Machine Learning Final Project

By: Michael Zeolla, Andrew Fisher, Marko Vila, and Reily Siegel

Introduction

For this project, the group analyzed the Red Wine Quality dataset provided. The main goal of the analysis was to properly predict if a specific wine could be classified into "Good" wine or "Bad" wine. The outcome would be a binary classification for the dataset entries. To accomplish this a wide range of analytical models and methods was implemented, such as Logistic Regression, K Nearest Neighbors (KNN), Quantitative Descriptive Analysis (QDA), and Neural Networks. The results from each prediction model can be seen below.

Exploratory Data Analysis (EDA)

First, the data is loaded into a programming IDE to be analyzed. For this section the language was R, but that is not common throughout the presentation. The CSV file is loaded from the computer, and then the data heads are printed to confirm the data was correctly imported. The seed is also set so that the results can be obtained upon reruns.

```
library(corrplot)
library(caret)
library(class)

data_main = read.table("C://Users//mjzeo//OneDrive//Desktop/School//B
Term 2021-2022//CS4342//Term
Project//winequality-red.csv",sep=",",header=T)
head(data_main)
set.seed(19)
```

```
> head(data_main)
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide total.sulfur.dioxide density
1           7.4             0.70        0.00            1.9     0.076                  11                   34  0.9978
2           7.8             0.88        0.00            2.6     0.098                  25                   67  0.9968
3           7.8             0.76        0.04            2.3     0.092                  15                   54  0.9970
4          11.2             0.28        0.56            1.9     0.075                  17                   60  0.9980
5           7.4             0.70        0.00            1.9     0.076                  11                   34  0.9978
6           7.4             0.66        0.00            1.8     0.075                  13                   40  0.9978
     pH sulphates alcohol quality
1 3.51      0.56     9.4       5
2 3.20      0.68     9.8       5
3 3.26      0.65     9.8       5
4 3.16      0.58     9.8       6
5 3.51      0.56     9.4       5
6 3.51      0.56     9.4       5
```

Figure 1: Head Representation of the Data

The summary() function in R can report important values for the dataset. To get the perfect value in which to predict the difference between "Good" and "Bad" wine this function can be used to calculate the median and mean of the quality dataset. Furthermore, the table()

function can produce how many different qualities there are in the data and the number of wines that belong to each quality grouping.

```
summary(data_main$quality)
table(data_main$quality)
```

```
> summary(data_main)
 fixed.acidity   volatile.acidity  citric.acid    residual.sugar     chlorides       free.sulfur.dioxide
 Min.   : 4.60   Min.   :0.1200   Min.   :0.000   Min.   : 0.900   Min.   :0.01200   Min.   : 1.00
 1st Qu.: 7.10   1st Qu.:0.3900   1st Qu.:0.090   1st Qu.: 1.900   1st Qu.:0.07000   1st Qu.: 7.00
 Median : 7.90   Median :0.5200   Median :0.260   Median : 2.200   Median :0.07900   Median :14.00
 Mean   : 8.32   Mean   :0.5278   Mean   :0.271   Mean   : 2.539   Mean   :0.08747   Mean   :15.87
 3rd Qu.: 9.20   3rd Qu.:0.6400   3rd Qu.:0.420   3rd Qu.: 2.600   3rd Qu.:0.09000   3rd Qu.:21.00
 Max.   :15.90   Max.   :1.5800   Max.   :1.000   Max.   :15.500   Max.   :0.61100   Max.   :72.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol         quality
 Min.   :  6.00       Min.   :0.9901   Min.   :2.740   Min.   :0.3300   Min.   : 8.40   Min.   :3.000
 1st Qu.: 22.00       1st Qu.:0.9956   1st Qu.:3.210   1st Qu.:0.5500   1st Qu.: 9.50   1st Qu.:5.000
 Median : 38.00       Median :0.9968   Median :3.310   Median :0.6200   Median :10.20   Median :6.000
 Mean   : 46.47       Mean   :0.9967   Mean   :3.311   Mean   :0.6581   Mean   :10.42   Mean   :5.636
 3rd Qu.: 62.00       3rd Qu.:0.9978   3rd Qu.:3.400   3rd Qu.:0.7300   3rd Qu.:11.10   3rd Qu.:6.000
 Max.   :289.00       Max.   :1.0037   Max.   :4.010   Max.   :2.0000   Max.   :14.90   Max.   :8.000

> table(data_main$quality)

  3   4   5   6   7   8
 10  53 681 638 199  18
```

Figure 2: Summary and Table of the Data

Based on the results it is clear there are only 6 different quality groupings despite there being values allowed between 3 and 10. The largest groupings are 5 and 6 which hold the majority of the data entries. The best split of the data is likely 5.5, which is around the mean of the quality predictor.

The next best step would be to analyze the correlation between each of the predictors and the outcome. Depending on the correlation of the data specific predictors can be removed to produce a more accurate model.

```
corrplot(cor(data_main))
cor(data_main)
```

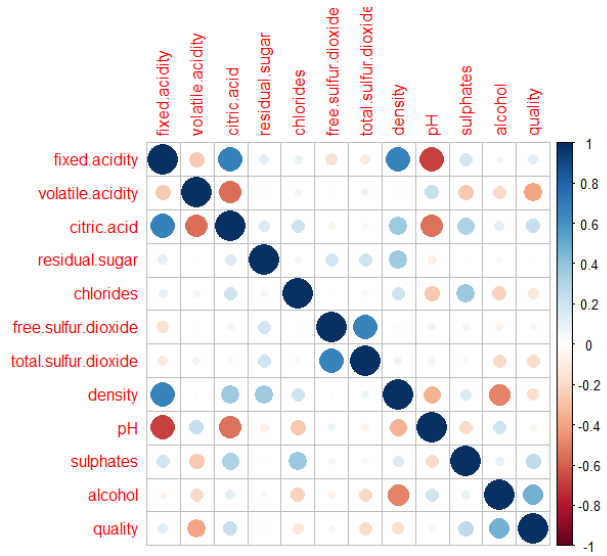


Figure 3: Correlation of the Data in a Plot

```
> cor(data_main)
                      fixed.acidity volatile.acidity citric.acid residual.sugar    chlorides free.sulfur.dioxide
fixed.acidity          1.00000000      -0.256130895  0.67170343    0.114776724  0.093705186        -0.153794193
volatile.acidity      -0.25613089       1.000000000 -0.55249568    0.001917882  0.061297772        -0.010503827
citric.acid            0.67170343      -0.552495685  1.00000000    0.143577162  0.203822914        -0.060978129
residual.sugar         0.11477672       0.001917882  0.14357716    1.000000000  0.055609535         0.187048995
chlorides              0.09370519       0.061297772  0.20382291    0.055609535  1.000000000         0.005562147
free.sulfur.dioxide   -0.15379419      -0.010503827 -0.06097813    0.187048995  0.005562147         1.000000000
total.sulfur.dioxide  -0.11318144       0.076470005  0.03553302    0.203027882  0.047400468         0.667666450
density                0.66804729       0.022026232  0.36494718    0.355283371  0.200632327        -0.021945831
pH                    -0.68297819       0.234937294 -0.54190414   -0.085652422 -0.265026131         0.070377499
sulphates              0.18300566      -0.260986685  0.31277004    0.005527121  0.371260481         0.051657572
alcohol               -0.06166827      -0.202288027  0.10990325    0.042075437 -0.221140545        -0.069408354
quality                0.12405165      -0.390557780  0.22637251    0.013731637 -0.128906560        -0.050656057
                      total.sulfur.dioxide     density          pH    sulphates     alcohol     quality
fixed.acidity              -0.11318144  0.66804729 -0.68297819  0.183005664 -0.06166827  0.12405165
volatile.acidity            0.07647000  0.02202623  0.23493729 -0.260986685 -0.20228803 -0.39055778
citric.acid                 0.03553302  0.36494718 -0.54190414  0.312770044  0.10990325  0.22637251
residual.sugar              0.20302788  0.35528337 -0.08565242  0.005527121  0.04207544  0.01373164
chlorides                   0.04740047  0.20063233 -0.26502613  0.371260481 -0.22114054 -0.12890656
free.sulfur.dioxide         0.66766645 -0.02194583  0.07037750  0.051657572 -0.06940835 -0.05065606
total.sulfur.dioxide        1.00000000  0.07126948 -0.06649456  0.042946836 -0.20565394 -0.18510029
density                     0.07126948  1.00000000 -0.34169933  0.148506412 -0.49617977 -0.17491923
pH                         -0.06649456 -0.34169933  1.00000000 -0.196647602  0.20563251 -0.05773139
sulphates                   0.04294684  0.14850641 -0.19664760  1.000000000  0.09359475  0.25139708
alcohol                    -0.20565394 -0.49617977  0.20563251  0.093594750  1.00000000  0.47616632
quality                    -0.18510029 -0.17491923 -0.05773139  0.251397079  0.47616632  1.00000000
```

Figure 4: Correlation of the Data Numerically

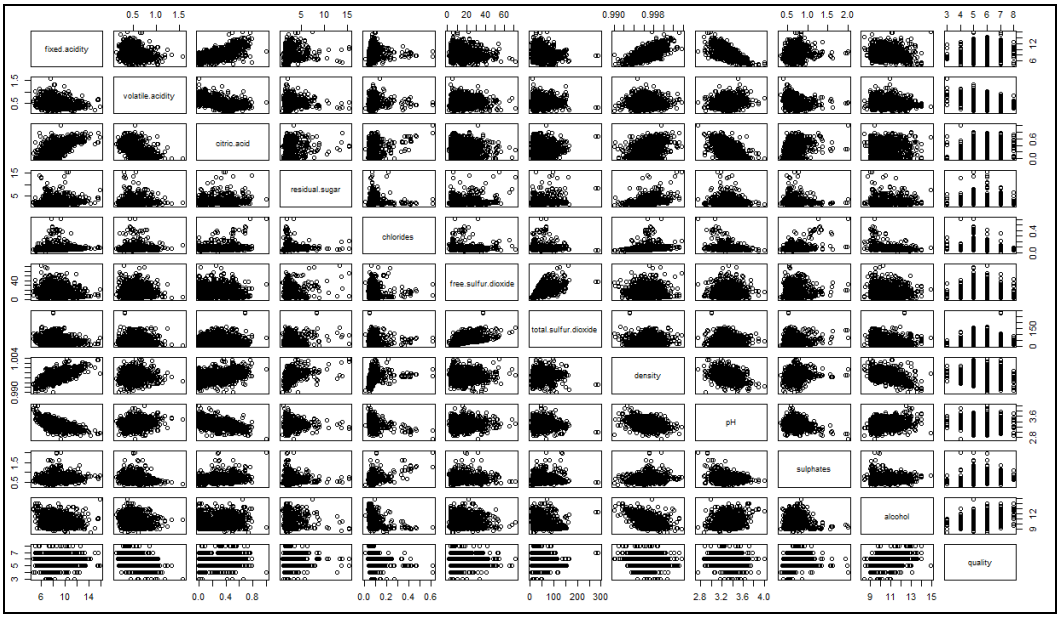


Figure 5: Plots of the Predictors for Correlation

Next, the data was modified so that each entry was classified into 0 or 1 based on the specific qualifications for the entry. If the quality was greater than 5.5 then it would have a quality of 1, which represents “Good”, and if the quality was less than 5.5 it would have a quality of 0, which represents “Bad”. This method of modification provided the boolean classification.

```
> data_main$quality
   [1] 5 5 5 6 5 5 5 7 7 5 5 5 5 5 5 5 7 5 4 6 6 5 5 5 6 5 5 5 5 6 5 6 5 6 5 6 6 7 4 5 5 4 6 5 5 4 5 5 5 5 5 6 6 5 6 5 5
  [58] 5 5 6 5 5 5 7 5 5 5 5 5 5 6 6 5 5 4 5 5 5 6 5 4 5 5 5 5 6 5 6 5 5 5 5 5 6 5 5 5 5 5 6 5 5 5 5 5 6 5 5 5 5 5 6
 [115] 5 6 6 6 6 6 6 5 6 5 5 5 5 5 5 7 5 5 5 5 6 6 5 5 5 5 5 5 5 5 6 5 6 5 5 5 5 6 6 6 4 5 5 5 5 5 5 5 6 5 4 6 5 5 5 5 4 6 5 4
 [172] 6 6 6 5 5 5 6 5 5 5 5 5 5 5 6 5 5 5 5 5 5 5 6 5 5 5 5 5 5 5 6 7 4 7 5 5 5 5 6 7 7 5 5 7 6 6 6 5 6 5 5 5 5 5 6 5 5 6 4 6 6 5
 [229] 6 5 7 6 6 5 6 6 6 6 6 6 5 6 6 7 7 6 5 5 6 6 6 6 5 5 6 5 5 5 5 5 7 5 4 5 5 5 5 7 4 8 6 6 6 6 6 5 5 5 6 6 6 8 7 6 7 5 7 5
 [286] 5 6 6 7 5 7 5 6 6 6 6 5 5 5 5 5 6 6 5 5 5 6 5 6 6 6 6 6 6 6 5 5 6 5 6 7 6 7 5 5 6 6 6 7 5 6 5 6 6 6 5 7 7 6 5 6 7 6 6
 [343] 6 6 6 5 7 6 6 6 6 6 5 5 6 6 5 7 7 6 5 6 5 5 5 7 6 7 5 5 7 5 6 6 5 6 7 6 7 6 6 6 6 6 6 5 6 6 6 6 7 8 6 5 5 5 7 5 6 6
 [400] 5 5 6 6 6 5 6 6 7 6 4 6 5 5 7 5 5 6 5 6 5 7 7 5 7 5 7 6 6 5 6 7 5 6 5 6 5 6 6 6 5 8 6 7 7 7 6 5 5 6 6 6 6 6 7 5 8
 [457] 5 5 7 3 6 5 5 5 6 5 6 6 6 6 5 5 6 6 5 6 5 5 6 5 6 5 8 5 5 6 5 5 5 6 7 6 6 7 7 6 6 8 6 5 8 6 6 7 7 7 7 7 7 6 6 7 5 6 6
 [514] 7 7 5 6 3 6 5 6 5 5 5 5 5 5 6 6 6 5 6 5 5 5 6 6 6 5 6 7 5 5 6 5 6 6 5 6 6 6 6 6 6 6 6 5 5 5 6 5 6 6 5 5 5 6 6 5 6 6 6 6
 [571] 6 6 5 4 6 6 4 5 5 6 5 5 5 5 7 7 6 7 5 8 7 5 6 5 5 5 5 5 6 6 6 6 4 6 5 6 6 6 7 6 6 6 5 5 6 5 6 5 5 6 5 5 5 5 5 5 6 5 5 5
 [628] 5 6 5 6 5 6 4 5 5 5 5 7 6 5 5 5 5 5 7 5 4 7 6 5 5 5 6 5 5 5 7 6 4 6 5 6 6 5 5 6 6 5 6 5 5 5 5 5 6 5 6 5 5 5 5 6 5 6 5 5
 [685] 5 5 5 5 5 5 3 5 5 5 5 6 6 6 5 6 6 6 6 4 4 5 5 5 6 6 5 5 5 5 5 6 5 5 5 5 5 5 5 5 4 5 6 5 5 6 6 5 5 5 5 5 5 5 6 6 5 5 6
 [742] 5 5 5 5 6 6 5 6 6 5 5 5 5 6 6 6 5 5 5 5 5 6 5 6 6 5 5 6 5 6 5 5 6 6 5 6 6 5 5 6 5 5 5 5 5 5 5 5 6 6 5 6 6 5 6 5 5 5 7
 [799] 6 6 5 5 7 6 6 7 7 7 5 6 5 6 5 4 6 5 6 6 5 5 5 7 5 5 5 5 5 7 5 8 6 4 6 3 4 5 5 7 7 7 5 7 5 6 5 6 5 5 6 5 5 5 5 5 6 6 6
 [856] 7 6 7 7 6 5 6 5 5 5 5 6 6 6 6 6 5 4 7 7 7 4 6 6 5 5 6 6 5 6 5 6 5 6 7 6 5 5 5 6 5 6 6 7 6 7 3 5 7 7 7 7 5 5 6 6 6 6 6
 [913] 6 7 6 6 5 6 6 6 6 5 6 6 6 5 7 6 4 5 7 5 5 6 5 5 6 6 4 7 5 7 7 7 7 7 7 7 7 7 7 7 7 7 6 5 6 6 7 5 6 5 5 6 6 6 7 5 6
 [970] 5 6 6 7 5 7 5 5 5 7 5 6 5 6 6 5 6 7 5 5 6 5 5 6 5 5 6 7 7 6 6
 [ reached getOption("max.print") -- omitted 599 entries ]
```

Figure 6: Quality Data before Modification

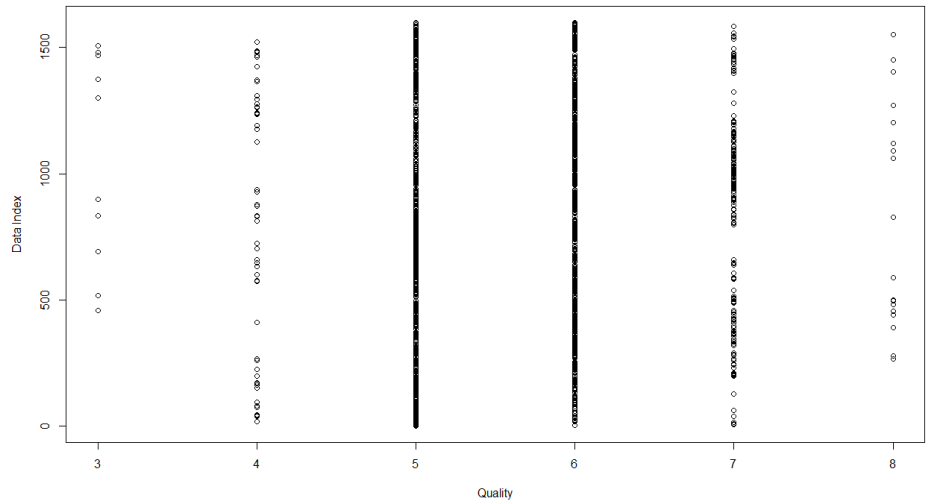


Figure 7: Scatter Plot of Quality vs Data

```
data_main$quality = ifelse(data_main$quality > 5.5,1,0)
data_main$quality <- as.factor(data_main$quality)
```

```
> data_main$quality
   [1] 0 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0
  [58] 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1
 [115] 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1 0 0
 [172] 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 0 0 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0 1 0 1 1 0
 [229] 1 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 1 0
 [286] 0 1 1 1 0 1 0 1 1 1 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 0 0 1 1 1 1 0 1 0 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1
 [343] 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 0 0 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1
 [400] 0 0 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 1 0 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1
 [457] 0 0 1 0 1 0 0 0 1 0 1 1 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 [514] 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 1 1 1 1
 [571] 1 1 0 0 1 1 0 0 0 1 0 0 0 1 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0
 [628] 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0
 [685] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1
 [742] 0 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 0 0 1 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 0 0 1
 [799] 1 1 0 0 1 1 1 1 1 1 1 0 1 0 1 0 0 1 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1
 [856] 1 1 1 1 1 0 1 0 0 0 0 1 1 1 1 1 0 0 1 1 1 0 1 1 0 0 1 1 0 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 1 1
 [913] 1 1 1 1 0 1 1 1 0 1 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 0 1
 [970] 0 1 1 1 0 1 0 0 0 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 1 1 1 1 1
 [ reached getOption("max.print") -- omitted 599 entries ]
Levels: 0 1
```

Figure 8: Quality Data after Modification

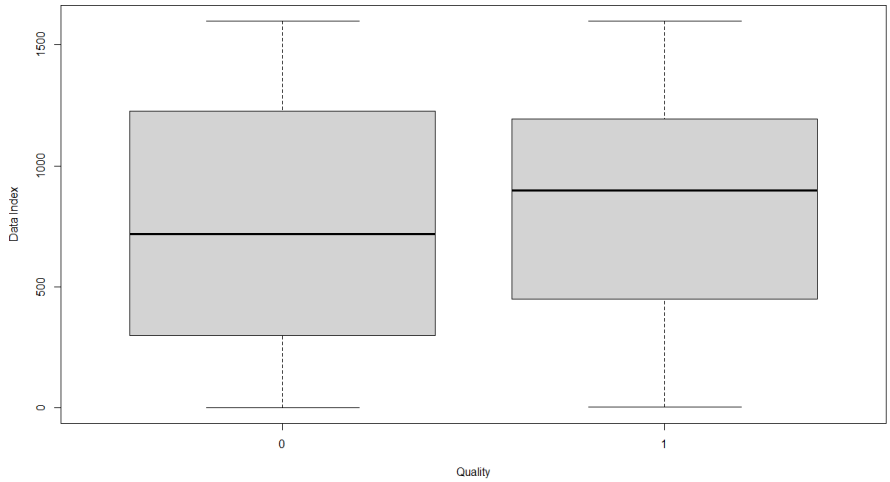


Figure 9: Box Plot of Quality vs Data

K-Nearest Neighbors (KNN) Analysis

The KNN section of the report was implemented using the R programming language and R studio. It makes use of the discoveries in the EDA section of the report.

The first step for KNN would be to scale/standardize the dataset. Judging off the information discovered in the EDA section of the report the data is not scaled properly. Each of the data values is on a different range of values, which can cause issues for prediction. Standardizing the inputs also creates a standard and numeric scale. To accomplish this the scale() function in R can be used to scale the data. This function is a generic function that centers and/or scales the columns of a numeric matrix. When scaling the data all but the last Quality column should be scaled since this is the classification column. The X predictors, all but the last column, should be on the same scale for prediction.

```
scaled_data = data.frame(scale(data_main[, -c(ncol(data_main))]))
scaled_data$quality = data_main$quality
```

The next major step of KNN would be to split the dataset into a Test and Training set of data for the model. There are a few ways of accomplishing this, but the one implemented below makes use of the createDataPartition() function in the R caret library. This function splits the specific indexes of the data in whatever percentage of distribution is entered. For this analysis, the data will be split into 70% Training and 30% Test. The new range of indexes can be used to get the Test and Training data from the data set provided.

```
trainIndex <- createDataPartition(scaled_data$quality, p = .7,
                                  list = FALSE,
                                  times = 1)
Train <- scaled_data[ trainIndex,]
Test <- scaled_data[-trainIndex,]
```

Now the Train dataset will contain just the scaled data which is from the scaled data set which accounts for 70% of the data. The Test dataset will contain the scaled test results which are 30% of the data provided. The 70/30 percent split is standard in practice. Then the last column of data, the quality column, must be dropped from the training and test dataset. The new dataset after the drop will be used in the KNN model to train and test.

```
X_Train = Train[-12]
X_Test = Test[-12]
```

Lastly, the correct classifications for the data in the Training and Test sets must be stored for later confirmation. The code below stores the correct values which are later accessed for confirmation of the results. This code drops all the columns but the last Quality column.

```
Train_Label = Train[,12, drop = TRUE]
Test_Label = Test[,12, drop = TRUE]
```

When conducting KNN it is also essential to calculate the best possible K value. The K value determines how many neighbors are considered when analyzing the data. To accomplish this Cross-Validation can be used for different models with different K values. Whichever model produces the least error, and in turn has the best accuracy, is the best K value. The code below tests all values from 1 to 100, each time storing the accuracy of the KNN confusion matrix. The confusion matrix is a table that describes the performance of a classification model on a set of test data for which the true values are known. The accuracy is determined by calculating the number of data values that are properly classified, and the number incorrectly classified.

```
k = 1:100
accuracy <- rep(NA, 100)
for(i in k){
  KNN_Fit = knn(X_Train, X_Test, Train_Label, k=i)
  conf = confusionMatrix(Test_Label, KNN_Fit)
  result <- conf$overall['Accuracy']
  accuracy[i] <- conf$overall['Accuracy']
}
```

Now the correct value for K can be found by finding the maximum accuracy in the accuracy stored data. The code below finds this optimal K value and the optimal accuracy computed by the model. It also prints the accuracy array for confirmation.

```
accuracy
maximum <- which.max(accuracy)
maximum
max(accuracy)
```

```
> accuracy
  [1] 0.7306889 0.6910230 0.7160752 0.6951983 0.7160752 0.7327766 0.7536534 0.7348643 0.7348643 0.7286013 0.7453027
 [12] 0.7557411 0.7515658 0.7369520 0.7494781 0.7327766 0.7348643 0.7369520 0.7265136 0.7244259 0.7202505 0.7202505
 [23] 0.7118998 0.7223382 0.7098121 0.7181628 0.7139875 0.7077244 0.7014614 0.7014614 0.7077244 0.7014614 0.6993737
 [34] 0.7035491 0.7118998 0.7077244 0.7160752 0.7118998 0.7098121 0.7118998 0.7139875 0.7181628 0.7118998 0.7118998
 [45] 0.7118998 0.7139875 0.7139875 0.7160752 0.7160752 0.7181628 0.7202505 0.7160752 0.7223382 0.7244259 0.7265136
 [56] 0.7202505 0.7306889 0.7244259 0.7202505 0.7181628 0.7223382 0.7244259 0.7286013 0.7160752 0.7160752 0.7181628
 [67] 0.7181628 0.7223382 0.7139875 0.7139875 0.7118998 0.7139875 0.7118998 0.7202505 0.7160752 0.7181628 0.7118998
 [78] 0.7160752 0.7098121 0.7160752 0.7139875 0.7098121 0.7139875 0.7181628 0.7098121 0.7118998 0.7098121 0.7098121
 [89] 0.7098121 0.7181628 0.7181628 0.7160752 0.7118998 0.7160752 0.7098121 0.7077244 0.7118998 0.7118998 0.7118998
[100] 0.7160752
```

Figure 10: Results from the Cross-Validation

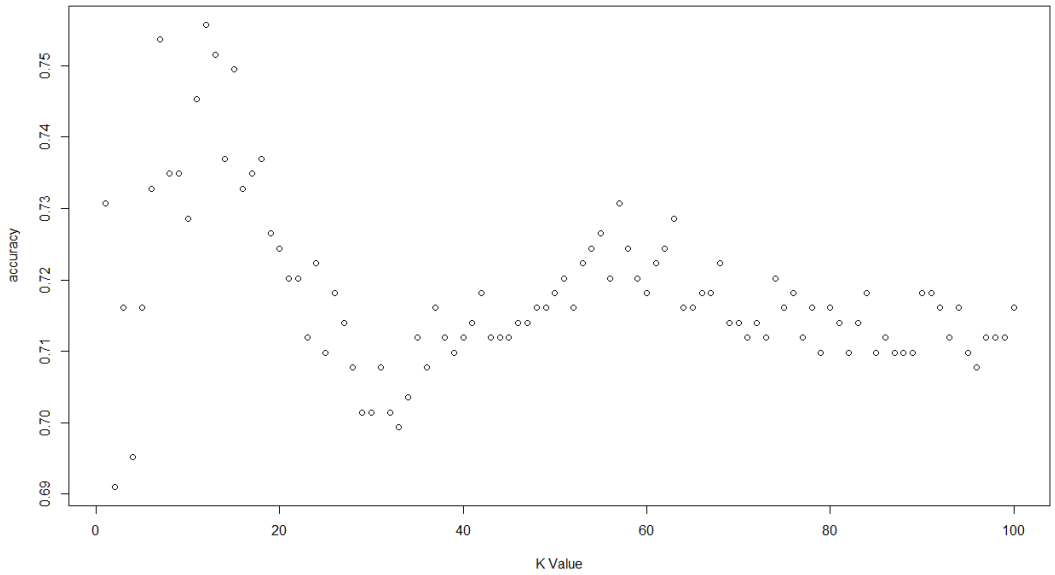


Figure 11: Accuracy of Model vs K-Value Cross-Validation

It would appear that the best accuracy was found with a K value of 12, and that accuracy was 0.7557411. This means that the model is accurate around 75% of the time it is given new data. Next, the conduction matrix for this specific model can be found once again and reviewed. This step is not required since it has already been calculated above. Below are the results.

```
KNN_Fit =knn(X_Train, X_Test, Train_Label, k=maximum)
confusionMatrix(Test_Label, KNN_Fit)
```

```
> confusionMatrix(Test_Label, KNN_Fit)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 153  70
         1  54 202

               Accuracy : 0.7411
                 95% CI : (0.6994, 0.7798)
    No Information Rate : 0.5678
    P-Value [Acc > NIR] : 2.621e-15

                  Kappa : 0.4774

 Mcnemar's Test P-Value : 0.178

            Sensitivity : 0.7391
            Specificity : 0.7426
         Pos Pred Value : 0.6861
         Neg Pred Value : 0.7891
             Prevalence : 0.4322
         Detection Rate : 0.3194
   Detection Prevalence : 0.4656
      Balanced Accuracy : 0.7409

       'Positive' Class : 0
```

Figure 12: Optimal K Confusion Matrix Classification

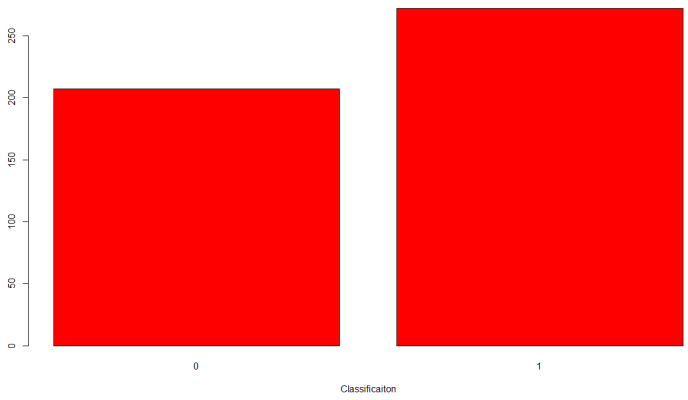


Figure 13: Plot Representing the KNN Fit

KNN Final Thoughts

The KNN method is effective at creating a classification model which accurately classifies new data. While the initial model produced only an accuracy of 74%, that is still a mostly accurate result. An important part of KNN is finding the best K value for the model. The implementation above does this via the cross-validation approach which finds the best K from a list of common possible K values. Further improvements can be made to KNN by scaling values in the data which is done above. In a model that does not scale the results are much worse. The results of a model without scaling can be seen below. Instead of scaling this model just uses the given data from the dataset, and only achieves an accuracy of 68%.

```
> confusionMatrix(Test_Label, KNN_Fit)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 148  75
         1  78 178

               Accuracy : 0.6806
                 95% CI : (0.6368, 0.7222)
    No Information Rate : 0.5282
    P-Value [Acc > NIR] : 8.678e-12

                  Kappa : 0.3587

 Mcnemar's Test P-Value : 0.8715

            Sensitivity : 0.6549
            Specificity : 0.7036
         Pos Pred Value : 0.6637
         Neg Pred Value : 0.6953
             Prevalence : 0.4718
         Detection Rate : 0.3090
   Detection Prevalence : 0.4656
      Balanced Accuracy : 0.6792

       'Positive' Class : 0
```

Figure 14: Model without Scaling

```
> confusionMatrix(Test_Label, KNN_Fit)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0   0  18
         1   0 460

               Accuracy : 0.9623
                 95% CI : (0.9411, 0.9775)
    No Information Rate : 1
    P-Value [Acc > NIR] : 1

                  Kappa : 0

 Mcnemar's Test P-Value : 6.151e-05

            Sensitivity :       NA
            Specificity : 0.96234
         Pos Pred Value :       NA
         Neg Pred Value :       NA
             Prevalence : 0.00000
         Detection Rate : 0.00000
   Detection Prevalence : 0.03766
      Balanced Accuracy :       NA

       'Positive' Class : 0
```

Figure 15: Split Value Equal to Four

Furthermore, changing the value at which the wine is considered “Good” or “Bad” can also result in a different KNN model accuracy. For example, if the new separating value is 4 instead of 5.5, this would result in all the quality 6 wines being considered as “Bad”. In this model the accuracy from the confusion matrix was higher, resulting in an accuracy of 96%. However, this model has an optimal K of 7 and is much simpler since the majority of the data entries are going to be considered “Good”, based on the table from EDA. Ultimately the specific quality of the wine is different, which results in the different accuracy between the two different models. Review the confusion matrix below for accurate results from R.

Logistic Regression

The logistic regression for this project was also implemented using the R programming language on R Studio. It also references the initial code done in the EDA section.

From the EDA section, we stated that a wine is of good quality if the quality was above a 5.5 (Figure 6). From here, we needed to split our data into a training and a test set. The training set would be used to build our model and the test set would be used to check the strength of our model. For this analysis, the training data was set to contain 1278 observations (0.8 or 80%) of the entire data set of 1599. The test data contained 321 observations (0.2 or 20%) of the data set. This can be seen in the code snippet below

```
split <- sample.split(wine$quality, SplitRatio = .8)
train <- subset(wine, split == TRUE)
test <- subset(wine, split == FALSE)
```

After splitting the set, we created a model using the R command glm(). This can be seen in the snippet below. For our fit, we used the variables alcohol, pH, sulfur dioxide, sulphates, chlorides, and volatile acidity. We chose these values based on the correlations that were produced from the initial correlation plot (Figure 3) in the EDA section.

```
fit <- glm(isGood ~ alcohol+ pH + total.sulfur.dioxide
           + sulphates + chlorides + volatile.acidity,
           data = train, family = binomial(link = 'logit'))
summary(fit)
```

After creating the fit, we ran the summary() command to get our coefficients for our model as well as some other useful information on our variables. This can be seen below.

```
Call:
glm(formula = isGood ~ alcohol + pH + total.sulfur.dioxide +
    sulphates + chlorides + volatile.acidity, family = binomial(link = "logit"),
    data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.3067  -0.8643   0.3030   0.8573   2.2272

Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)         -5.622355   1.668945  -3.369 0.000755 ***
alcohol              0.900550   0.080098  11.243  < 2e-16 ***
pH                  -0.818587   0.483841  -1.692 0.090674 .
total.sulfur.dioxide -0.012053   0.002095  -5.753 8.77e-09 ***
sulphates            2.461877   0.482231   5.105 3.30e-07 ***
chlorides           -4.560597   1.649211  -2.765 0.005687 **
volatile.acidity    -2.769228   0.428495  -6.463 1.03e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1765.6  on 1277  degrees of freedom
Residual deviance: 1341.5  on 1271  degrees of freedom
AIC: 1355.5

Number of Fisher Scoring iterations: 4
```

Figure 16: Summary output of the model

From that output summary, we see the coefficients that we previously mentioned and their estimated values. We use these values to make the following formula:

$\log\left(\frac{p(Y=1)}{1-p(Y=1)}\right) = -5.6223 + 0.9X_1 - 0.8186X_2 - 0.0121X_3 + 2.4619X_4 - 4.5606X_5 - 2.7692X_6$

Where X_1 is alcohol, X_2 is pH, X_3 is total sulfur dioxide, X_4 is sulphates, X_5 is chlorides, and X_6 is volatile acidity.

Furthermore, from the summary output, we are given the p-values of each of the variables which indicate how significant they are to the model. From it, we can see that the values alcohol, total sulfur dioxide, sulphates, and volatile acidity are significant in predicting whether the wine will be good or bad because they have p-values that are less than 0.005. Chlorides have a p-value of 0.005687. For all intents and purposes, we can also say this is significant as there may be some rounding error to account for when making these calculations. pH was the variable that was deemed as insignificant as it had a p-value of 0.09.

Lastly, we created a confusion matrix to determine the accuracy of our model. This was done with the confusionMatrix() command in r.

```
pred <- predict.glm(fit, newdata = test, type = 'response')
vals <- ifelse(pred > 0.5 , 1, 0)

confusionMatrix(as.factor(vals), as.factor(test$isGood))
```

The results of our confusion matrix can be seen in the image below. From this image, we see that our model had a 0.7632 or 76.32% accuracy when predicting the quality of the wine.

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 112  39
         1  37 133

               Accuracy : 0.7632
                 95% CI : (0.7129, 0.8087)
    No Information Rate : 0.5358
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.5245

 Mcnemar's Test P-Value : 0.9087

            Sensitivity : 0.7517
            Specificity : 0.7733
         Pos Pred Value : 0.7417
         Neg Pred Value : 0.7824
             Prevalence : 0.4642
         Detection Rate : 0.3489
   Detection Prevalence : 0.4704
      Balanced Accuracy : 0.7625

       'Positive' Class : 0
```

Figure 17: Output of the confusion matrix and accuracy percentage

Logistic Regression Final Thoughts

The logistic regression model was a fairly effective model to use for our data set in classifying new data. Our model of:

$\log\left(\frac{p(Y=1)}{1-p(Y=1)}\right) = -5.6223 + 0.9X_1 - 0.8186X_2 - 0.0121X_3 + 2.4619X_4 - 4.5606X_5 - 2.7692X_6$

predicted the test data set with an accuracy of 76%. This is similar to what we found with the KNN analysis which had an accuracy of 74%. Overall, 76% succession rate is a good success rate when it comes to predicting the quality of the wine based on the observations we had from the data set. We can further improve on this model by potentially trying out different variables and seeing how it affects the accuracy result. We can also try removing different variables from the model to see its effects. For instance, removing the pH and the chlorides variables gives us a slightly less accurate model with only 75% success. This can be seen in the screenshot below.

```
               Accuracy : 0.7539
                 95% CI : (0.703, 0.8)
    No Information Rate : 0.5358
    P-Value [Acc > NIR] : 6.288e-16

                  Kappa : 0.5046

 Mcnemar's Test P-Value : 0.822

            Sensitivity : 0.7248
            Specificity : 0.7791
         Pos Pred Value : 0.7397
         Neg Pred Value : 0.7657
             Prevalence : 0.4642
         Detection Rate : 0.3364
   Detection Prevalence : 0.4548
      Balanced Accuracy : 0.7520

       'Positive' Class : 0
```

Figure 18: Output of the confusion matrix and accuracy percentage when removing pH and chloride variables.

Figure 19: Confusion matrix and accuracy percentage when removing all variables except volatile acidity.

This is interesting to see how small of a decrease in accuracy pH and chlorides resulted in but it is to be expected as from the previous summary, we saw that these variables were insignificant based on p-values.

Furthermore, we can experiment to see what removing all but one variable does to the model. For instance, by removing all variables besides the volatile acidity, we get a model that is only 62.3% accurate. This can be seen by the figure on the right. By testing these changes in our model, that is the removal of a few or almost all variables, we seem to get lower and lower accuracy reports. Based on this, we can conclude that the model that we came up with that involved the 6 previously mentioned variables, is the best-suited logistic regression model for prediction.

```
               Accuracy : 0.6231
                 95% CI : (0.5675, 0.6763)
    No Information Rate : 0.5358
    P-Value [Acc > NIR] : 0.0009801

                  Kappa : 0.2371

 Mcnemar's Test P-Value : 0.2031148

            Sensitivity : 0.5436
            Specificity : 0.6919
         Pos Pred Value : 0.6045
         Neg Pred Value : 0.6364
             Prevalence : 0.4642
         Detection Rate : 0.2523
   Detection Prevalence : 0.4174
      Balanced Accuracy : 0.6177

       'Positive' Class : 0
```

Quadratic Discriminant Analysis

First, we set up a test and training dataset. The training dataset contains a randomly selected 70% of the data, while the remaining 30% of the data is assigned to the test dataset. Here, we define quality.high as TRUE when it is higher than the mean, and FALSE otherwise. The original quality column is removed.

```
library(caTools)
wine = read.csv("winequality-red.csv", sep=";", header = TRUE)
wine$quality.high = wine$quality > 5.5
wine = subset(wine, select=-c(quality))
split = sample.split(wine, SplitRatio=0.7)
wine.train = subset(wine, split==TRUE)
wine.test = subset(wine, split==FALSE)
print(summary(wine))
```

Based on the correlation plot in Exploratory Data Analysis, we can see that the quality correlates most strongly with alcohol, volatile.acidity, sulphates, and citric.acid. Thus, we train QDA on these features. Below are the results from the QDA fit which shows the true and false rating of the model fit.

```
library("MASS")
wine.qda =
  qda(quality.high ~ alcohol + volatile.acidity + sulphates + citric.acid,
                wine.train)
print(wine.qda)
```

```
qda(quality.high ~ alcohol + volatile.acidity + sulphates + citric.acid,
    data = wine.train)

Prior probabilities of groups:
    FALSE      TRUE
0.4595865 0.5404135

Group means:
        alcohol volatile.acidity sulphates citric.acid
FALSE  9.924642        0.5825358 0.6187117   0.2419427
TRUE  10.822870        0.4731391 0.6978087   0.3017913
```

Figure 20: QDA Fit Summary

The next step is to assess the accuracy of the model. To accomplish this a prediction must be made using the test dataset. The predict() function can be used to get an accurate estimate. Then via a confusion matrix, the accuracy of the model can be ascertained. To get a confusion matrix in R the confusionMatrix() function can be used. For the confusion matrix, it is important to pass in the test results, as well as the predicted values. The code below implemented the above methods to calculate the accuracy of the model via a confusion matrix.

```
library(caret)
wine.qda.preds = predict(wine.qda, wine.test)
confusionMatrix(wine.qda.preds$class, as.factor(wine.test$quality.high))
```

```
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE   202   87
     TRUE     53  193

               Accuracy : 0.7383
                 95% CI : (0.6989, 0.7751)
    No Information Rate : 0.5234
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4786

 Mcnemar's Test P-Value : 0.005287

            Sensitivity : 0.7922
            Specificity : 0.6893
         Pos Pred Value : 0.6990
         Neg Pred Value : 0.7846
             Prevalence : 0.4766
         Detection Rate : 0.3776
   Detection Prevalence : 0.5402
      Balanced Accuracy : 0.7407

       'Positive' Class : FALSE
```

Figure 21: Confusion Matrix from QDA Model

We obtain an overall accuracy of ~74%, which is similar to other models trained on the same data. To validate the results the confusion matrix was printed. Based on the True Negative and True Positive results it is clear that most testing values are classified correctly. The results of this model can also be validated by hand by calculating the accuracy using the formula $\frac{TP + TN}{TP + TN + FP + FN}$ for accuracy.

Quadratic Discriminant Analysis Final Thoughts

Lastly, other models of the data can be used to cross-validate the performance of the fit above. To accomplish this it is possible to select predictors that did not correlate well with the quality and test how they impact the overall accuracy of a model. For example, if the model was instead fitted using all the predictors instead of those that correlate well with the quality feature. The below results show the model which implemented all the possible predictors.

```
wine.qda2 =qda(quality.high ~ .,wine.train)
wine.qda.preds = predict(wine.qda2, wine.test)
```

```
confusionMatrix(wine.qda.preds$class,
as.factor(wine.test$quality.high))
```

Based on the results it is clear that this model is much less accurate since it only calculated an accuracy of ~68%. This result is likely due to the model having outliers and non-critical data, which skewed the results of the model. For this model, there was much more misclassified data for the False Positive category. This meant that the model evaluated that the quality of the wine was "Good" when in reality the quality was "Bad" according to the scale.

```
> confusionMatrix(wine.qda.preds$class, as.factor(wine.test$quality.high))
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE   140   70
     TRUE     99  225

               Accuracy : 0.6835
                 95% CI : (0.6422, 0.7228)
    No Information Rate : 0.5524
    P-Value [Acc > NIR] : 4.113e-10

                  Kappa : 0.3525

 Mcnemar's Test P-Value : 0.03125

            Sensitivity : 0.5858
            Specificity : 0.7627
         Pos Pred Value : 0.6667
         Neg Pred Value : 0.6944
             Prevalence : 0.4476
         Detection Rate : 0.2622
   Detection Prevalence : 0.3933
      Balanced Accuracy : 0.6742

       'Positive' Class : FALSE
```

Figure 22: Confusion Matrix for QDA with All Predictors

The QDA fitting model for the data was about as accurate as the models that were implemented prior. KNN and Logistic fit produced a similar accuracy rating for the data. The only difference is that QDA seems to be more susceptible to changing the predictors of the model. Furthermore, this difference is possibly due to how the data was chosen to be split. A model which was more distinctly splitting the original quality would likely result in more accurate results. For example, if the quality was originally classified by a value of > 6 instead of 5.5 then more data would be easily classified. This can be concluded by the results below for a QDA model. The model splits the data for quality at a value of 6 instead of 5.5.

```
wine$quality.high = wine$quality > 6
wine = subset(wine, select=-c(quality))
split = sample.split(wine, SplitRatio=0.7)
```

```
> confusionMatrix(wine.qda.preds$class, as.factor(wine.test$quality.high))
Confusion Matrix and Statistics

          Reference
Prediction FALSE TRUE
     FALSE   140   70
     TRUE     99  225

               Accuracy : 0.6835
                 95% CI : (0.6422, 0.7228)
    No Information Rate : 0.5524
    P-Value [Acc > NIR] : 4.113e-10

                  Kappa : 0.3525

 Mcnemar's Test P-Value : 0.03125

            Sensitivity : 0.5858
            Specificity : 0.7627
         Pos Pred Value : 0.6667
         Neg Pred Value : 0.6944
             Prevalence : 0.4476
         Detection Rate : 0.2622
   Detection Prevalence : 0.3933
      Balanced Accuracy : 0.6742

       'Positive' Class : FALSE
```

Figure 23: Confusion Matrix for QDA with a Split of 6

Artificial Neural Network

See NeuralNetwork.ipynb for full code

For the fourth and final method of explaining our data, we used an artificial neural network. To construct this model, we used the python libraries Keras, Tensorflow, SciKit-learn, and Pandas. Using Keras' to_categorical utility, the wine quality was converted into a one-hot representation, with the first column indicating bad quality wine, and the second indicating good quality wine.

```
predictors = wine.loc[:, wine.columns != 'quality']
labels = wine['quality'] > wine['quality'].mean()
catlabels = tf.keras.utils.to_categorical(labels)
# order: bad wine, good wine
```

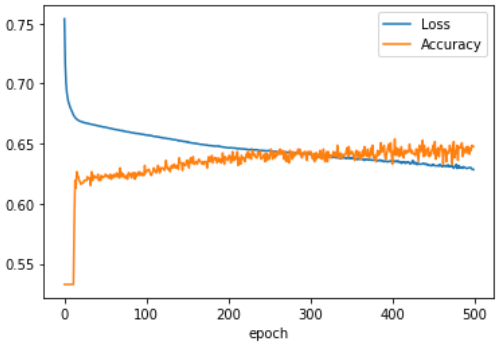
The network layout chosen was a sequential network composed of 4 layers: the input layer with 12 relu perceptrons, an internal layer with 12 sigmoid perceptrons, another internal layer with 6 sigmoid perceptrons, and finally a softmax layer with 2 perceptrons to show confidence for each of the one-hot columns. The model was built using categorical cross-entropy for the loss function, to work with the one-hot representation, and the metric tracked was categorical accuracy.

```
# build the model
model = Sequential([
    Dense(12, activation='relu'),
    Dense(12, activation='sigmoid'),
    Dense(6, activation='sigmoid'),
    Dense(2, activation='softmax')
])

model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['categorical_accuracy'])
```

The dataset was split into training and test data, and the model was fit to this data in batches of 64 for 500 epochs.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(predictors, catlabels)
history = model.fit(
    X_train,
    y_train,
    batch_size=64,
    epochs=500,
    validation_split=0.2,
)
```



Over the course of training, the model reached a final categorical accuracy score of 64.75% on the training data. When evaluated on the test data, this model scored 64.25% of the wines correctly, making it the least performant of our selected models.

Figure 24: Neural Network Training accuracy/loss

```
test_loss, test_acc = model.evaluate(X_test, y_test, batch_size=128)

print("Loss on test set:", test_loss)
print("Accuracy on test set:", test_acc)
```

✓ 0.1s Python

```
4/4 [==============================] - 0s 2ms/step - loss: 0.6292 - categorical_accuracy:
0.6425
Loss on test set: 0.6291632652282715
Accuracy on test set: 0.6424999833106995
```

The flexibility of the Artificial Neural Network is great strength in creating models, but in this case, as in many, domain knowledge and human common sense allow a human with some data exploration tools to find key factors in a model. In a classification problem with only a few predictors and a well-described dataset, methods like logistic regression or QDA can better allow data scientists to embed their domain knowledge in the construction of the model.