DATA 624 Project 2

Predicting pH

Group 1: Jered Ataky, Matthew Baker, Christopher Bloom,

David Blumenstiel, Dhairav Chhatbar.

Problem Statement

You are given a simple data set from a beverage manufacturing company. It consists of 2,571 rows/cases of data and 33 columns / variables. Your goal is to use this data to predict PH (a column in the set). Potential for hydrogen (pH) is a measure of acidity/alkalinity, it must conform in a critical range and therefore it is important to understand its influence and predict its values. This is production data. pH is a KPI, Key Performance Indicator.

You are also given a scoring set (267 cases). All variables other than the dependent or target. You will use this data to score your model with your best predictions.

Background

The measure 'potential of hydrogen,' commonly denoted as pH, is a measure of how acidic or basic a solution is. Acids will raise the concentration of hydrogen and bases will lower the concentration of hydrogen. A lower pH measurement indicates an acidic solution while a higher pH measurement indicates a basic solution. The pH measurement scale is logarithmic, and a solution is typically considered to be 'neutral' (neither acidic nor basic) at pH = 7; the pH of water at a room temperature. pH is calculated as the $- \log_{10}[H^+]$, where $[H^+]$ is the concentration of hydrogen ions in a solution.

Being able to determine the pH of a solution has many applications within and outside the chemical field, such as agriculture, manufacturing, environmental science, medicine, biology, and nutrition. In solutions such as beverages, an acidic pH is important to prevent dangerous pathogens and bacteria such as botulism, and often has an appealing taste. Many household products also contain acids (lactic acid, citric acid, vinegar, etc) which can be functional in and of themselves, or make the product shelf-stable. While pH itself is important, other factors such as temperature during the packaging process must also be considered to prevent pathogens growing later. A combination of low pH, and high temperature

(pasteurization) and processing speed can help ensure a long shelf life for a slightly acidic food product. Other household products such as toothpaste, baking soda, and some household cleaners will measure basic (high pH). Obtaining an accurate pH measurement of a product is important, and be the key to a safe and acceptable manufacturing process and final product.

This analysis explores the possibility of predicting the pH of an unknown solution using 32 different variables within a manufacturing process. We attempt to compile a variety of machine learning models capable of pH prediction based on the given variables. Each machine learning algorithm will train on a dataset, weigh the importance of each variable, and make predictions of pH.

While no detailed legend is available, the predictor variables appear to be manufacturing related measurements of different volumes, pressures, temperatures, flows, and more. All of these measurements are germane to a resulting pH, so all of the variables will be investigated. We will select the best model based on it's predictive accuracy.

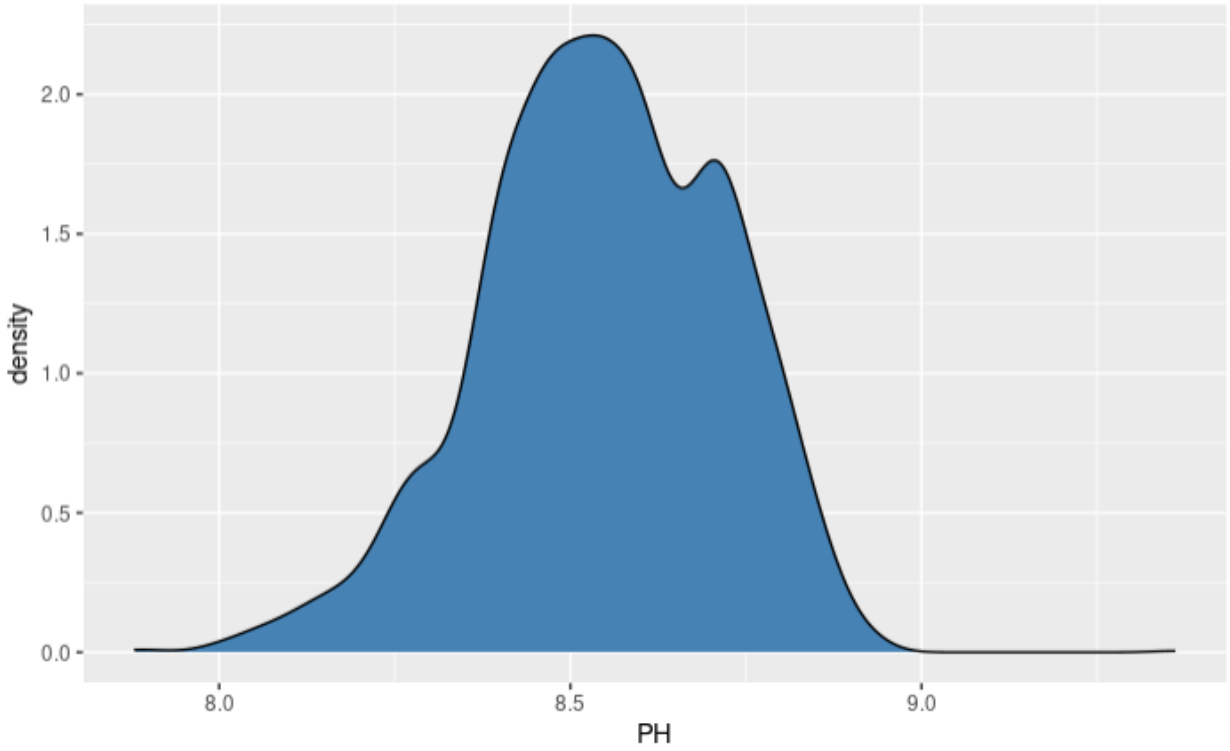
Preliminary Analysis & Data Preparation

The dataset contains 32 predictor variables and one response variable (pH) over 2571 separate observations. Before training any models the data was prepared, and some initial analysis was performed. Proper data preparation is important to optimizing the performance of machine learning models, and analysis can offer us insight into how the data needs to be prepared, and perhaps what to expect out of the models.

We only needed to do some basic data preparation. The data was originally contained in a .csv file, and was imported into R. One of the variables, brand, had a character data type (not numeric); it was converted to a categorical data type to aid in modeling. The other 32 variables were all, appropriately, numeric data. There were missing values throughout the dataset, however, less than 1% of the total dataset was missing, and no specific variable had more than 10% of it's data missing. Thus we were able to 'impute' (guess the missing values) using the predictive mean matching (a widely-used imputation method) via the 'mice' package in R. In addition, all data was centered and scaled (subtract mean, divide by standard deviation) prior to being fed into the machine learning algorithms; this aids in training the models.

As for preliminary data analysis, we first examined the distribution of pH values in the training set. The distribution of pH values in the training dataset (Figure 1) shows a near normal distribution (slight left skew) with a median pH of around 8.5. This indicates that the measured solutions are typically weakly basic. Given that pH is a logarithmic measurement, we considered whether or not it would be better to reshape the data to directly represent hydrogen ion concentration. However, the distribution of pH values was gaussian, which we decided was preferential; it was left as is. Having a normally distributed response variable is desirable for machine learning models, and can often improve performance.

Fig. 1 Distribution of pH Values in the Training Data



We also examined the distributions of each predictor variable (Figure 2). There were several different types of distributions, including many multimodal distributions. We ultimately decided not to do any transformations on these variables aside from centering and scaling in order to keep our process simpler, and to make it easier to directly interpret models where possible.

Fig. 2 Distributions of Predictor Variables

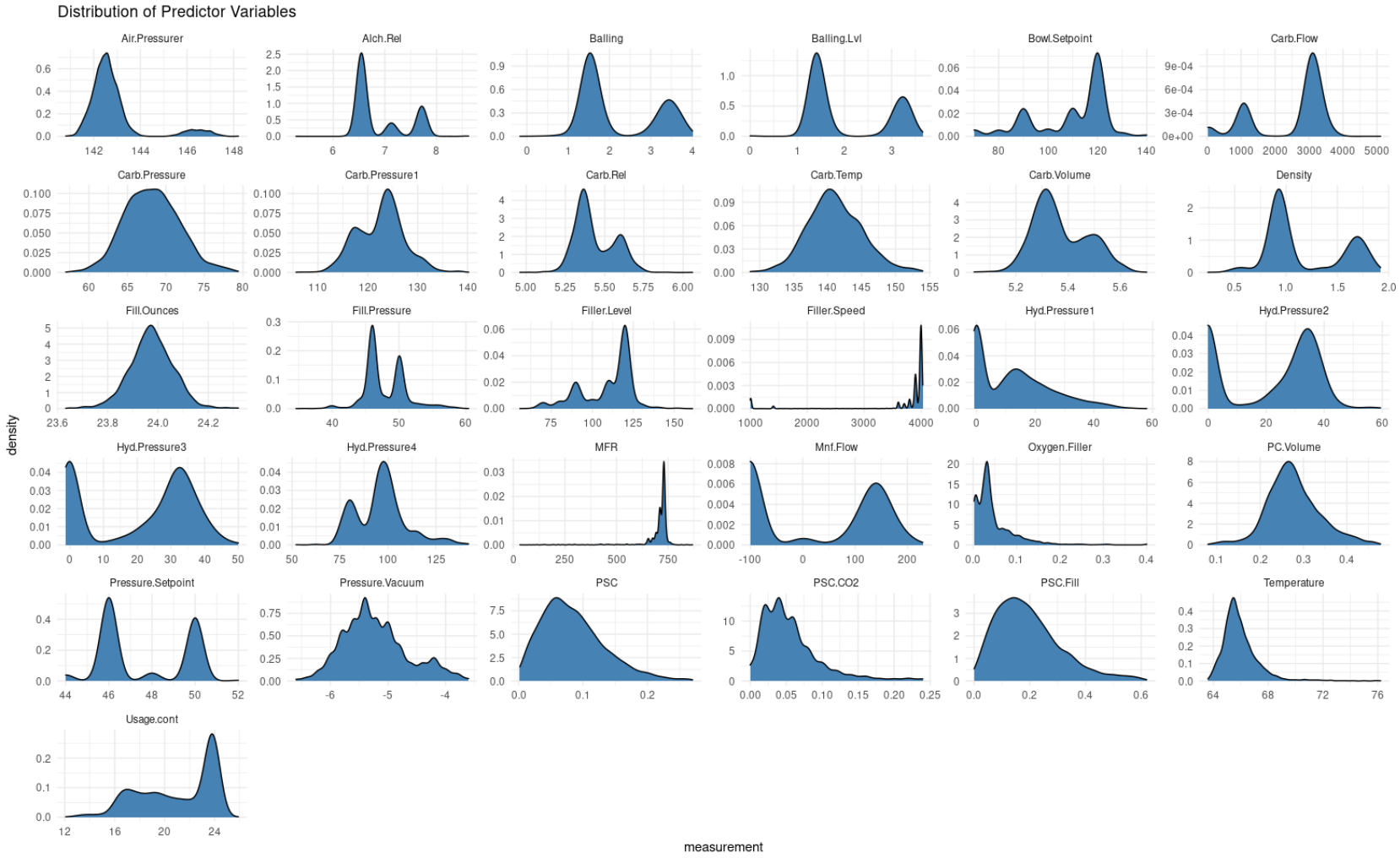
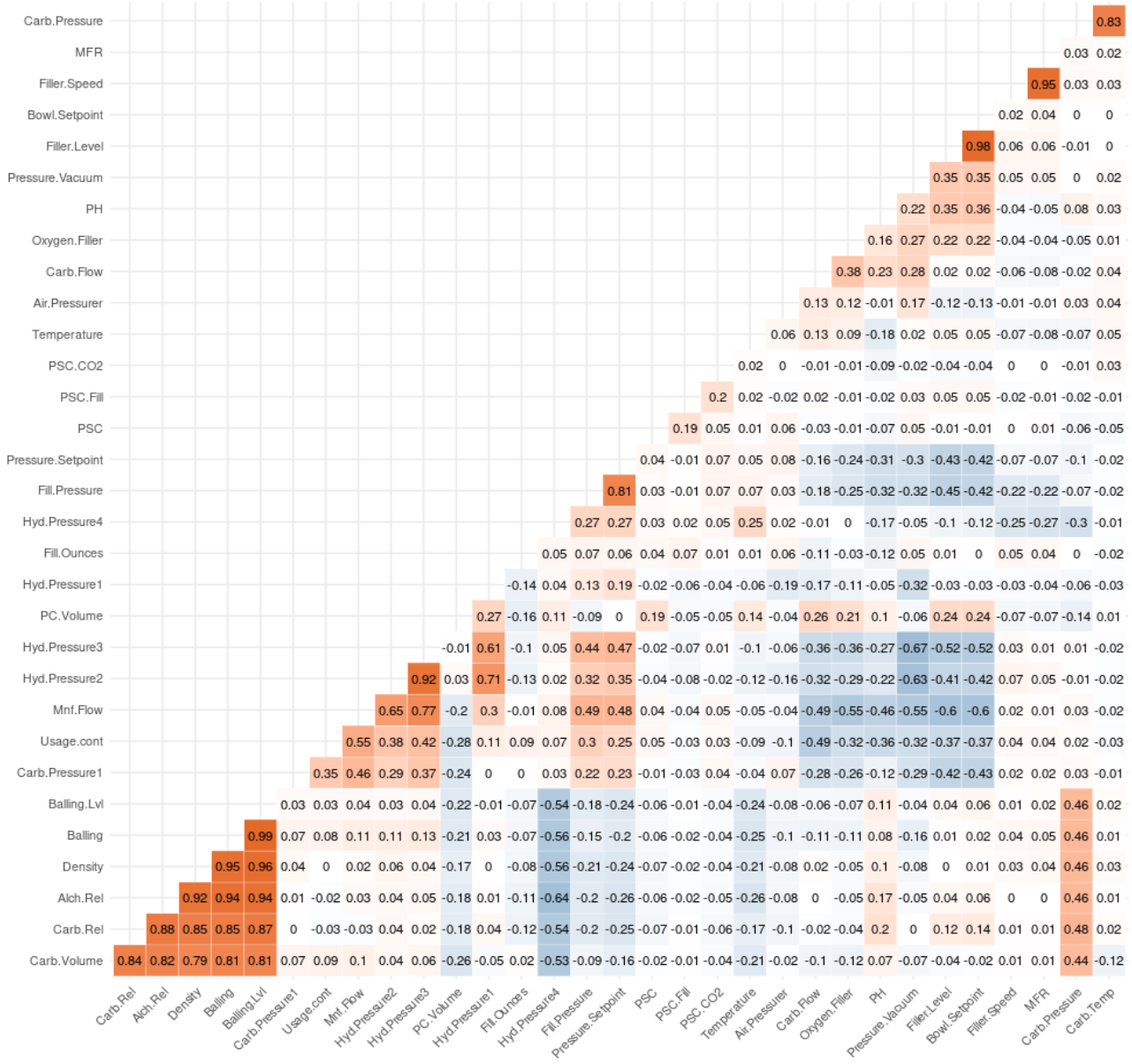


Fig. 3 Correlation Matrix of all variables including pH



We also briefly examine correlations between variables. The correlation map above (Figure 3) shows correlations between variables. The orange cells indicate positive correlation and the blue cells indicate negative correlation. The darker the color, the more correlated (either positively or negatively) two variables are. We want the predictor variables to be correlated to the response variable, but we do not want predictor variables to be correlated to each other. Generally, high correlation between different predictor variables can lead to problems for some machine learning models. Several of the predictor variables (look for dark cells in figure 3) are highly correlated. When training models, we'll have the option of either removing correlated variables, or using models that can deal with correlations between predictor variables (we mostly did the latter).

. There were, however, several notable correlations between the response variable (pH) and other variables. These include strong negative correlations between pH and the *Mnt. Flow (-0.46)* and *Usage.cont* (-0.36) variables, and strong positive correlations between pH and the *Filler.Level* (0.35) and *Bowl.Setpoint* (0.36) variables. It's likely that these variables will be of significance to the models.

Model Consideration and Analysis

Multiple different models were created to predict pH given the 32 different predictor variables, and potentially to understand the relationships between the predictors and pH. Ultimately, each trained model can be used to predict the pH of a solution where the predictor variables are known but the pH is not known. We considered many different types of supervised learning models, including a neural network, a support-vector machine, and several different types of decision tree based and linear models. Our goal in trying so many different types of models was to find one that worked the best, and use that one to make predictions on future data.

The dataset was split into training (80%) and testing (20%) sets; the training set is used to train the models, while the testing set is used to evaluate the models. It's important to use data that a model was not trained on when evaluating a model, so you have an idea of what it will do with data it has not seen before. Each of the models were tuned individually via grid-search, which automatically tries out different combinations of 'parameters'. Model parameters are basically the configurable, structural parts of a model, and models using different combinations of them need to be trained and tested separately to find the most appropriate ones. 5-fold cross-validation was also used for each model, which basically allows us to be more certain in terms of how well each model performs. Models were compared to one another using their performance on the test dataset.

The results of each tuned model (models using the best parameters) are measured by root mean square error (RMSE) and R-squared. RMSE is a standard method of measuring error when predicting a quantitative result, and more specifically measures the distance between the values predicted by the trained model and observed values, and has a minimum of 0 (no error).

R-squared is a measure of how closely the observed values are to the fitted value prediction, and maxes out at 1 (perfect predictions). Good models have high R-squared and low RMSE.

Model Results

Table 1. Model accuracy metrics.

Model type	RMSE	R-squared	Comments
Least Absolute Shrinkage and Selection Operator (LASSO)	0.128	0.403	
Elastic Net	0.128	0.402	
Multivariate Adaptive Regression Spline (MARS)	0.119	0.499	Overfitting probable
Partial Least Squares	0.128	0.402	
Support Vector Machine (SVM)	0.115	0.533	
Random Forest	0.0915	0.708	Best performer
Neural Network	0.116	0.508	
Cubist (M5)	0.0932	0.683	

eXtreme Gradient Boosting (XGB)	0.0964	0.663	
K-Nearest Neighbors (KNN)	0.124	0.450	

There were definitely some similarities between models. Most of the linear models (LASSO, Elastic - Net, Partial Least Squares) performed very similarly (R-squared around 0.4), with the exception of the MARS which did slightly better but raised concerns of overfitting (significantly worse performance on holdout set, presence of correlated predictors). The Elastic-net and LASSO models both employed very low lambda values, which lead to a wide selection of features, suggesting the models may have had trouble getting rid of variables without significant performance penalties.

All of the decision tree based models (Random Forest, Cubist, XGB) performed very well in comparison to the other models, with our Random Forest performing best of all. These models consistently performed well with R-squared values above 0.6. However, these took the longest to train, and are hard to interpret, but for our purposes they work very well.

The Neural Network, K-Nearest Neighbors, and SVM models all had R-squared values between those of the linear and decision tree models. Of these three models, the SVM performed the best with an R-squared of 0.533.

Because the goal of this project is more about making accurate predictions than it is interpreting the models, we use our best performing model, Random Forest, to make our final predictions. If instead the goal was discovering what affects the pH beverages, we might be more inclined to choose a model like LASSO or KNN, which don't produce as good results, but would allow us to better understand the influence of each predictor variable on pH.

Best Model Interpretation (Random Forest)

All of the decision tree models performed well, but the Random Forest model performed best considering a balance of low residuals and variance. This model used 1000 trees, and has 18 variables available for splitting at each node. Interpretation of models is based on three measurements focusing on residual measures, which are the differences between observed and predicted values. The RMSE score of *0.0915* measures the standard deviations of the residuals in the residuals. The R-squared score of *0.708* measures variance of the dependent variables in the trained model. Mean Absolute Error (MAE) score of *0.0657* measures an average of the

residual values. Relative to the other models, the Random Forest model has the best RMSE and R-squared, yielding accurate predictions.

Fig 4. Random forest predicted pH values vs Actual on test set.

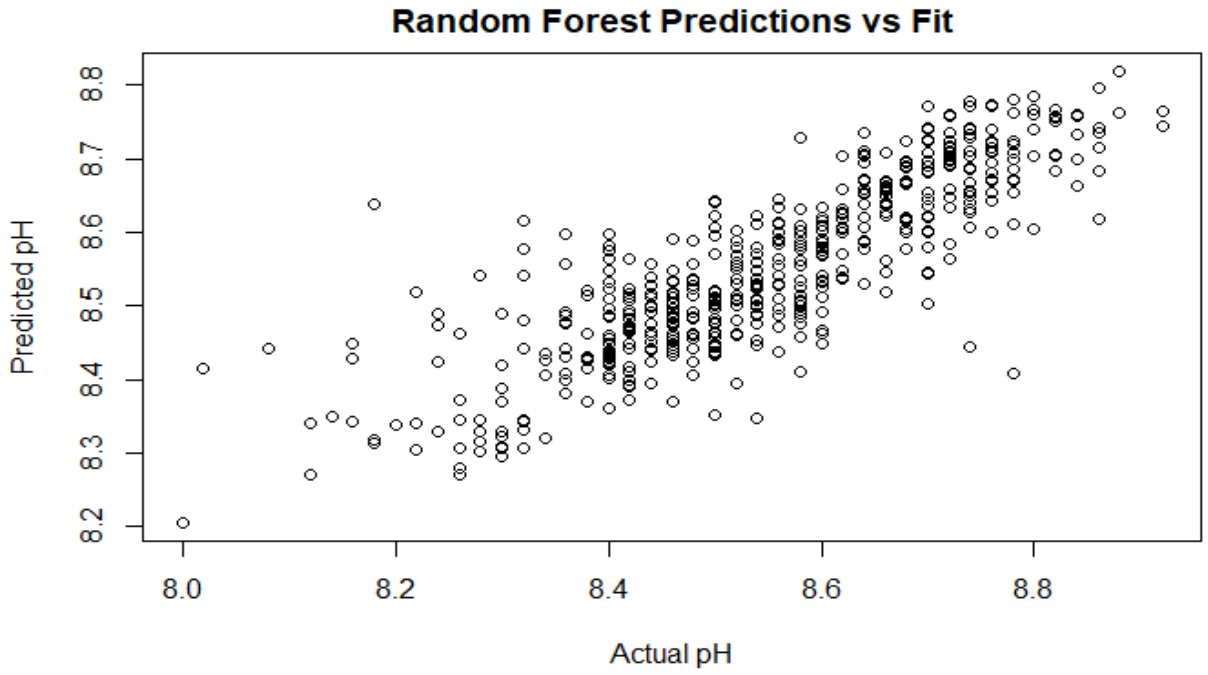
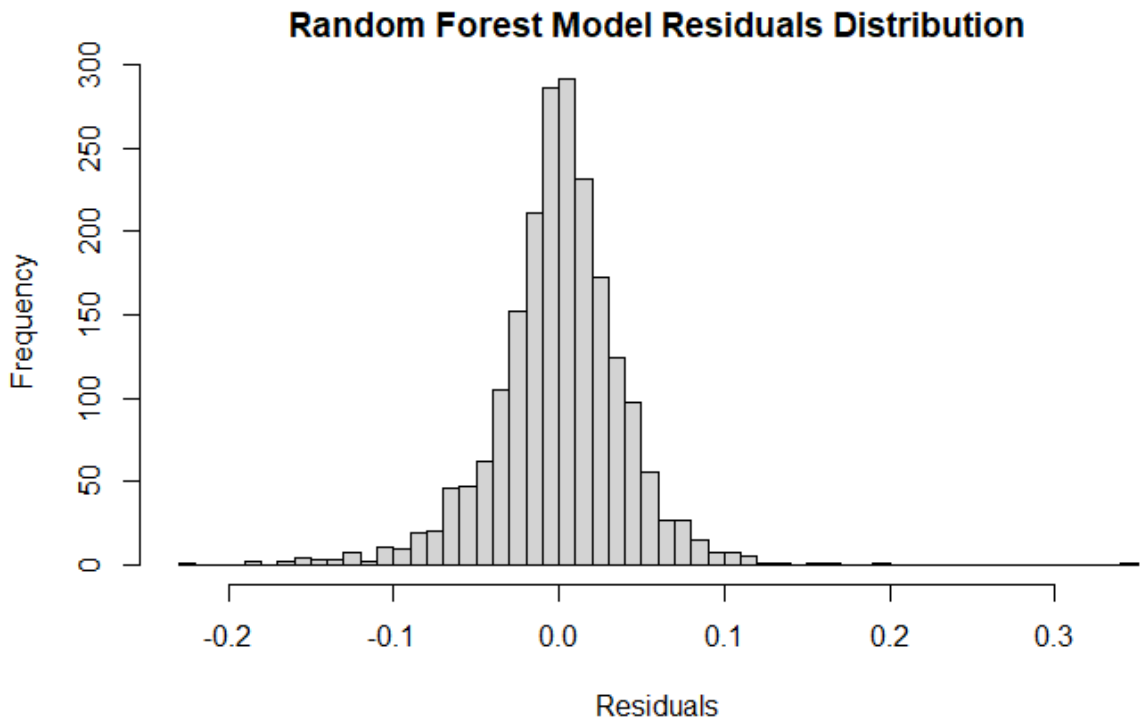


Figure 4 shows the predicted pH values against the actual pH values for each observation in the testing dataset, which was withheld from training. As the linear trend indicates, it is a fairly close fit. There is some heteroskedasticity, but considering the models strong performance on the testing set we aren't too worried. Figure 5 shows the distribution of residuals for the model, illustrating that the distribution is fairly Gaussian.

Random Forest is a type of ensemble learning (multiple algorithms technique) method that uses multiple decision trees, an analysis technique which splits variables and results into nodes and branches. The random forest method returns the average prediction of the generated trees, and iterates the process to prevent overfitting to the training set. It has the advantage of being somewhat easy to implement and tune, and often performs very well, but is something of a black box when it comes to interpretation. With respect to the random forest model, it's difficult to interpret the trained model to gain insight into its learned relationships between variables and outcomes, however the tradeoff is good accuracy. We are not as concerned with interpretability in this case as we are model accuracy.

Fig 5. The distribution of residuals for the Random Forest model.



However, we can impute basic feature importances, a relative measurement of how much each variable affects the predictions. The five most important variables in this model are Mnf.Flow (100%), Usage.count (43%), Brand C (one brand, 36%), Oxygen.Filler (25%), and Bowl.Setpoint (24%). This makes some sense because several of these are significantly correlated, inversely or otherwise, to pH (see Fig 3). Given the nature of Random Forest models, it is somewhat difficult to quantitatively assess how much each of these variables affect pH.

Conclusions

Our Random Forest model performed the best out of all models we tried, and should be well suited for predicting pH of beverages similar to those in the dataset we use. However, given the nature of Random Forests, the model is difficult to interpret at a fine level and a change in the available variables would likely significantly affect the model; we don't have simple coefficients to go with. It also bears mentioning that these models were trained on a dataset consisting of fairly alkaline beverages (median pH around 8.5). This model might not be applicable to most beverage manufacturing (which tends to run acidic), even given a similar process

The applicability could be useful for screening safety and standards of household product manufacturing, as well as industrial chemicals that are potentially released into the environment or local ecosystems.

Appendix

Below are the code for the project:

Libraries used

```{r}
library(dplyr)
library(readxl)
library(skimr)
library(tidyr)
library(kableExtra)
library(mice)
library(VIM)
library(corrplot)
library(ggcorrplot)
library("caret")
library("glmnet")
library("earth")
library("pls")
library("e1071")
library("xgboost")
```

Data Preparation

```{r}

train_dataset <-
read.csv("https://raw.githubusercontent.com/jnataky/Predictive_Analytics/main/Project2/Student Data%20-%20TO%20MODEL.csv")

evaluation_dataset <-
read.csv("https://raw.githubusercontent.com/jnataky/Predictive_Analytics/main/Project2/Student Evaluation-%20TO%20PREDICT.csv")

evaluation_dataset$PH <- NULL  #Having an empty column get's in the way for most of this

head(train_dataset)

```

Exploratory Data Analysis

Now we want to analyse the data. See the report for a detailed analysis; this produces some figures which are helpful.

```{r}

skim(train_dataset)

aggr(train_dataset, col=c('#F8766D','#00BFC4'), numbers=TRUE, sortVars=TRUE,
labels=names(train_dataset), cex.axis=.7, gap=3, ylab=c("Missing data","Pattern"))

train_dataset %>% select(PH) %>% ggplot( aes(PH)) + geom_density(fill="steelblue", bins = 30)

train_dataset %>% select(-PH, -Brand.Code) %>%
  tidyr::gather(key = "variable", value = "measurement",everything()) %>%
  arrange(desc(variable)) %>%
  ggplot(aes(measurement)) + geom_density(position=position_dodge(), fill="steelblue") +
  facet_wrap(~variable, scales = "free") +
  theme_minimal() +
  ggtitle("Distribution of Predictor Variables")
q <- cor(train_dataset%>%select(-Brand.Code), use = "na.or.complete")
ggcorrplot(q, type = "lower", outline.color = "white", hc.order = TRUE,
           colors = c("#6D9EC1", "white", "#E46726"),
           lab = TRUE, show.legend = FALSE, tl.cex = 8, lab_size = 3)

```

Data Transformation

Below will impute missing data via pmm using the mice package, and rename the first column and change it to factor

```{r}
preprocess <- function(df) {
  colnames(df)[1] <- "Brand"  #Changes name of first column to something less obtuse
  df$Brand <- as.factor(df$Brand)#Changes brand to factor

  #Uses MICE for imputation of missing values.  Going with mostly defaults.

  imputed <- mice(df,
           m = 5,
           maxit = 5,
           seed = 10,
           trace = FALSE)

  df <- complete(imputed)
```

```
  return(df)
}
train_dataset <- preprocess(train_dataset)
evaluation_dataset <- preprocess(evaluation_dataset)
```

Modeling

R has this really nice package called Caret, which gives us a function that allows us to easily train a bunch of models with only minor adjustments. It can search for optimal parameters, take manual specifications, you name it. We'll use it to train all of the models.

But, first, we need to split off a test set to judge performance. We'll do 80:20 train/test

```{r}
set.seed(1234567890) #So you see the same thing I do
splitdex <- createDataPartition(train_dataset$PH, p = 0.8, list = FALSE) #Index for split
train <- train_dataset[splitdex,]
test <- train_dataset[-splitdex,]

```

LASSO

We can probably do away with a lot of these values and get a simpler model. LASSO, a penalized model, will aim to do so. We'll implement this with caret for simplicity.

```{r}

set.seed(1234567890) #So you see the same thing I do

```

#got help from: http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/

#and: https://stackoverflow.com/questions/57712116/regarding-preprocessing-in-lasso-using-caret-package-in-r

#The parameters we want it to try

```{r}

grid <- expand.grid(alpha = 1,
                    lambda =10^seq(-5, 5, length = 1000)) #Lot's of lambda values to choose from
```

```
#Adds cross validation
tc = trainControl(method = "cv",   #Cross-validation
                  number = 10)
#Fit's the model
lassoFit <- caret::train(PH ~ ., data = train,
                  method = "glmnet",   #glmnet lets us fit penalized maximum likelihood glms (lasso, ridge, elastic)
                  preProcess = c("center", "scale"), #Data needs to be centered and scaled for this
                  tuneGrid = grid,
                  trControl = tc)
lassoFit$bestTune
coef(lassoFit$finalModel,lassoFit$bestTune$lambda)
```

#Makes predictions

```{r}
lasso_predictions <- predict(lassoFit, test)

#Calculates RMSE and R2

print(paste("RMSE: ", RMSE(lasso_predictions, test$PH), "     R2: ", caret::R2(lasso_predictions, test$PH)))
```

Performance wise, looks fairly good. Let's compare it to some other models.

Elastic-Net

Easy enough to try out one of these, having already made the LASSO model. For this, we just need to give some values of alpha to choose from.

```{r}
set.seed(1234567890) #So you see the same thing I do
```

#got help from: http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/153-penalized-regression-essentials-ridge-lasso-elastic-net/

#and: https://stackoverflow.com/questions/57712116/regarding-preprocessing-in-lasso-using-caret-package-in-r

```{r}

tc = trainControl(method = "cv",   #Cross-validation

                  number = 10)

elasticFit <- caret::train(PH ~ ., data = train,

                  method = "glmnet",   #glmnet lets us fit penalized maximum likelihood glms (lasso, ridge, elastic)

                  preProcess = c("center", "scale"), #Data needsa to be centered and scaled for this

                  tuneLength = 20, #Going to let it choose alpha and lambda without any sugestions

                  trControl = tc)

elasticFit$bestTune

coef(elasticFit$finalModel,elasticFit$bestTune$lambda)

```

It chose a similarly small lambda as the LASSO model. Let's see how it performs on the test set.

```{r}

elastic_predictions <- predict(elasticFit, test)

print(paste("RMSE: ", RMSE(elastic_predictions, test$PH), "    R2: ", caret::R2(elastic_predictions, test$PH)))

```

Does only slightly better than LASSO.

###MARS

Let's try one of these. I'm a bit concerned that the collinear variables could lead to overfitting, but we can test things out on the validation set.

```{r}

set.seed(1234567890)
```

```
#got help from here: https://bradleyboehmke.github.io/HOML/mars.html

tc = trainControl(method = "cv",   #Cross-validation

                  number = 10)

#The parameters we want it to try

grid <- expand.grid(degree = 4:7,   #automatic tune won't let you try more than one of these

                    nprune = seq(60, 150, length = 10))

marsFit <- caret::train(PH ~ ., data = train,

                        method = "earth",   #earth has mars model.  I know how that sounds

                            preProcess = c("center", "scale"), #why not

                        tuneGrid = grid,

                        trControl = tc)

marsFit$bestTune
```

An initial run showed better performance with the highest degree and number of tunes used. Above, the model was re-trained using higher degree and term numbers (not including the original set to speed this up).

It found 60 terms and 5 degrees to have the best performance., with an RMSE and R2 around 0.12 and 0.52 respectively. Let's evaluate the validation set.

#Makes predictions

```{r}

mars_predictions <- predict(marsFit, test)

#Get's RMSE and R2

print(paste("RMSE: ", RMSE(mars_predictions, test$PH), " R2: ", caret::R2(mars_predictions, test$PH)))
```

```

*It seems the concerns over overfitting were valid.  The RMSE and R2 are not nearly as good when tested on the validation set.*

### Partial Least Squares

*This should handle multicollinearity better, and should do well considering the number of variables.*

```{r}

set.seed(1234567890) #so you see what I do

#took help from here: http://www.sthda.com/english/articles/37-model-selection-essentials-in-r/152-principal-component-and-partial-least-squares-regression-essentials/

tc = trainControl(method = "cv",   #Cross-validation

                  number = 10)

#This one automatically tries different parameters

plsFit <- caret::train(PH ~ ., data = train,

                       method = "pls",   # from the "pls" package

                       preProcess = c("center", "scale"),

                       tuneLength = 32,  #Try different parameters.  Max in this case is nvar - 1

                       trControl = tc)

plsFit

```

Let's try it out:

```{r}

pls_predictions <- predict(plsFit, test)

print(paste("RMSE: ", RMSE(pls_predictions, test$PH), "     R2: ", caret::R2(pls_predictions, test$PH)))
```
```

```

Does just about the same.

### SVM (for regression)

Let's try a SVM for regression (not classification)

```{r}

set.seed(1234567890) #so you see what I do

#took help from here: https://stackoverflow.com/questions/49543307/svm-with-radial-kernel-for-numeric-response-in-caret-package

tc = trainControl(method = "cv",   #Cross-validation

                  number = 5)

#Will also try to adjust parameters automatically

svmFit <- caret::train(PH ~ ., data = train,

                       method = "svmRadial",   # from the "e1071" package

                       preProcess = c("center", "scale"),

                       tuneLength = 10,

                       trControl = tc)

svmFit

```

Let's see how it does on the validation set.

```{r}

svm_predictions <- predict(svmFit, test)

print(paste("RMSE: ", RMSE(svm_predictions, test$PH), "  R2: ", caret::R2(svm_predictions, test$PH)))

```
```

It does particularly well.

Random Forest

```{r}

set.seed(1234567890) #so you see what I do

#took help from here:

tc <- trainControl(method = "cv",   #Cross-validation

                   number = 5)

```

#Because this takes a long time to train, the values below were pre-selected as the best tune from several runs.

```{r}

grid <- expand.grid(.mtry = c(18))

rfFit <- caret::train(PH ~ ., data = train,
                      method = "rf",
                      preProcess = c("center", "scale"),
                      tuneGrid = grid,
                      trControl = tc,
                      ntrees = 1000)
rfFit

```

Promising. I tried several combinations of metrics and ntrees, and above did just as well in a shorter time.

```{r}

rf_predictions <- predict(rfFit, test)

print(paste("RMSE: ", RMSE(rf_predictions, test$PH), "       R2: ", caret::R2(rf_predictions, test$PH)))

```

The best model so far. better on the testing set than the training set

Neural Network

Not particularly transparent, but might get the job done.

```{r}
set.seed(1234567890) #so you see what I do
#took help from here:
tc <- trainControl(method = "cv",   #Cross-validation
                   number = 5)
#Tries a combinations of the parameters below
grid <- expand.grid(decay = 10^seq(-3,-0.5,length = 10),
                    size = 1:5)
nnFit <- caret::train(PH ~ ., data = train,
                      method = "nnet",
                      preProcess = c("center", "scale"),
                      linout=TRUE,
                      trace = FALSE,
                      tuneGrid = grid,
                            trControl = tc)
nnFit
```

Selected some values right in the middle, but ultimately doesn't look as promising as random forest or svm.

```{r}
nnet_predictions <- predict(nnFit, test)
print(paste("RMSE: ", RMSE(nnet_predictions, test$PH), "  R2: ", caret::R2(nnet_predictions, test$PH)))
```

```

Decent, but not the best so far.

### Cubist (M5)

A strange take on decision trees.  Random forest did well; maybe this will build on that?

```{r}

set.seed(1234567890) #so you see what I do

#took help from here:

tc <- trainControl(method = "cv",   #Cross-validation

                   number = 5)

#Another case where tuning was done separately due to long runtimes.  The values below are the best selections for both parameters.

grid = expand.grid(committees = 100,

                    neighbors = 8)

cubistFit <- caret::train(PH ~ ., data = train,

                      method = "cubist",

                      preProcess = c("center", "scale"),

                      tuneGrid = grid,

                      trControl = tc)

cubistFit

```

I trained this with additional parameter combinations to those tried above (not included in case you want to run this in under half an hr). In the above attempt, the best combination was found

```{r}

cubist_predictions <- predict(cubistFit, test)

print(paste("RMSE: ", RMSE(cubist_predictions, test$PH), "     R2: ", caret::R2(cubist_predictions, test$PH)))
```
```

```

Just about on par with the random forest

### eXtreme Gradient Boosting

Saw on list, Sounded cool.  Gradient boosted decision trees.

```{r}

set.seed(1234567890) #so you see what I do
#took help from here:
tc <- trainControl(method = "cv",   #Cross-validation
                   number = 5)
#Tries a lot of different parameters.  These were narrowed down in separate training sessions.
grid <- expand.grid(lambda = c(1.39 * 10^-5, 10^-5, 1.90 * 10^-5),
                    alpha = c(0.0001,0.00046, 0.001),
                    nrounds = c(80,90,100),
                    eta = 0.3)
xgbFit <- caret::train(PH ~ ., data = train,
                       method = "xgbLinear",
                       preProcess = c("center", "scale"),
                       tuneGrid = grid,
                       trControl = tc)
xgbFit

```

parameters were tuned individually. The last iteration of training is shown above. This model seems to perform well, but let's double check on the test set.

```{r}

xgb_predictions <- predict(xgbFit, test)

print(paste("RMSE: ", RMSE(xgb_predictions, test$PH), "   R2: ", caret::R2(xgb_predictions, test$PH)))

```

It does very well.

K-Nearest Neighbors

Another basic but good one.

```{r}

```
set.seed(1234567890) #so you see what I do

#took help from here:

tc <- trainControl(method = "cv",   #Cross-validation

                   number = 5)

#Very simple to tune these types of models. K is usually like 5

grid <- expand.grid(k = 1:10)

knnFit <- caret::train(PH ~ ., data = train,

                       method = "knn",

                       preProcess = c("center", "scale"),

                       tuneGrid = grid,

                       trControl = tc)

knnFit

knn_predictions <- predict(knnFit, test)

print(paste("RMSE: ", RMSE(knn_predictions, test$PH), "   R2: ", caret::R2(knn_predictions,
test$PH)))
```

Not bad, but not the best.

Best Model Analysis

Important variables

```{r}
varImp(rfFit)

rfFit
```

Residuals

Let's plot out the residuals

```{r}

plot(rf_predictions ~ test$PH, main = "Random Forest Predictions vs Fit", #Plot's predictions vs actual

     xlab = "Actual pH",

     ylab = "Predicted pH")

hist(residuals(rfFit), # Histogram of residuals

     breaks = 50,

     xlab = "Residuals",

     main = "Random Forest Model Residuals Distribution")

```

Forecasting & Conclusion

We'll go to a random forest. It performed among the best on the test set.

```{r}

#Makes predictions

evaluation_dataset$PH <- predict(rfFit, evaluation_dataset)

#Saves them to wherever you downloaded this

write.csv(evaluation_dataset$PH, "Group1_Project2_Predictions.csv")

```