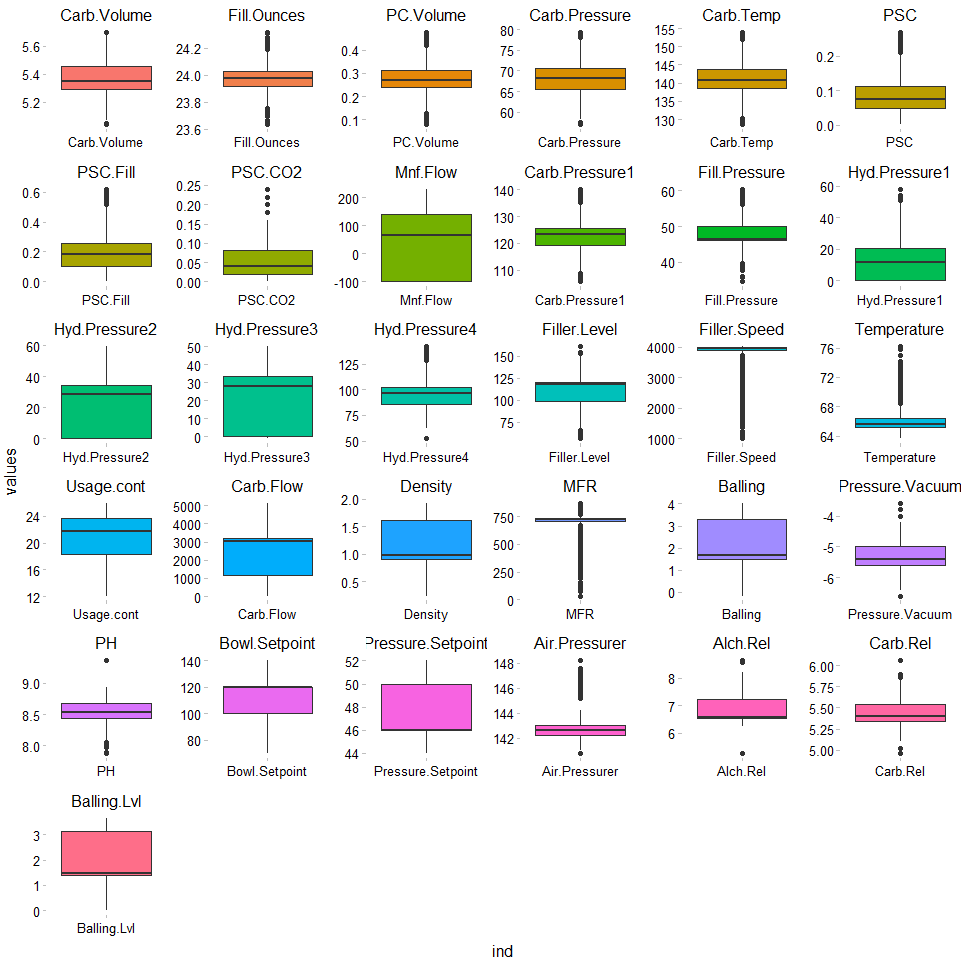
```
vis1 <- data1 %>% select(-PH -Brand.Code)

par(mfrow=(c(1,1)))
ggplot(stack(vis1), aes(values, fill=ind))+
  facet_wrap(~ind, scales = "free") +
  geom_histogram() +
  theme_pander()+
  theme(legend.position="none")
```



We have a varying mix of distributions for the different data points. For example,

- Discrete: The variable Pressure.Setpoint appears to represent a categorical setting ranging from 44 to 52; however, we do not know at first glance if our training set contains all possible values that we might see in the test set or in general practice.
- Normal Continuous: Carb.Temp appears to be normally distributed.
- Multi Modal data: Density appears to have many values around 1 and another large concentration of values around 1.5.

```
ggplot(stack(vis1), aes(x = ind, y = values, fill=ind))+
  facet_wrap(~ind, scales = "free") +
  geom_boxplot() +
  theme_pander() +
  theme(legend.position="none")
```

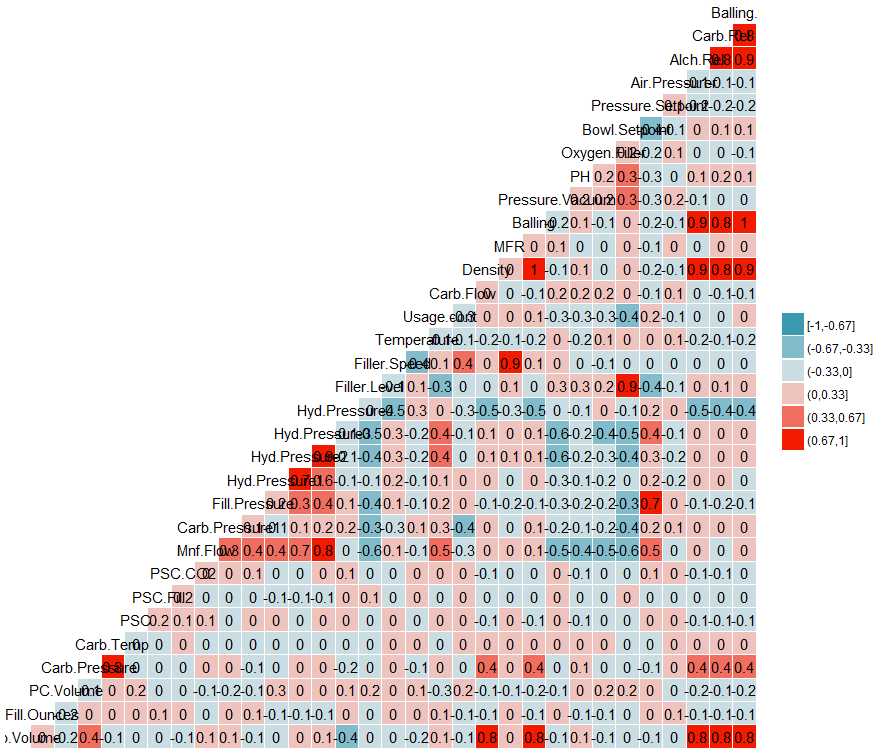


The boxplots above demonstrate the skewness of a number of variables as we have discussed.

Correlation

The tables below represent correlation between response and predictor variables. There is high collinearity among certain variables. For instance, in Density and Alch.Rel. This is also true of Alch.Res and Brand D.

```
ggcorr(data1, method = "pairwise", label=TRUE, nbreaks=6)
```



Below is a look at the unique values for each variable. As mentioned, it appears that some of these variables represent various non-continuous settings.

```
apply(data1, 2, function(x)length(unique(x)))
```

Data Manipulation

Data manipulation techniques were used to adjust the data for use in predictions.

Dummy Variables

Dummy variables are used to replace categorical values, specifically where Brand Code is used.

```
data1$A <- ifelse(data1$Brand.Code == "A", 1, 0)
data1$B <- ifelse(data1$Brand.Code == "B", 1, 0)
data1$C <- ifelse(data1$Brand.Code == "C", 1, 0)
data1$D <- ifelse(data1$Brand.Code == "D", 1, 0)
data1 <- data1 %>% select(-Brand.Code)


toPred$A <- ifelse(toPred$Brand.Code == "A", 1, 0)
toPred$B <- ifelse(toPred$Brand.Code == "B", 1, 0)
toPred$C <- ifelse(toPred$Brand.Code == "C", 1, 0)
toPred$D <- ifelse(toPred$Brand.Code == "D", 1, 0)
toPred <- toPred %>% select(-Brand.Code)
```

Handling Missing Values

As previously discussed, missing values are important to either skip or impute. After analysis and getting a deeper understanding of our data, the decision has been made to impute missing values to preserve the data and records.

```
#The following will impute the data. For efficiency the data sets
can be loaded from github below.
# ##For Windows to run in parallel
# library(parallel)
# library(doParallel)
#
# cluster <- makeCluster(detectCores() - 1) # convention to leave
1 core for OS
# registerDoParallel(cluster)
#
# #impute missing training data
# set.seed(123)
# dfImputed <- missForest(data1, parallelize = 'forests')
# write.csv(dfImputed$ximp, "StudentDataImputedMF")
#
# #impute missing test data
# set.seed(123)
# predImputed <- missForest(toPred, parallelize = 'forests')
# write.csv(predImputed$ximp, "PredictImputedMF")
#
#
# #turn off parallel processing
# stopCluster(cluster)
# #resume use of the sequential backend
# registerDoSEQ()

#read imputed train set
imputed <- read.csv("https://raw.githubusercontent.com/624-
Group2/Project-2/master/StudentDataImputedMF")
imputed <- imputed %>% select(-X)
sum(is.na(imputed))

#read imputed test set
predictImp <- read.csv("https://raw.githubusercontent.com/624-
Group2/Project-2/master/PredictImputedMF")
predictImp <- predictImp %>% select(-X)
sum(is.na(predictImp))
```

Predictions

With the data prepped, we used a variety of models to find the most predictive features in finding the PH levels of the batch.

Test & Training Sets

Prior to testing our models on the actual prediction data set, it is prudent to evaluate our models against data where the response is known, so our predictions can be compared.

```
smp <- floor(0.70 * nrow(imputed))


set.seed(123)
train_index <- sample(seq_len(nrow(imputed)), size = smp, replace
= FALSE)

train_set <- imputed[train_index, ]
validation_set <- imputed[-train_index, ]
```

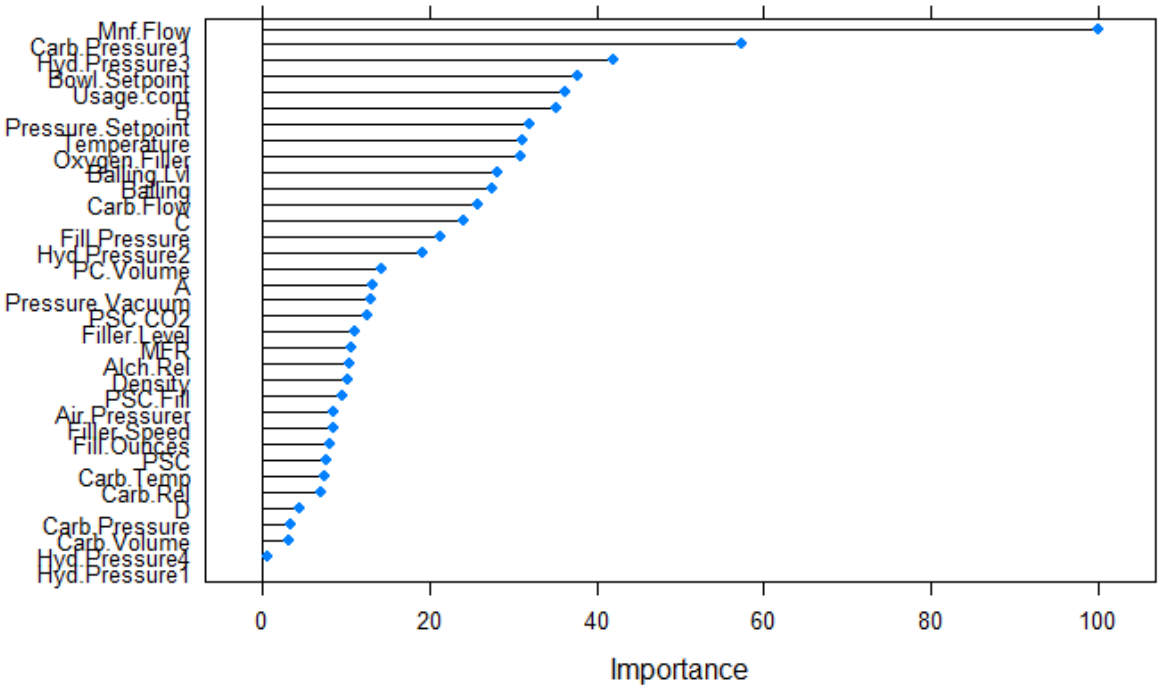
Linear Regression Models

Ordinary Linear Regression

Using the `train` function, we can preprocess the data using methods that would reduce the issues we've identified in the previous stages, namely outliers and skew.

```
set.seed(123)
lmTune <- train(PH ~ ., data=train_set, method='lm',
preProcess=c('BoxCox', 'center', 'scale'))


RMSE(pred = lmTune$finalModel$fitted.values, obs = train_set$PH)
R2(pred = lmTune$finalModel$fitted.values, obs = train_set$PH)

#Ordinary Linear Regression against holdout validation set
linP <- predict(lmTune, newdata = validX)
postResample(pred = linP, obs = validY$PH)[1:2]
```

After training the dataset, we get an RMSE of 0.1296 and an R2 of 0.422. The most predictive variables for this model are Mnf.Flow, CarbPressure1, Hyd.Pressure3, Bowl.Setpoint, Usage.cont, and B. Against the validation holdout set we get an RMSE of 0.1339 and an R2 of 0.4331.



Partial Least Squares

The optimal model using PLS shows an RMSE of 0.1468 and R^2 of 0.259. Against the validation holdout set we get an RMSE of 0.1548 and an R^2 of 0.235.

```
ctrl <- trainControl(method = "cv")

set.seed(123)
plsTune <- train(PH ~ ., data= train_set, method='pls',
preProcess=c('BoxCox', 'center', 'scale'), tuneLength=5,
trControl=ctrl)
plsTune


#PLS against holdout validation set
plsP <- predict(plsTune, newdata = validX)
postResample(pred = plsP, obs = validY$PH)[1:2]
```

Ridge-Regression

The optimal ridge-regression model shows an RMSE of 0.1325 and R^2 of 0.400. Against the validation holdout set we get an RMSE of 0.1342 and an R^2 of 0.430.

```
ridgeGrid <- data.frame(.lambda=seq(0, 0.1, length=15))
ridgeTune <- train(PH ~ ., data= train_set, method='ridge',
preProcess=c('BoxCox', 'center', 'scale'), tuneGrid=ridgeGrid,
trControl=ctrl)
ridgeTune

#Ridge Regression against holdout validation set
```

```
ridP <- predict(ridgeTune, newdata = validX)
postResample(pred = ridP, obs = validY$PH)[1:2]
```

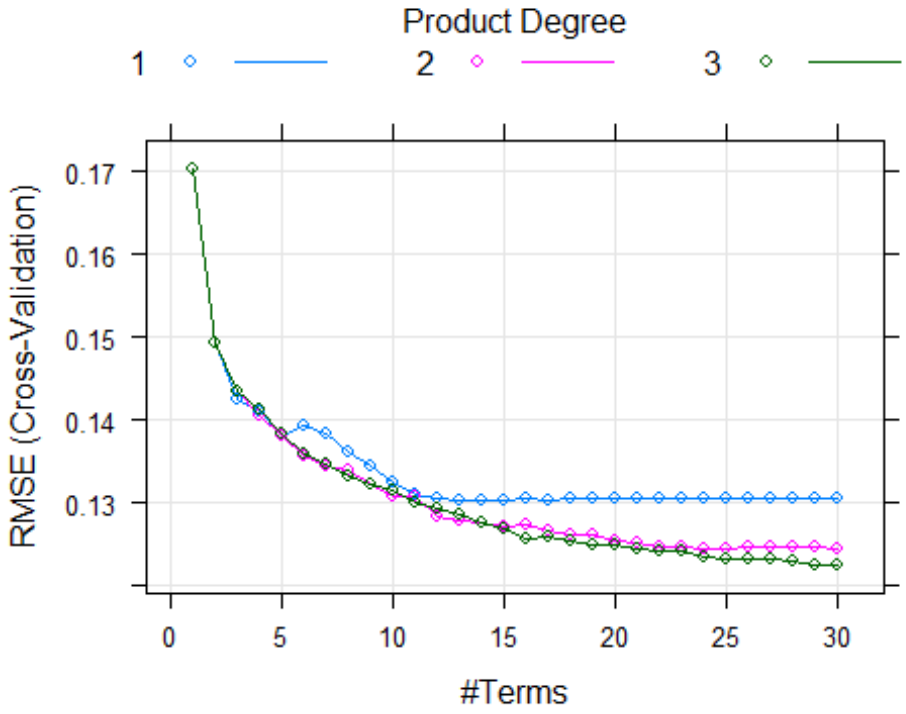
Nonlinear Regression Models

We recommend running the next several models using parallel processing; Code to do so in a windows environment is provided.

MARs

The optimal MARS model model shows an RMSE of 0.1224 and R^2 of 0.491. Against the validation holdout set we get an RMSE of 0.1257 and an R^2 of 0.495.

```
library(parallel)
library(doParallel)
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

indx <- createFolds(trainY$PH, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)

mars1 <- train(x = trainX, y = trainY$PH,
                  method = "earth",
                  tuneGrid = expand.grid(degree = 1:3,
                  nprune = 1:30),
                  trControl = ctrl)

#MARS against holdout validation set
marsP <- predict(mars1, newdata = validX)
postResample(pred = marsP, obs = validY$PH)[1:2]
```



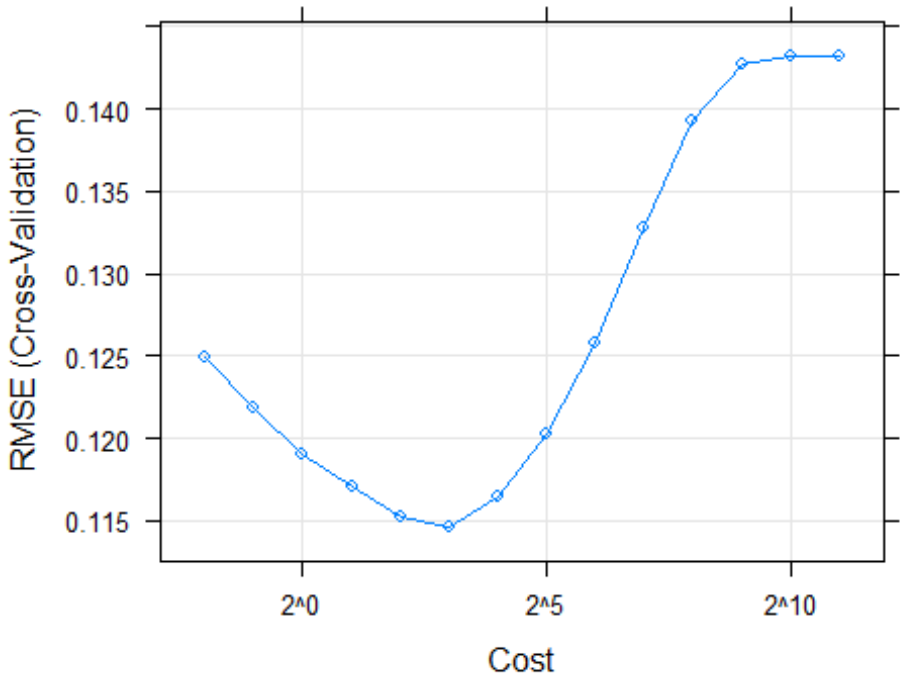
Support Vector Machines

The optimal SVM model shows an RMSE of 0.1146 and R^2 of 0.553. Against the validation holdout set we get an RMSE of 0.1199 and an R^2 of 0.549.

```
set.seed(123)
svmPTune <- train(x = trainX, y = trainY$PH,
                  method = "svmRadial",
                  preProc = c("center", "scale"),
                  tuneLength = 14,
                  trControl = trainControl(method = 'cv'))


#SVM against holdout validation set
postResample(pred = svmP, obs = validY$PH)[1:2]
svmP <- predict(svmPTune, newdata = validX)
```

The top five most important variables for the SVM Model are:

```
                  Overall
Mnf.Flow          100.000
Usage.cont         66.766
Bowl.Setpoint      54.498
Filler.Level       46.784
Pressure.Setpoint  43.739
```



Neural Network

The optimal Neural Network model shows an RMSE of 0.1125 and R^2 of 0.569. Against the validation holdout set we get an RMSE of 0.1161 and an R^2 of 0.571.

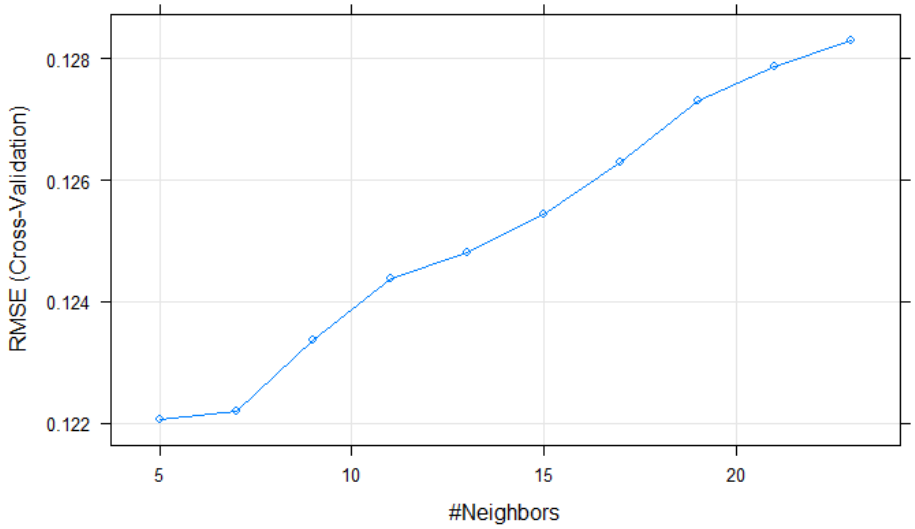
```
set.seed(100)
nnetTune <- train(x = trainX, y = trainY$PH,
                  method = "avNNet",
                  tuneGrid = nnetGrid,
                  trControl = ctrl,
                  preProc = c("center", "scale"),
                  linout = TRUE,
                  trace = FALSE,
                  maxit = 100,
                  allowParallel = TRUE)
```

K-Nearest Neighbors

The optimal model using K-Nearest Neighbors shows an RMSE of 0.1221 and R^2 of 0.496 The corresponding final value used for the model was k = 5. The most important predictor is Mnf.Flow which takes about 50% of the weight followed by Bowl.Setpoint, Filler.Lever, Usage.cont Pressure.Setpoint etc. For resample data, the end result for RMSE is 0.1234 and R^2 is 0.526.

```
set.seed(100)
knnTune <- train(PH ~ ., data=train_set, method = "knn",
preProcess = c('BoxCox', 'center', 'scale'), tuneLength = 10,
trControl = ctrl)
```

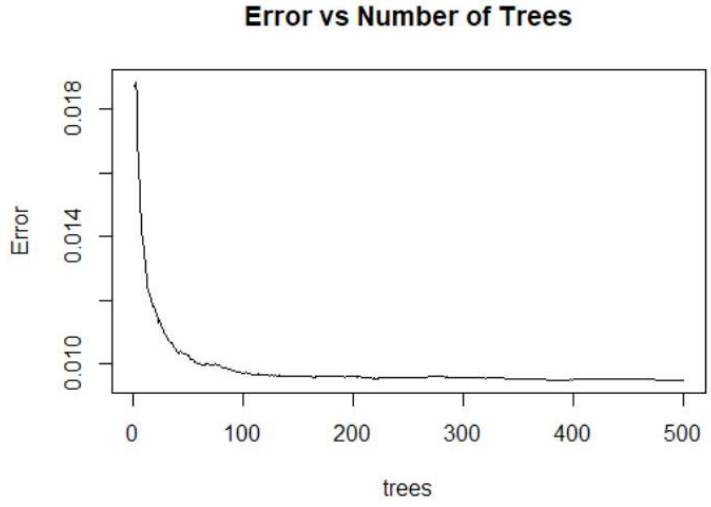
```
plot(varImp(knnTune), top = 10)
plot(knnTune)
knnTune

knnP <- predict(knnTune, newdata = validX)
postResample(pred = knnP, obs = validY$PH)
```



Regression Tree / Rule-Based Models

Regression Trees via Random Forest

The optimal model using K-Nearest Neighbors shows an RMSE of 0.100 and R^2 of 0.837. The tree count used was 500, as the approximate point of diminishing returns, beyond which prediction error does not appreciably decline. The relevant code is found beginning on the next page.



Impute the data with MissingForest, a Random Forests approach

```{r}
library(randomForest)

data <- read.csv("https://raw.githubusercontent.com/624-Group2/Project-2/master/StudentDataImputedMF.csv")
# Take 80% of data as training
rownums <- sample(1:nrow(data), round(nrow(data)*0.8))
train <- data[rownums, ]
test <- data[-rownums, ]

x <- train
x$PH <- NULL

# Get Best mTry parameter = 20
set.seed(1)
res <- tuneRF(x = x, y = train$PH, ntreeTry = 500, doBest = TRUE)
print(res)
```

Use Grid Search to identify optimal hyperparameters

```{r}
# Establish a list of possible values for mtry, nodesize and sampsize
mtry <- 20 #From doBest above; alternately, can use: seq(4, ncol(x) * 0.8, 2)
nodesize <- seq(4, 6, 1) # Originally seq(3,8,2), which returned value of 5 as optimal
sampsize <- 2054 # 80% more effective than 70%: round(nrow(x) * c(0.7, 0.8))

# Create a data frame containing all combinations
hyper_grid <- expand.grid(mtry = mtry, nodesize = nodesize, sampsize = sampsize)

# Create an empty vector to store OOB error values
var_exp <- c()

# Write a loop over the rows of hyper_grid to train the grid of models
for (i in 1:nrow(hyper_grid)) {

    # Train a Random Forest model
    model <- randomForest(formula = train$PH ~ .,
                          data = x,
                          mtry = mtry,
                          nodesize = hyper_grid$nodesize[i],
                          sampsize = sampsize)

    # Store OOB error for the model
    var_exp[i] <- model$rsq[500]
}

# Identify optimal set of hyperparmeters based on OOB error
```

```
opt_i <- which.max(var_exp)
print(hyper_grid[opt_i,])

# Results
mtry = 20
nodesize = 4
sampsize = 1796

# Create model off training set
modelRF <- randomForest(formula = PH ~ .,
                        data = train_set,
                        mtry = mtry,
                        nodesize = nodesize,
                        sampsize = sampsize)
modelRF
summary(modelRF)


plot(modelRF, main = "Error vs Tree Count")

predictRF <- predict(object = modelRF,
                     newdata = validX,
                     type = "response")

predDF <- data.frame(predRF=predictRF, actual = validY)
predDF$SE.RF <- (predDF$PH - predDF$predRF)^2
RMSE.RF <- sqrt(mean(predDF$SE.RF))
RMSE.RF
cor(predDF$PH, predDF$predRF)
```

Bagged Tree

A bagged tree via a Gradient Boosting Model was used to forecast PH. An RMSE of 0.1357 was obtained, corresponding to an R-squared value of 0.659 on the test set withheld from model-building. A gaussian distribution with 10,000 trees were selected as the hyper parameters.

```
library(gbm)
set.seed(10)
modelGBM <- gbm(formula = PH~.,
                data = train_set,
                distribution = "gaussian", # hist(data$PH) is
approx. normal
                n.trees = 10000)
modelGBM

# Predict test set values and add to predictions dataframe
predDF$predGBM <- predict(object = modelGBM,
                  newdata = validX,
                  n.trees = 10000,
                  type = "response")

predDF$SE.GBM <- (predDF$PH - predDF$predGBM)^2
RMSE.GBM <- sqrt(mean(predDF$SE.GBM)) #0.1278

cor(predDF$PH, predDF$predGBM)
```

Conclusion

As new regulation is looming, it is imperative for ABC Beverage to understand our manufacturing process.

We, as data scientists, built a predictive model for batches' PH level. The model was originated from a data set which contains 2571 observations across 32 variables. Out of three linear regression models, four non-linear regression models, and two regression tree/rule-based models, it has been determined that the Random Forest model provides the optimal resampling and test set performance. Random Forest model decorrelates the trees which are generated on bootstrapped samples. It reduces the variances by averaging the predictions..

The criteria we are using to compare models' performance is Root Mean Squared Error (RMSE). RMSE measures the standard deviation of the residuals (difference between predicted values and actual values). Random Forest model obtains the lowest value on RMSE of 0.100 among all predictive models

The Random Forest model is heavily weighted on Mnt.Flow, Brand Code, Usage,cont, Temperature, and Bowl.Setpoint predictors. It is finally used to predict evaluation data. The output for PH levels is saved as file *PredictionsRF.csv*.

Appendix: Project Prompt

This is role playing. I am your new boss. I am in charge of production at ABC Beverage and you are a data scientist 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 report in a Word readable format and your predictions in an Excel readable format.

I also rely on a colleague for advice. She is very data savvy and can provide info on good code form to me, and just make me feel better about a technical solution. Please provide all your code and technical dialogue so she can review it. She should be able to quickly cut and paste into R studio. NOTE, include R library calls in your code.

Questions? We can discuss. As always, I am busy (boss role, not professor) so I really want you to take the ball and run with it the best you can. But, I will answer as I can. Let's talk more in our weekly meeting next Tuesday.