UNIVERSITY OF CATANIA

DEPARTMENT OF ECONOMY

MASTER OF DATA SCIENCE FOR MANAGEMENT

Analysis and Prediction Report on

White Wine Quality Dataset

ACADEMIC YEAR 2020- 2021

PROFESSOR S. INGRASSIA

Manuel Scionti

David Anthony Pagano

Francesco Maria Speciale

SOMMARIO

DATA DESCRIPTION

The dataset that we will use is “White wine quality” dataset, which exists in the project folder. The data set is divided into three data sets:

+ Train data: about 60% of the units of the original dataset

+ Validation data: about 20% of the units of the original dataset

+ Test data: about 20% of the units of the original dataset.

This data will allow us to create different machine learning models to determine how different independent variables help predict our determined dependent variable, which is **quality**.

EXPLORATORY DATA ANALYSIS

The sets contain physicochemical properties of red and white Vinho Verdes wines and their respective sensory qualities as assessed by wine experts. For easier handling, the red wine dataset was excluded from the analysis. The final dataframe comprises 2937 observations, 12 columns and no missing values were found.

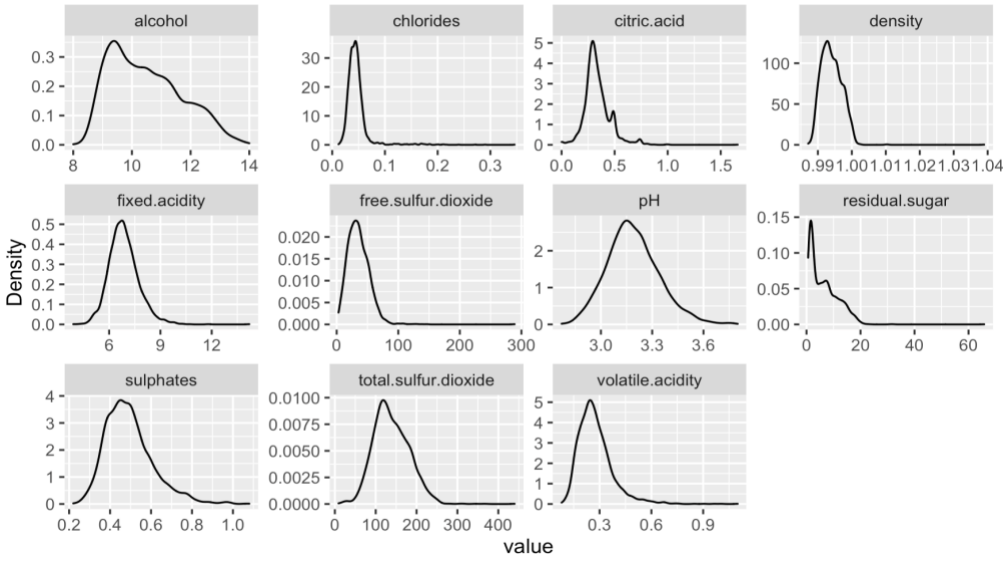
Attribute	Unit	Description	Data Type
Fixed Acidity	g/L	Concentration of non-volatile tartaric acid in the wine.	Float
Volatile Acidity	g/L	Concentration of volatile acetic acid in the wine.	Float
Citric Acid	g/L	Concentration of citric acid in the wine.	Float
Residual Sugar	g/L	Concentration of sugar remaining after the fermentation in the wine.	Float
Chlorides	g/L	Concentration of sodium chloride in the wine.	Float
Free Sulfur Dioxide	mg/L	Concentration of free, gaseous sulfur dioxide in the wine.	Float
Total Sulfur Dioxide	mg/L	Total concentration of sulfur dioxide in the wine.	Float

Density	g/cm3	Density of the wine.	Float
pH	1 - 14	Acidity of the wine.	Float
Sulphates	g/L	Concentration of potassium sulfate in the wine.	Float
Alcohol	vol%	Alcohol content of the wine.	Float
Quality	1 - 10	Wine quality score as assessed by experts.	Categorical

NUMERICAL VARIABLES

Here we proceed with some more in-depth analysis: mean: standard deviation and quartiles were computed, after that, we plotted the variables' distributions.

```
── Variable type: numeric ──────────────────────────────────────────────────────────────────────────
   skim_variable        n_missing complete_rate     mean       sd    p0     p25     p50     p75    p100
 1 fixed.acidity                0             1   6.85    0.865   3.9     6.3     6.8     7.3    14.2
 2 volatile.acidity             0             1   0.275   0.0994  0.08    0.21    0.26    0.32    1.1
 3 citric.acid                  0             1   0.337   0.122   0       0.27    0.32    0.39    1.66
 4 residual.sugar               0             1   6.41    5.13    0.6     1.7     5.2     9.9    65.8
 5 chlorides                    0             1   0.0457  0.0216  0.012   0.036   0.043   0.05    0.346
 6 free.sulfur.dioxide          0             1  35.2    17.6     3      23      34      46     289
 7 total.sulfur.dioxide         0             1 138.     42.9     9     109     134     168     440
 8 density                      0             1   0.994   0.00304 0.987   0.992   0.994   0.996   1.04
 9 pH                           0             1   3.19    0.152   2.77    3.08    3.18    3.28    3.8
10 sulphates                    0             1   0.493   0.116   0.22    0.41    0.48    0.55    1.08
11 alcohol                      0             1  10.5     1.22    8       9.5    10.3    11.3    14
```



Most distributions encountered during the exploration of the parameters looked rather usual. In general, they were positively skewed with a narrow main peak.

- *Acidity is measured in terms of fixed and volatile components. Most wines have a pH of around 3.2.*

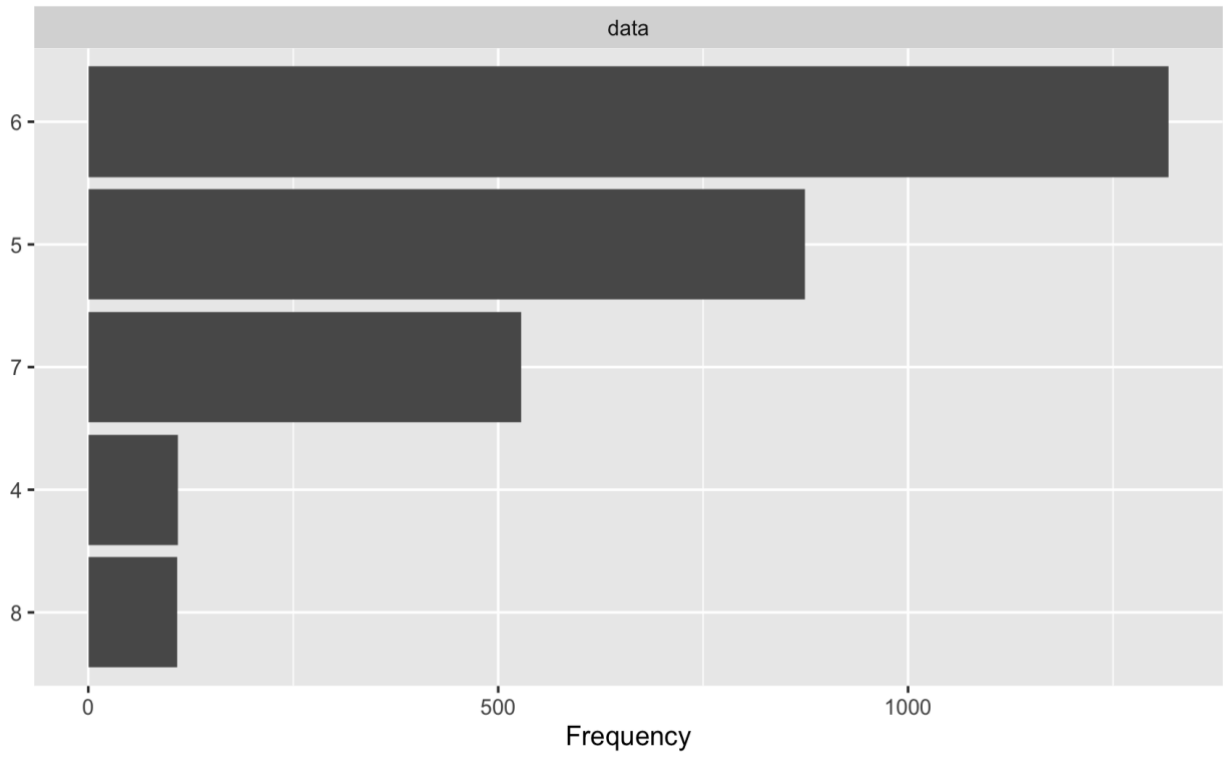
- *Sulfur dioxide concentration varies widely in the investigated wines.*

- *The wines have an alcohol content ranging between 8 and 15 vol%.*

- *Density appears to most closely resemble a normal distribution.*

ORDINAL VARIABLE

Wine quality shows a rather symmetrical distribution. Most wines have a quality score of 6. The highest score is 8 and the worst wines got a rating of 4.



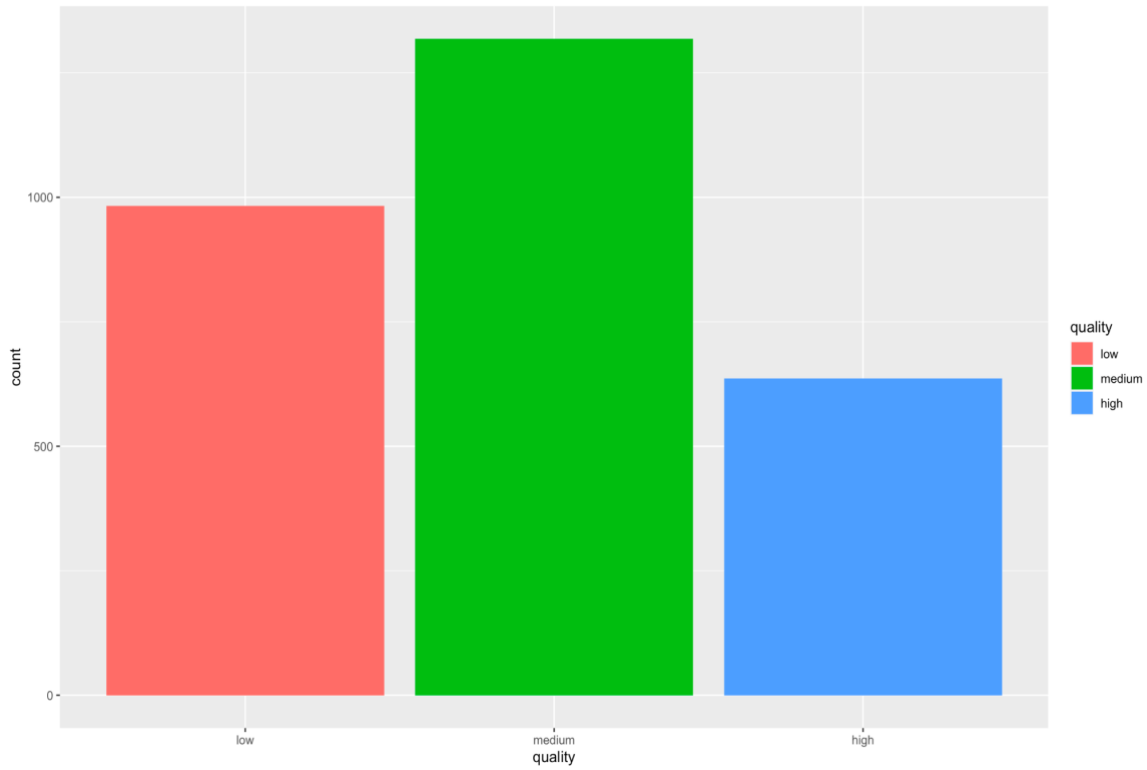
In order to facilitate our investigation, we aggregate *quality* in three different levels: **low, medium, high.**

Low → quality levels < 6

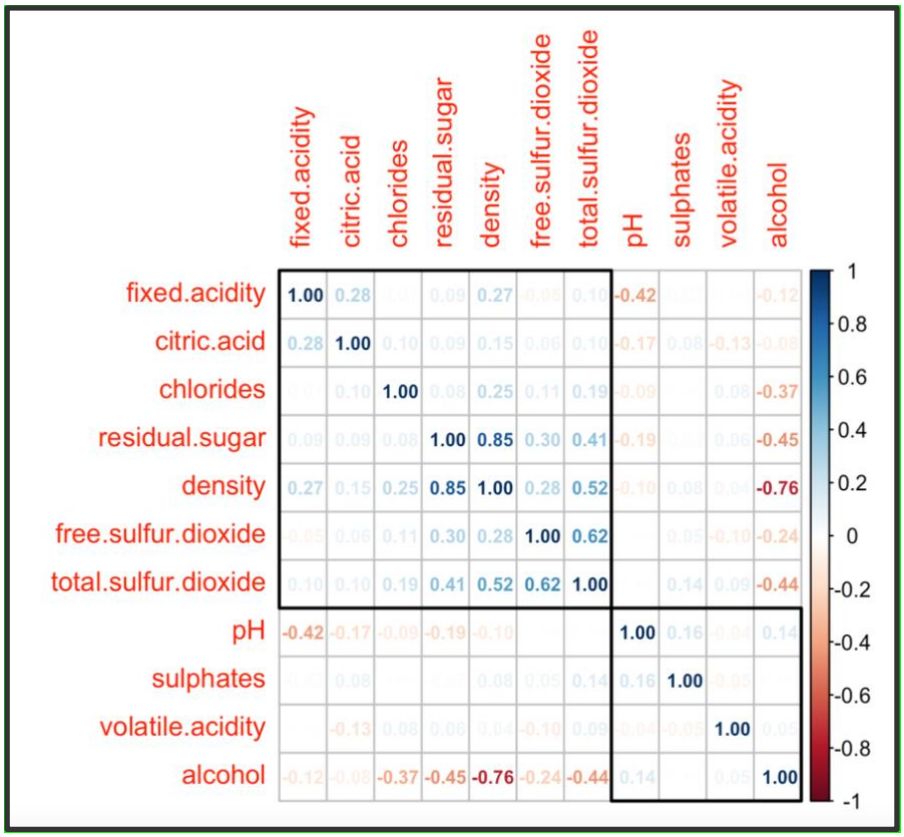
Medium → quality levels = 6

High → quality levels > 6

Done so, we computed quality levels' frequencies:



In order to get a deeper understanding of the predictors' interactions, corrplot() functions perform hierarchical clustering in two groups, based on the correlation values. The groups are reported in the graph below and have a squared border. We see that the predictors of the two clusters are generally positively correlated or uncorrelated between them and negatively correlated or uncorrelated to the members of the other cluster. This can help us in finding similar trends for variables belonging to the same clusters. But first, it's worth making some observations on the values found.



- *Since alcohol weighs less than water there is no surprise that the alcohol variable and density are very strongly negatively correlated with a -0.76 negative correlation.*

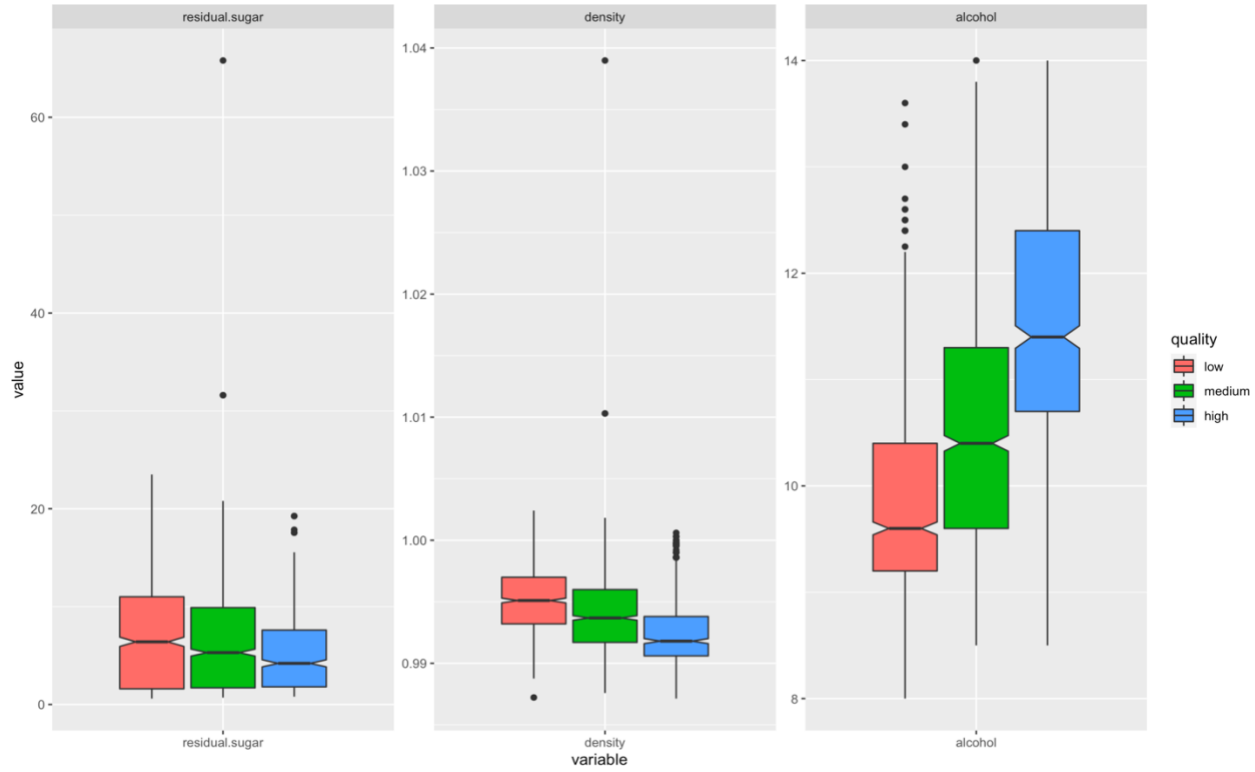
- *Since the fermentation of sugar is what produces the alcohol it is also not surprising that residual.sugar and alcohol have a negative correlation of -0.45.*

- *Residual.sugar and density have the strongest correlation of 0.85 which is due to the fact that sugar is heavier than water so where there is more residual.sugar in the wine the wine is more dense or heavy.*

From the boxplots here reported it comes clear how differently some predictors are distributed dependently on the quality:

Alcohol rate usually increases with quality. The boxes for low and high quality wines are practically separated. In particular, the range for high quality wines is pretty well defined, whereas for low-quality wines some outliers are present.

residual sugar range of values becomes narrower increasing in quality, density range of values becomes narrower and shifts down. No wonder density and residual sugar belong to the same cluster.



To watch event better the differences between high and low wines, a scatter plot of all but the medium (6-rated) wines predictors, for which is rather difficult to detect characteristic features, is reported below. The graph shows that the points of the two classes are not well separated in most of the cases and the density plots are overlapped. Only alcohol row shows a rather good separation for the two categories, as anticipated by the boxplot. For the rest of the predictors, in general it appears that the high-quality and low-quality points are distributed similarly, even though usually the most exquisite wines occupy a smaller area, largely a subset of the domain of the poorest wines. This suggests that understanding what makes a bad wine may be easier than understanding what makes a good one. All points that would fall outside the blue area would

probably be rightly classified as poor, but what about all predictor pairs within the blue area, i.e. also within the red area?

Considering that scatter plot lacks 6-rated wines, which are considered high-rated according to the task assigned and so make even more difficult detecting patterns, predicting wine's quality doesn't appear to be an easy task and the role of most of the predictors is ambiguous.

LOGISTIC REGRESSION

We tried to find the best logistic regressor by using the library _*bestglm*_ according to different criteria.

Preliminarily to the analysis, variance inflation factors were calculated. Most of the predictors present very low collinearity. But density especially, and also alcohol and residual sugar, have an important level of collinearity. This addresses to not include all of them in the selected regressor.

```
       fixed.acidity     volatile.acidity          citric.acid       residual.sugar            chlorides
            2.547139            1.133482             1.152298            11.902113             1.252629
free.sulfur.dioxide total.sulfur.dioxide              density                    pH            sulphates
            1.797904            2.218609            24.948221             2.100415             1.138128
             alcohol
            6.544907
```

Executing the command *bestglm* from the homonymous library the best logistic regressor according to a specified information criterion is returned. For this analysis, we were limited to searching linear relationships between the predictors and the categorical output, excluding non-linear dependencies and interaction effects.

BEST REGRESSOR ACCORDING TO AIC

```
AIC
BICq equivalent for q in (0.731786613723476, 0.970004141860054)
Best Model:
                       Estimate   Std. Error    z value     Pr(>|z|)
(Intercept)         2.450163e+02 55.730938811   4.396414 1.100538e-05
volatile.acidity   -6.228823e+00  0.510974760 -12.190080 3.510748e-34
residual.sugar      1.673917e-01  0.022836417   7.330033 2.300954e-13
free.sulfur.dioxide 6.655438e-03  0.002759572   2.411765 1.587553e-02
density            -2.571109e+02 55.819657576  -4.606100 4.102917e-06
pH                  1.007396e+00  0.323998049   3.109267 1.875522e-03
sulphates           1.691834e+00  0.443313139   3.816340 1.354457e-04
alcohol             7.478188e-01  0.084997168   8.798161 1.390760e-18
```

Even though this regressor minimizes AIC, there's more than a suspicion that it is unacceptable. Indeed, it contains all the three highly-collinear predictors and it should suggest that the real relationships between input and output are actually hidden by this regressor. But that's not all.

Std.Error for density, which is, moreover, the most influential predictor for this regressor, is really too big compared to its estimate. There's a lot of uncertainty on the intercept too and also other predictors show consistent std.errors in comparison to their estimates. Furthermore, most of the predictors have estimates very close to 0. This means that their relationship with the quality is not so distinguishable: except for density's dominating influence on this regressor, only the contribution of volatile acidity is somehow significant.

BEST REGRESSOR ACCORDING TO BIC

```
BIC
BICq equivalent for q in (0.258106042177806, 0.731786613723476)
Best Model:
                     Estimate  Std. Error    z value     Pr(>|z|)
(Intercept)       254.9498169 55.17048064   4.621127 3.816604e-06
volatile.acidity   -6.2992654  0.50899764 -12.375824 3.532489e-35
residual.sugar      0.1773963  0.02241944   7.912610 2.520482e-15
density          -266.8931508 55.26845391  -4.829032 1.371981e-06
pH                  1.0816777  0.32272653   3.351685 8.032134e-04
sulphates           1.7417166  0.44111438   3.948447 7.865996e-05
alcohol             0.7207588  0.08363170   8.618248 6.798608e-18
```

Considerations for this regressor are analogue to those made for the previous one.

BEST REGRESSOR ACCORDING TO CROSS-VALIDATION

```
CV(K = 11, REP = 5)
BICq equivalent for q in (1.95110594347625e-12, 0.00108331696606778)
Best Model:
                    Estimate  Std. Error    z value     Pr(>|z|)
(Intercept)      -8.63624767 0.552099245 -15.642564 3.733873e-55
volatile.acidity -6.56624784 0.504861487 -13.006038 1.130529e-38
residual.sugar    0.07371911 0.009673119   7.621028 2.516637e-14
alcohol           1.03592204 0.051949618  19.940898 1.798079e-88
```

Cross validation returns the most trustable results. Data were randomly split in 11 non-overlapping folds of 45 units five times, to reduce variance. The output model shows better p-values and above all std. errors more proportioned to their estimates. VIF values are low for the three predictors and prediction accuracy on the validation set (over 75%) is slightly better than that obtained but the previous two regressors (around 73%). Despite being much simpler than

the competitors, the percentage loss in terms of AIC and BIC from the best methods is really small (-1.34% for AIC and - 0.60% for BIC). For all these reasons this must be considered the best logistic regressor. However, the first impression was confirmed: accuracy is still quite low and far from being satisfying.

COLLINEARITY

The very low VIF values for the three predictors of the last regressor confirms its goodness of fit.

```{r}
vifx(wwtrd.class.glm[c('alcohol', 'residual.sugar', 'volatile.acidity')])
```

```
          alcohol    residual.sugar volatile.acidity
         1.266401         1.268410         1.012343
```

PERCENTUAL LOSS

As you can see from the following chunk, the percentual loss in terms of AIC and BIC from the best methods is really small.

```{r}
AIC(best.reg.cv$BestModel)
BIC(best.reg.cv$BestModel)

((AIC(best.reg.aic$BestModel)-AIC(best.reg.cv$BestModel))/AIC(best.reg.aic$BestModel))*100
((BIC(best.reg.bic$BestModel)-BIC(best.reg.cv$BestModel))/BIC(best.reg.aic$BestModel))*100
```

```
[1] 3043.598
[1] 3067.538
[1] -1.345891
[1] -0.6058989
```

PROBABILITY OF BEING HIGH-QUALITY WINE:

```
         1          2          3          4          5          6
0.56768908 0.39435286 0.67863257 0.05986188 0.06051194 0.78587893
```

PREDICTED CLASS OF THE FIRST SIX ROWS:

```
     1      2      3      4      5      6
"high"  "low" "high"  "low"  "low" "high"
```

ACCURACY AND ERROR RATE:

```
[1] 75.2809
[1] 24.7191
```

CONFUSION MATRIX

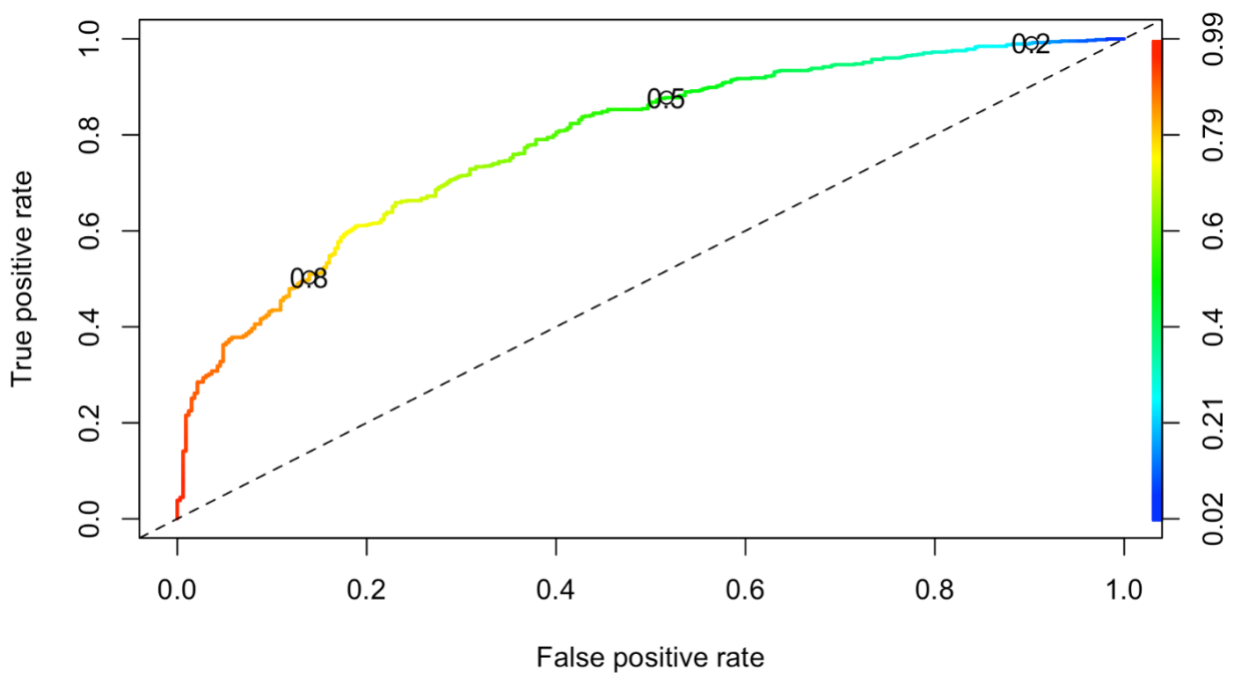
```
          real
predicted high low Sum
     high  573 170 743
     low    80 160 240
     Sum   653 330 983
```

TRUE POSITIVE RATE, FALSE POSITIVE RATE AND ROC CURVE

True Positive Rate: `[1] 87.74885`

False Positve Rate: `[1] 51.51515`

Area Under Curve: `[1] 0.7866908`



Confusion matrix and ROC curves help us understanding the reasons of the poor results we got. **While true positive rate is quite good 87.75%, false positive rate 51.16% is dramatically high and affects the whole accuracy**. This means that a lot of low-quality wines are classified as precious. This confirms what we had already observed from the scatter plot in EDA. Since it looks like that the domain of high-quality wines is a subset of the domain of low-quality wines, lots of the lasts can be wrongly classified. The classifier we have implemented is binary and Bayesian, so TPR and FPR we reported are located on the ROC curve in correspondence 0.5 value for the cut point. Area under curve is around 78%.

LINEAR DISCRIMINANT ANALYSIS

In light of the disappointing results of the logistic regression, the linear discriminant analysis was also tried. Results were very similar to those obtained by logistic regression but slightly worse. We could expect this, since the necessary assumption for a good linear discriminant analysis - that observations are sampled from a gaussian distribution - is not valid for this dataset.

```
Call:
lda(quality ~ volatile.acidity + residual.sugar + +alcohol, data = wwtrd.class.lda,
    family = binomial)

Prior probabilities of groups:
     high       low
0.6653047 0.3346953

Group means:
     volatile.acidity residual.sugar   alcohol
high        0.2593142       6.106858 10.813391
low         0.3066582       6.999288  9.848623

Coefficients of linear discriminants:
                         LD1
volatile.acidity  6.28774974
residual.sugar   -0.06520926
alcohol          -0.89930598
```

CONFUSION MATRIX

```
          real
predicted high low Sum
     high  577 179 756
     low    76 151 227
     Sum   653 330 983
```

ACCURACY AND MISCLASSIFICATION ERROR

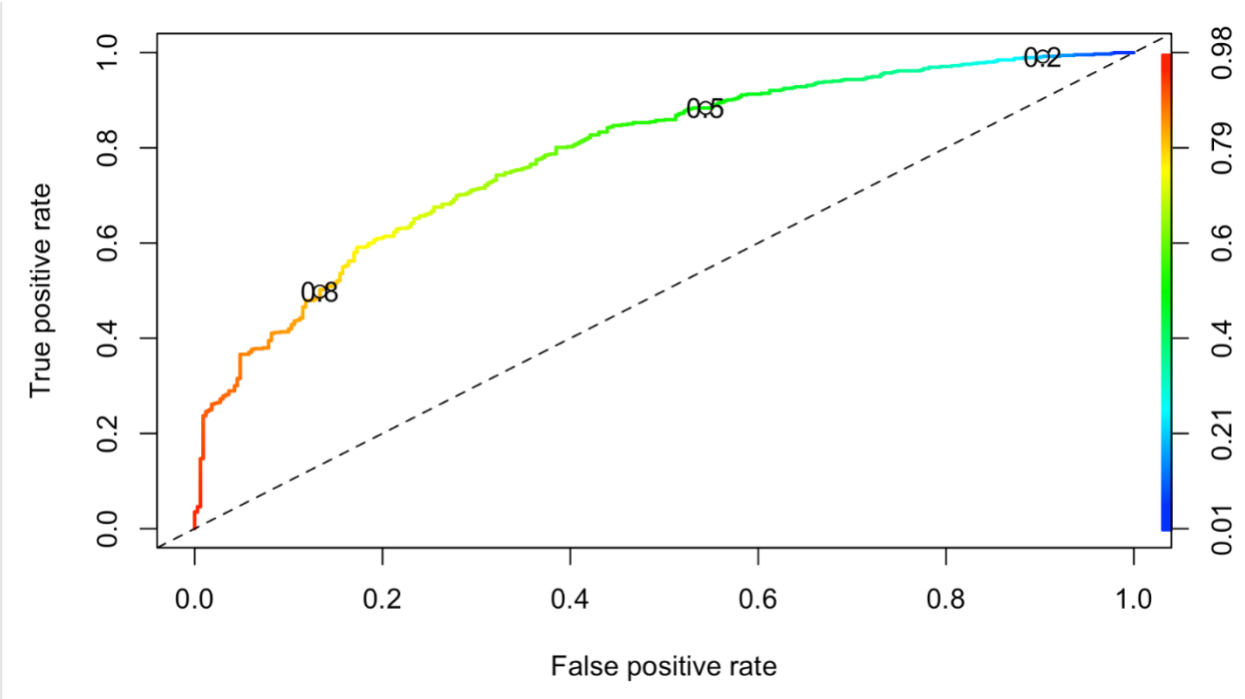
```
[1] 74.90637
[1] 25.09363
```

TRUE POSITIVE RATE. FALSE POSITIVE RATE AND ROC CURVE

True Positive Rate: `[1] 88.36141`

False Positive Rate: `[1] 54.24242`

Area Under Curve: `[1] 0.7855074`



FINAL FINDINGS:

All the methods performed give similar and disappointing performances. Choosing cross-validation as an optimal criterion to choose the regressor, we obtain slightly better results with just 3 predictors. This is enough to choose this model as the best. Alcohol, residual sugar are always selected in each subset, so they should explain most of the data variability for sure.

DECISION TREES

In this section we will describe our analysis of decision tree-based methods and how effective they were at predicting our test data quality levels. Decision tree methods can be used for regression and classification and the basic premise of a decision tree is that they define decision thresholds or splitting rules that segment the predictor space into several simple regions that are then used for prediction.

The first step in our decision tree analysis was to determine which type of decision tree would likely yield the best prediction results for our model. To make this determination we used the 4 different types of decision tree methods listed below and compared the accuracy results on the validation data.

DECISION TREE METHODS ANALYZED:

1. Basic Decision Tree Model
2. Boosting Decision Tree Model
3. Bagging Decision Tree Model
4. Random Forest Decision Tree Model

DECISION TREE METHOD RESULTS USED TO SELECT THE BEST DECISION TREE METHOD TO USE:

1. The Basic Decision Tree method confusion matrix produced an accuracy of 71.6%.

base.tree.pred	No	Yes	Sum
No	126	75	201
Yes	204	578	782
Sum	330	653	983

2. The Boosted Decision Treed method confusion matrix showed an accuracy of 75.99%

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 178  84
         1 152 569

               Accuracy : 0.7599
```

3. The Bagging Decision Tree method confusion matrix produced an accuracy of 81.6%

```
pred.bag  No Yes Sum
     No  225  76 301
     Yes 105 577 682
     Sum 330 653 983
```

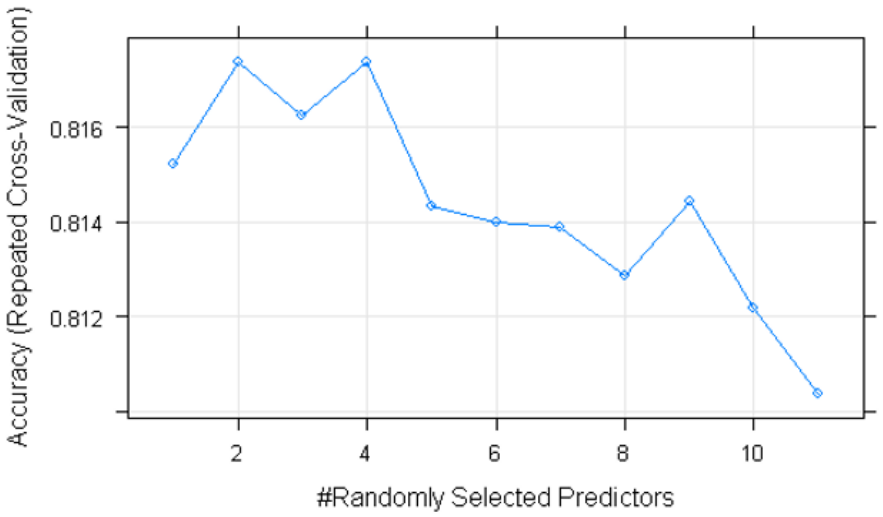
4. The Random Forest Model using default parameters showed an accuracy rate of 81.6%

```
Random Forest

2937 samples
  12 predictor
   2 classes: 'No', 'Yes'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 2643, 2643, 2643, 2643, 2644, 2644,
...
Resampling results:

  Accuracy   Kappa
  0.8159151  0.5732368

Tuning parameter 'mtry' was held constant at a value
 of 3.316625
```

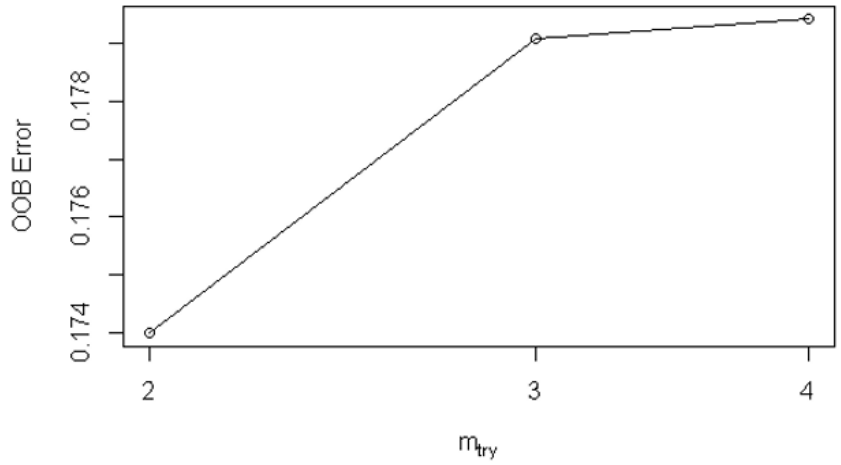
Since the default parameters Random Forest Method showed the best accuracy rate results on the validation data, we chose it as our decision tree method to use in our analysis. We then moved forward with trying to identify an optimized Random Forest Model to use for our prediction.

To optimize the Random Forest Model we used both the "*Caret*" library grid search and the "tuneRF" Algorithm tuning to determine the optimal value for the mtry. The "mtry" value in a Random Forest Model defines the number of variables randomly sampled as candidates at each split.

The Caret Grid Search Results showed that the optimal mtry value was 4 followed closely by 2.



The Algorithm Tuning Process showed that the optimal "mtry" value was clearly the value of 2.



Based on this analysis we then created a Random Forest Model using 2 as the "mtry" value and "ntree" or the number of trees to use in the model = 5000 instead of the default value of 500. This resulted in an accuracy of 82.2% on the validation data, as shown below.

OPTIMIZED RANDOM FOREST MODEL RESULTS

```
        OOB estimate of  error rate: 17.98%
Confusion matrix:
     No  Yes class.error
No  653  330   0.3357070
Yes 198 1756   0.1013306

pred.rf2  No Yes Sum
     No  215  60 275
     Yes 115 593 708
     Sum 330 653 983
```

```{r}
print((215+593)/983)
```

```
[1] 0.8219736
```

VARIABLE IMPORTANCE

	No	Yes
fixed.acidity	90.95347	92.64623
volatile.acidity	156.19539	165.23512
citric.acid	101.50625	103.97396
residual.sugar	85.62582	107.89664
chlorides	103.07575	109.18100
free.sulfur.dioxide	125.91331	113.58049
total.sulfur.dioxide	93.22523	100.11819
density	84.45668	108.37448
pH	95.40834	87.67035
sulphates	77.54477	85.72450
alcohol	161.40718	127.54405

The optimized model also showed that the 4 most valuable input variables for the prediction were *volatile.acidity*, *alcohol*, *free.sulfer.dioxide* and *density*.

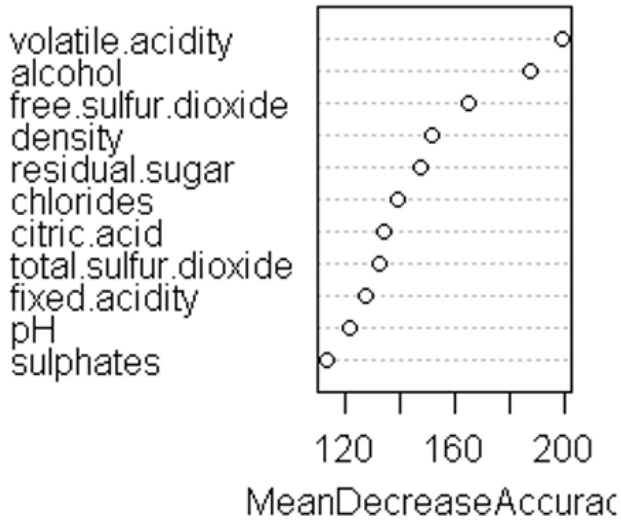
VARIABLE MEAN DECREASE IN ACCURACY

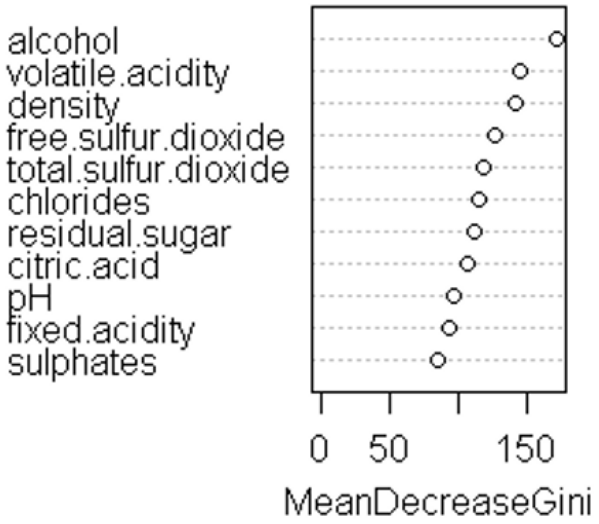
	MeanDecreaseAccuracy
fixed.acidity	126.8112
volatile.acidity	198.9177
citric.acid	133.7800
residual.sugar	147.3171
chlorides	138.8465
free.sulfur.dioxide	164.9712
total.sulfur.dioxide	132.4665
density	151.2019
pH	121.2364
sulphates	112.7569
alcohol	187.6158

VARIABLE MEAN DECREASE IN THE GINI INDEX

	MeanDecreaseGini
fixed.acidity	92.17986
volatile.acidity	144.23232
citric.acid	105.41725
residual.sugar	111.39591
chlorides	114.92775
free.sulfur.dioxide	126.86367
total.sulfur.dioxide	117.31437
density	141.55305
pH	96.45802
sulphates	84.85166
alcohol	171.31663

VARIABLE IMPORTANCE PLOTS





NEURAL NETWORKS

Finally, classification through a feed-forward neural network was tried. Keras package, which interfaces to the *tensorflow* package which in turn links to efficient python code, was used to build and fit the network. Data were prepared scaling the predictors, such that the mean of each of them was equal to 0 and the standard deviation to 1. Quality variable was converted from string to numeric ("low=0", "high"=1).

Once prepared data, the network was built. Starting from the certainty that the input layer has 11 neurons (number of predictors) and 2 output layers (corresponding to the two classes), many tries were performed in order to find the best number of layers and the best number of neurons per layer. Different rules of thumb were tested, but no substantial change in the performance was registered. Different tries were done with one or two hidden layers and various numbers of units within each layer. A single hidden layer with a large number of units, one or two hidden layers with decreasing numbers of neurons between the input shape and the output shape were tested, and various functions were used to calculate the upper limit to the number of neurons. Even though no one of these criteria was followed to the letter, they helped in getting a general idea of the values and the framework that was worthy to investigate and take the final decision.

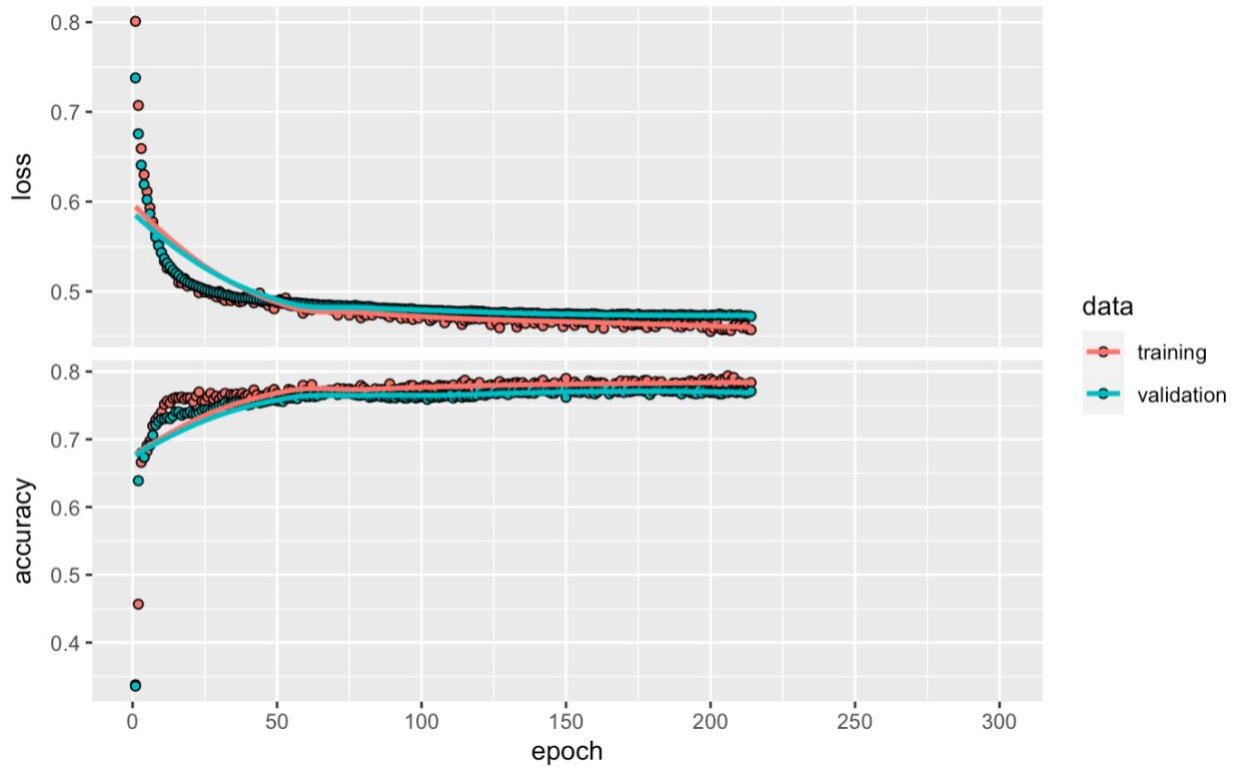
```{r message=FALSE, error=FALSE}
modelnn <- keras_model_sequential()
modelnn %>%
   layer_dense(units = 32, activation = "relu",
       input_shape = ncol(wwtrd.nn.scaled)) %>%
   layer_dropout(rate=0.2) %>%
     layer_dense(units = 4, activation = "sigmoid")%>%
   layer_dropout(rate=0.1) %>%
   layer_dense(units = 2, activation = "softmax")


summary(modelnn)
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 32)                384
_________________________________________________________________
dropout_7 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 4)                 132
_________________________________________________________________
dropout_6 (Dropout)          (None, 4)                 0
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 10
=================================================================
Total params: 526
Trainable params: 526
Non-trainable params: 0
_________________________________________________________________
```

The chosen feed-forward neural network is composed of two hidden layers: the first one contains 32 neurons (much more than the input shape), whereas the second just 4. The activation functions used are ReLU in the first layer and the sigmoid in second. The choice of the latter is motivated by the fact that it looked like to limit overfitting with respect to the ReLU, with minimum decrease in training accuracy. Overfitting is reduced also by two dropout layers at the end of the two hidden layers. The rates which gave out the best results in terms of accuracy are 0.2 for the first layer and 0.1 for the second one. In the output layer, finally, *softmax* function is used to compute the probabilities for the two classes. Batch size of 128 (46 SGD steps) looks like it returns the best results with consistent speed. Regarding epochs, an early stopping system was implemented: a maximum number of 300 epochs was set, but if a decrease of at least 0.001 in the loss metric (categorical cross-entropy) isn't registered in a range of 50 epochs, then training is stopped earlier and the model weights from the epoch with the best loss are restored and saved.

```
Epoch 289/300
46/46 [==============================] - 0s 2ms/step - loss: 0.4581 - accuracy: 0.7899 - val_loss: 0.4678 - val_accuracy: 0.7813
```



Results are quite poor. In the chosen model, accuracy stabilizes around 77-78% for both training and validation set. Better results for the training set (accuracy over 80%, or even over 85% in few

cases) were achieved with other solutions, but the worse results achieved with the validation set witnessed a consistent overfitting.

CONCLUSIONS

White wine classification succeeds only partially. **Random forests seem to be the best method** among those tested, even though accuracy still stays far from being satisfying. This is mainly related to the fact that the role of most of the predictors is not clear in determining wine quality. They lie in very similar ranges for both classes; some predictors show high collinearity, whereas others don't look like to be important at all. Furthermore, 6-rated wines are classified as high despite being "medium" (6 is the mode and the median quality) and probably this somehow brings even more difficulties in distinguishing different patterns for the two classes. However, excluding them would make the dataset too small for complex techniques like neural networks. Maybe, inspecting interactions between predictors or possible non-linear relationships in logistic regression could bring better results. However, it does seem the techniques considered don't really fit the particular features of this dataset. **After all, evaluating wines has little to do with rational metrics and widely depends on tastes, sensations and subjectivity**.

As mentioned above, of the models evaluated, the optimized random forest model achieved the best results on the validation data with an accuracy rate of 82.2%. Therefore, we want to note that running the optimized random forest model on a set of test data (see the last part of the code in the appendix) we were pleasantly surprised to achieve an accuracy of 83.3%, speaking to the stability and non-overfitting of the model developed.

APPENDIX. A

R-Notebook Code

Load the needed packages and libraries

```
```{r message=FALSE, warning=FALSE}

install.packages("xgboost")

install.packages("caret")

install.packages('mlbench')

install.packages('ROCR')

library(visdat)

library(skimr)

library(DataExplorer)

library(corrplot)

library(ggplot2)

library(reshape)

library(GGally)

library(bestglm)

library(MASS)

library(randomForest)

library(caret)

require(xgboost)

library(reshape)

library(tree)

library(gbm)

library(keras)

library(ROCR)
```
```

```

Set the working directory and load the training (wwtrd) data

```{r message=FALSE, warning=FALSE}

setwd("/Users/manuelscionti/Desktop/Materiale")

wwtrd<-read.csv('winequality-white_train.csv')

wwtrd<-subset(wwtrd, select=-X)

vis_dat(wwtrd)

```

Exploratory Data Analysis

```{r}

skim(wwtrd)

```

```{r}

plot_intro(wwtrd)

```

```{r}

plot_bar(as.factor(wwtrd$quality))

```

```{r}

plot_histogram(wwtrd[-length(wwtrd)])

```

```{r}

plot_density(wwtrd[-length(wwtrd)])

```

```{r}

plot_qq(wwtrd[-length(wwtrd)])

```

```{r}

corrplot.mixed(cor(wwtrd[-length(wwtrd)]), number.cex=0.7, tl.cex=0.7)

```

Initial Draft Data Analysis Summary

1. Luckily, there is no missing data to deal with.

2. There are 11 numeric predictor variables and 1 outcome ordinal variable.

3. The density variable appears to most closely resemble a normal distribution.

4. Since alcohol is weight less than water there is no surprise that the alcohol variable and density are very strongly negatively correlated with a -0.76 negative correlation.

5. Since the fermentation of sugar is what produces the alcohol it is also not surprising that residual.sugar and alcohol have a negative correlation of -0.45.

6. Residual.sugar and density have the strongest correlation of 0.85 which is due to the fact that sugar is heavier than water so where there is more residual.sugar in the wine the wine is more dense or heavy.

7. Looking at the variable scatter plots as compared to quality it looks like higher quality wines are generally higher in alcohol, lower in residual sugar and density that the lower quality wines.

```{r}

str(wwtrd)

wwtrd.class<-wwtrd

wwtrd.class[wwtrd.class$quality<6, 'quality']<-'low'

wwtrd.class[wwtrd.class$quality!='low' & wwtrd.class$quality>6, 'quality']<-'high'

wwtrd.class[wwtrd.class$quality!='low' & wwtrd.class$quality!='high', 'quality']<-'medium'

wwtrd.class$quality<-factor(wwtrd.class$quality, levels=c('low', 'medium', 'high'))
```
```

```
summary(wwtrd.class$quality)
```

```{r}
ggplot(wwtrd.class, aes(x=quality))+geom_bar(aes(fill=quality))
```

```{r message=FALSE, warning=FALSE}
meltData <- melt(wwtrd.class)
meltData$variable<-factor(meltData$variable)
ggplot(meltData, aes(variable, value))+ geom_boxplot() + facet_wrap(~variable,scale="free")
```

```{r warning= FALSE, message=FALSE}
ggp = ggpairs(wwtrd, progress=TRUE, upper = list(continuous = wrap("cor", size = 2.5)),
        lower = list(continuous = wrap("points", alpha = 0.2, size=0.1),
            combo = wrap("dot", alpha = 1, size=0.2) ))+theme_grey(base_size = 5.3)
print(ggp, progress = FALSE)
```

To get a deeper understanding of the predictors' interactions, corrplot() functions perfomrs hierarchical clustering in two groups, based on the correlation values. This can help us in finding similar trend for variables belonging to the same clusters.

```{r}
corrplot(cor(wwtrd[-length(wwtrd)]), method="number", order="hclust", number.cex=0.7, addrect=2)
```

```{r warning= FALSE, message=FALSE}}
ggp2 <- ggpairs(wwtrd.class, aes(color=quality),
        upper = list(continuous = wrap("cor", size = 2)),
        lower = list(continuous = wrap("points", alpha = 0.2, size=0.05),
```

```
            combo = wrap("dot", alpha = 1, size=0.2) ))+theme_grey(base_size = 5.3)

print(ggp2, progress = FALSE)

```

```{r}

wwtrd.mean.aggr<-aggregate(meltData$value, by=list(meltData$quality,meltData$variable), mean)

colnames(wwtrd.mean.aggr)<-c('quality', 'variable', 'mean')



wwtrd.sd.aggr<-aggregate(meltData$value, by=list(meltData$quality,meltData$variable), sd)

colnames(wwtrd.sd.aggr)<-c('quality', 'variable', 'sd')

```

```{r}

ggplot(meltData[meltData$variable %in% c('fixed.acidity', 'citric.acid'),], aes(value, color=quality))+

geom_density()+facet_wrap(~variable,scales="free")

```



```{r}

ggplot(meltData[meltData$variable %in% c('fixed.acidity', 'citric.acid'),], aes(variable, value, fill=quality))+

geom_boxplot(notch=T) + facet_wrap(~variable,scale="free")

```

```{r}

subset(wwtrd.mean.aggr, variable %in% c('fixed.acidity', 'citric.acid'))

subset(wwtrd.sd.aggr, variable %in% c('fixed.acidity', 'citric.acid'))

```

```{r}

ggplot(meltData[meltData$variable %in% c('chlorides','residual.sugar',

'density','free.sulfur.dioxide','total.sulfur.dioxide'),], aes(value, color=quality))+

geom_density()+facet_wrap(~variable,scales="free")
```

```
```{r}
ggplot(meltData[meltData$variable %in% c('chlorides','residual.sugar',
'density','free.sulfur.dioxide','total.sulfur.dioxide'),], aes(variable, value, fill=quality))+ geom_boxplot(notch=T) +
facet_wrap(~variable,scale="free")
```
```{r}
subset(wwtrd.mean.aggr, variable %in% c('chlorides','residual.sugar',
'density','free.sulfur.dioxide','total.sulfur.dioxide'))
subset(wwtrd.sd.aggr, variable %in% c('chlorides','residual.sugar', 'density','free.sulfur.dioxide','total.sulfur.dioxide'))
```
```{r}
ggplot(meltData[meltData$variable %in% c('pH', 'sulphates'),], aes(value, color=quality))+geom_density() +
facet_wrap(~variable,scales="free")
```
```{r}
ggplot(meltData[meltData$variable %in% c('pH', 'sulphates'),], aes(variable, value, fill=quality))+
geom_boxplot(notch=T) + facet_wrap(~variable,scale="free")
```
```{r}
subset(wwtrd.mean.aggr, variable %in% c('pH', 'sulphates'))
subset(wwtrd.sd.aggr, variable %in% c('pH', 'sulphates'))
```
```{r}
ggplot(meltData[meltData$variable %in% c('volatile.acidity', 'alcohol'),], aes(value, color=quality))+geom_density() +
facet_wrap(~variable,scales="free")
```
```

```{r}
ggplot(meltData[meltData$variable %in% c('volatile.acidity', 'alcohol'),], aes(variable, value, fill=quality))+
geom_boxplot(notch=T) + facet_wrap(~variable,scale="free")
```

```{r}
subset(wwtrd.mean.aggr, variable %in% c('volatile.acidity', 'alcohol'))
subset(wwtrd.sd.aggr, variable %in% c('volatile.acidity', 'alcohol'))
```

Removing all the 6-ranked wines ("the medium" class), for which is rather difficult detecting characteristic features - they are average wines, after all - the scatterplot shows that the points are quite well separated for each couple of variables, reflecting the opposite qualities of the wines.

```{r}
ggp3 <- ggpairs(wwtrd.class[wwtrd.class$quality!='medium',], aes(color=quality),
        upper = list(continuous = wrap("cor", size = 2)),
        lower = list(continuous = wrap("points", alpha = 0.2, size=0.1),
             combo = wrap("dot", alpha = 1, size=0.2) ))+theme_grey(base_size = 5.3)
print(ggp3, progress = FALSE)
```

Thanks to the library _bestglm_ the best logistic regressor is found according to different criteria.

Logistic Regression

```{r}
# Find the best logistic regressor - Data preparation
wwtrd.class[wwtrd.class$quality=='medium', 'quality']<-'high'
wwtrd.class$quality<-factor(wwtrd.class$quality, levels=c('low', 'high'))
```

```
wwtrd.class.glm <- within(wwtrd.class, {
    y    <- quality         # bwt into y
    quality  <- NULL        # Delete bwt
})

wwtrd.class.glm$y<-as.character(wwtrd.class.glm$y)
wwtrd.class.glm[wwtrd.class.glm$y=='low',]$y<-0
wwtrd.class.glm[wwtrd.class.glm$y=='high','y']<-1
wwtrd.class.glm$y<-as.integer(wwtrd.class.glm$y)
```

Most of the predictors present very low collinearity. Density especially, but also alcohol and residual sugar have an important level of collinearity. This addresses to not include all of them in the selected regressor.

```{r}
#Collinearity of the predictors
vifx(wwtrd.class.glm[-12])
```

```{r}
# Best regressor according to AIC
best.reg.aic<-bestglm(wwtrd.class.glm, family=binomial, IC='AIC')
best.reg.aic
```

Even though this regressor minimizes AIC, Std.Error for density is really too big with respect to its estimate. There's a lot of uncertainty on the intercept too and also other predictors show consistent std.errors in comparison to their estimates.

```{r}
```

```
#Validation - data preparation

setwd("/Users/manuelscionti/Desktop/Materiale")

wwvd<-read.csv('winequality-white_validation.csv')

wwvd<-subset(wwvd, select=-X)



#Validation

glm.probs.aic<-predict(best.reg.aic$BestModel, wwvd, type='response')

#probability of being high-quality wine

head(glm.probs.aic)
```

```{r}
glm.pred.aic<-glm.probs.aic

glm.pred.aic[glm.probs.aic<.5]<-"low"

glm.pred.aic[glm.probs.aic>.5]<-"high"



#predicted class

head(glm.pred.aic)
```

```{r}
#Confusion matrix

table(glm.pred.aic,wwvd$quality)

confMat.aic<-addmargins(table(glm.pred.aic,wwvd$quality, dnn=c('predicted', 'real')))

confMat.aic
```

```{r}
#Accuracy and error rate

accuracy.aic<-(confMat.aic[1,1]+confMat.aic[2,2])/nrow(wwvd)*100
```

```
Err.aic<-100-accuracy.aic   # misclassification error

accuracy.aic

Err.aic
```

```{r}
# Best regressor according to BIC

best.reg.bic<-bestglm(wwtrd.class.glm, family=binomial, IC='BIC')

best.reg.bic
```

The proportion of the standard error is not so better compared to previous methood.

```{r}
glm.probs.bic<-predict(best.reg.bic$BestModel, wwvd, type='response')

head(glm.probs.bic)
```

```{r}
glm.pred.bic<-glm.probs.bic

glm.pred.bic[glm.probs.bic<.5]="low"

glm.pred.bic[glm.probs.bic>.5]="high"

#predicted class

head(glm.pred.bic)
```

```{r}
#Confusion matrix

table(glm.pred.bic,wwvd$quality)

confMat.bic<-addmargins(table(glm.pred.bic,wwvd$quality, dnn=c('predicted', 'real')))

confMat.bic
```

```

```{r}

#Accuracy and error rate

accuracy.bic=(confMat.bic[1,1]+confMat.bic[2,2])/nrow(wwvd)*100

err.bic=100-accuracy.bic

accuracy.bic

err.bic

```

```{r}

# Best regressor according to cross-validation

best.reg.cv<-bestglm(wwtrd.class.glm, family=binomial, IC='CV', CVArgs = list(Method='HTF', K=11, REP=5))

best.reg.cv

```

Simpler method with best p-values and std. errors. As you can see from the following chunk, the percentual loss in terms of AIC and BIC from the best methods is really small.

```{r}

AIC(best.reg.cv$BestModel)

BIC(best.reg.cv$BestModel)



((AIC(best.reg.aic$BestModel)-AIC(best.reg.cv$BestModel))/AIC(best.reg.aic$BestModel))*100

((BIC(best.reg.bic$BestModel)-BIC(best.reg.cv$BestModel))/BIC(best.reg.aic$BestModel))*100

```

```{r}

#probability of being high-quality wine

glm.probs.cv<-predict(best.reg.cv$BestModel, wwvd, type='response')

head(glm.probs.cv)
```
```

```

```{r}

glm.pred.cv<-glm.probs.cv

glm.pred.cv[glm.probs.cv<.5]<-"low"

glm.pred.cv[glm.probs.cv>.5]<-"high"

#predicted class

head(glm.pred.cv)

```

#Confusion matrix

table(glm.pred.cv,wwvd$quality, dnn=c('predicted', 'real'))

confMat.cv<-addmargins(table(glm.pred.cv,wwvd$quality, dnn=c('predicted', 'real')))

confMat.cv

```

#Accuracy and error rate

accuracy.cv<-(confMat.cv[1,1]+confMat.cv[2,2])/nrow(wwvd)*100

Err.cv<-100-accuracy.cv   # misclassification error

accuracy.cv

Err.cv

```

#TPR and FPR

TPR.cv<-confMat.cv[1,1]/(confMat.cv[1,1]+confMat.cv[2,1])*100

FPR.cv<-confMat.cv[1,2]/(confMat.cv[1,2]+confMat.cv[2,2])*100

TPR.cv

FPR.cv

```

ROC curve

```
pred.t.LR_T=prediction(glm.probs.cv, wwvd$quality, c('low', 'high'))

perf.t.LR_T=performance(pred.t.LR_T, measure = "tpr", x.measure = "fpr")

plot(perf.t.LR_T,colorize=TRUE,lwd=2)

plot(perf.t.LR_T,colorize=TRUE,lwd=2, print.cutoffs.at=c(0.2,0.5,0.8))

abline(a=0,b=1, lty=2)

LR_T.auc=performance(pred.t.LR_T, measure = "auc")

LR_T.auc@y.values[[1]]
```

Linear Discriminant Analysis

```{r}
wwtrd.class.lda<-wwtrd.class

wwtrd.class.lda$quality<-as.character(wwtrd.class.lda$quality)

wwtrd.class.lda[wwtrd.class.lda$quality=='medium' | wwtrd.class.lda$quality=='high', 'quality']<-1

wwtrd.class.lda[wwtrd.class.lda$quality!=1, 'quality']<-0

wwtrd.class.lda

lda.fit<-lda(quality~volatile.acidity+residual.sugar+alcohol, family=binomial, data=wwtrd.class.lda)

lda.fit
```

```{r}
lda.predict=predict(lda.fit, wwvd, type='response')

lda.pred.class<-lda.predict$class

head(lda.pred.class)
```

```{r}
```

```
confMat.LDA<-addmargins(table(lda.pred.class,wwvd$quality, dnn=c('predicted', 'real')))

confMat.LDA

```

```{r}

accuracy.LDA=(confMat.LDA[1,1]+confMat.LDA[2,2])/nrow(wwvd)*100 # accuracy

Err.LDA=100-accuracy.LDA # misclassification error

accuracy.LDA

Err.LDA

```

```{r}

TPR.LDA<-confMat.LDA[1,1]/(confMat.LDA[1,1]+confMat.LDA[2,1])*100

FPR.LDA<-confMat.LDA[1,2]/(confMat.LDA[1,2]+confMat.LDA[2,2])*100



TPR.LDA

FPR.LDA

```

```{r}

pred.t.LDA_T<-prediction(lda.predict$posterior[,1], wwvd$quality, c('low','high'))

perf.t.LDA_T<-performance(pred.t.LDA_T, measure = "tpr", x.measure = "fpr")

plot(perf.t.LDA_T,colorize=TRUE,lwd=2, print.cutoffs.at=c(0.2,0.5,0.8))

abline(a=0,b=1, lty=2)```

```{r}

LDA_T.auc=performance(pred.t.LDA_T, measure = "auc")

LDA_T.auc@y.values[[1]]

```
```

All the methods performed give similar and disappointing performances. Choosing cross-validation as optimal criterion to choose the regressor, we obtain slightly better results with just 3 predictors. This is enough to choose this model as the best. Alcohol, residual sugar are always selected in each subset, so they should explain most of the data variability for sure.

Decision Tree Analysis

Base Tree & Pruned Base Tree

```{r}
wwtrd$qlevels <- factor(ifelse(wwtrd$quality <= 5, "No", "Yes"))
table(wwtrd$qlevels)
table(wwtrd$quality)
setwd("/Users/manuelscionti/Desktop/Materiale")
wwvald<-read.csv('winequality-white_validation.csv')
wwvald<-subset(wwvald, select=-X)
wwvald$qlevels <- factor(ifelse(wwvald$quality <= 5, "No", "Yes"))
table(wwvald$qlevels)
table(wwvald$quality)
```

```{r}
base.tree.fit = tree(qlevels ~ . -quality, data=wwtrd)
```

```{r}
summary(base.tree.fit)
```

```{r}
base.tree.fit
```

```{r}
```

```
set.seed(99)

base.tree.pred <- predict(base.tree.fit, wwvald, type = "class")

print(addmargins(table(base.tree.pred , wwvald$qlevels)))
```

```{r}
print((126+578)/983) #shows the accuracy rate on the validation data
```

```{r}
summary(base.tree.pred)
```

```{r}
#Now let us prun the base tree to see if we get a better result on the validation data.

cv.wwtrd.base.tree <- cv.tree(base.tree.fit , FUN = prune.misclass)
```

```{r}
cv.wwtrd.base.tree
```

```{r}
par(mfrow = c(1, 2))

plot(cv.wwtrd.base.tree$size, cv.wwtrd.base.tree$dev, type = "b")

plot(cv.wwtrd.base.tree$k, cv.wwtrd.base.tree$dev, type = "b")
```

```{r}
prune.wwtrd.basetree <- prune.misclass(base.tree.fit , best = 6)

plot(prune.wwtrd.basetree)

text(prune.wwtrd.basetree , pretty = 0)
```

```{r}

prun.tree.pred <- predict(prune.wwtrd.basetree , wwvald, type = "class")

table(prun.tree.pred , wwvald$qlevels)

```

```{r}

print(addmargins(table(prun.tree.pred , wwvald$qlevels)))

```

```{r}

print((126+578)/983) #shows the accuracy rate on the test data  #It is interesting how the optimal pruned model at

71.6% was the same as the basic model at 71.6%.

```

Boosting - Now let us see how a boosting model does on the data.

#SINCE GBM requires the response to be 0 or 1, I will modify wwtrd & wwvald. Instructions to run model found at:

https://rpubs.com/omicsdata/gbm

```{r}

setwd("/Users/manuelscionti/Desktop/Materiale")

xgb.train<-read.csv('winequality-white_train.csv')

xgb.train<-subset(xgb.train, select=-X)

xgb.vald<-read.csv('winequality-white_validation.csv')

xgb.vald<-subset(xgb.vald, select=-X)

```

```{r}

xgb.train.f<-xgb.train

xgb.train.n<-xgb.train

xgb.vald.f<-xgb.vald

xgb.vald.n<-xgb.vald
```

```
xgb.train.f$quality <- factor(ifelse(xgb.train$quality <= 5, "0", "1"))

xgb.train.n$quality <- as.numeric(ifelse(xgb.train$quality <= 5, "0", "1"))

xgb.vald.f$quality <- factor(ifelse(xgb.vald$quality <= 5, "0", "1"))

xgb.vald.n$quality <- as.numeric(ifelse(xgb.vald$quality <= 5, "0", "1"))

table(xgb.train$quality)

table(xgb.vald$quality)
```

```{r}
gbm.model = gbm(quality~., data=xgb.train.n, shrinkage=0.01, distribution = 'bernoulli', cv.folds=5, n.trees=3000,

verbose=F)
```

```{r}
best.iter = gbm.perf(gbm.model, method="cv")
```

```{r}
best.iter
```

```{r}
summary(gbm.model)
```

```{r}
plot.gbm(gbm.model, 1, best.iter)

plot.gbm(gbm.model, 2, best.iter)

plot.gbm(gbm.model, 3, best.iter)

plot.gbm(gbm.model, 4, best.iter)

plot.gbm(gbm.model, 5, best.iter)

plot.gbm(gbm.model, 6, best.iter)
```

```
plot.gbm(gbm.model, 7, best.iter)

plot.gbm(gbm.model, 8, best.iter)

plot.gbm(gbm.model, 9, best.iter)

plot.gbm(gbm.model, 10, best.iter)

plot.gbm(gbm.model, 11, best.iter)
```

```{r}
mydata=xgb.train.f

set.seed(123)

fitControl = trainControl(method="cv", number=5, returnResamp = "all")

model2 = train(quality~., data=mydata, method="gbm",distribution="bernoulli", trControl=fitControl, verbose=F,

tuneGrid=data.frame(.n.trees=best.iter, .shrinkage=0.01, .interaction.depth=1, .n.minobsinnode=1))
```

```{r}
model2
```

```{r}
summary(model2)
```

```{r}
confusionMatrix(model2)
```

```{r}
mPred = predict(model2, xgb.vald.f, na.action = na.pass)

postResample(mPred, xgb.vald.f$quality)
```

```{r}
```

confusionMatrix(mPred, xgb.vald.f$quality)

```

```{r}

#Now let us try Bagging and RandomForest Trees

set.seed(99)

bag.wwtrd<-randomForest(qlevels ~. - quality, data = wwtrd, mtry = 11, importance = TRUE, type = class)

```

```{r}

bag.wwtrd

```

```{r}

pred.bag <- predict(bag.wwtrd , wwvald, type="class")

print(addmargins(table(pred.bag , wwvald$qlevels)))

```

```{r}

print((225+577)/983)

#The accuracy results of the bagging tree that uses all variables when branching is 81.6%.

```

Random Forest Model

Now we will look at the random forest model in detail to find the optimal random forest model for the data. First I will run a default random forest model then I will hyper tune the random forest model to find the optimal model parameters using a grid search and then I will use the tuneRF algorithm. Based on the results we will choose the optimal mtry parameter for the random forest model.

```{r}

x<-wwtrd[,1:11]

y<-wwtrd[,13]

```

```{r}

# Create model with default parameters

control <- trainControl(method="repeatedcv", number=10, repeats=3)

seed <- 99

metric <- "Accuracy"

set.seed(seed)

mtry <- sqrt(ncol(x))

tunegrid <- expand.grid(.mtry=mtry)

rf_default <- train(qlevels~. - quality, data=wwtrd, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control)

print(rf_default)

```

TUNE THE MODEL USING GRID SEARCH

```{r}

control <- trainControl(method="repeatedcv", number=3, repeats=1, search="grid")

set.seed(seed)

tunegrid <- expand.grid(.mtry=c(1:11))

rf_gridsearch <- train(qlevels~. -quality, data=wwtrd, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control)

print(rf_gridsearch)

plot(rf_gridsearch)

```

```{r}

# Algorithm Tune (tuneRF)

set.seed(seed)

bestmtry <- tuneRF(x, y, stepFactor=1.5, improve=1e-5, ntree=500)

print(bestmtry)

```
```

Now we will use mtry = 2 on the validaton data

```{r}

set.seed(99)

rf2.wwtrd<-randomForest(qlevels ~. - quality, data = wwtrd, mtry = 2, importance = TRUE, type = class, ntree = 5000)

rf2.wwtrd

pred.rf2 <- predict(rf2.wwtrd , wwvald, type="class")

```

```{r}

print(addmargins(table(pred.rf2 , wwvald$qlevels)))

```

```{r}

print((215+593)/983)

```

```{r}

importance(rf2.wwtrd)

```

```{r}

varImpPlot(rf2.wwtrd)

```

Neural Network Model

```{r}

library('keras')

wwtrd.nn<-wwtrd.class

wwvd.nn<-wwvd[,1:12]

```

```
wwtrd.nn$quality<-as.character(wwtrd.nn$quality)

wwtrd.nn[wwtrd.nn$quality=='high', 'quality']<-1

wwtrd.nn[wwtrd.nn$quality!=1, 'quality']<-0



wwvd.nn$quality<-as.character(wwvd.nn$quality)

wwvd.nn[wwvd.nn$quality=='high', 'quality']<-1

wwvd.nn[wwvd.nn$quality!=1, 'quality']<-0



wwtrd.nn.scaled<-scale(subset(wwtrd.nn, select = - quality))

wwvd.nn.scaled<-scale(subset(wwvd.nn, select = - quality))

x_train <- array_reshape(wwtrd.nn.scaled, c(nrow(wwtrd.nn.scaled), ncol(wwtrd.nn.scaled)))

x_valid <- array_reshape(wwvd.nn.scaled, c(nrow(wwvd.nn.scaled), ncol(wwvd.nn.scaled)))



y_train <- to_categorical(wwtrd.nn$quality, 2)

y_valid <- to_categorical(wwvd.nn$quality, 2)

wwtrd.nn
```

```{r message=FALSE, error=FALSE}

modelnn <- keras_model_sequential()

modelnn %>%

  layer_dense(units = 32, activation = "relu",

      input_shape = ncol(wwtrd.nn.scaled)) %>%

  layer_dropout(rate=0.2) %>%

    layer_dense(units = 4, activation = "sigmoid")%>%

  layer_dropout(rate=0.1) %>%

  layer_dense(units = 2, activation = "softmax")
```

```{r}
summary(modelnn)
```

```{r}
modelnn %>% compile(loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(), metrics = c("accuracy")
  )

callback<-callback_early_stopping(
    monitor="val_loss",
    patience=50,
    min_delta = 0.001,
    mode="min",
    restore_best_weights = TRUE
)
```

```{r}
   system.time (
     history <- modelnn%>%
      fit(x_train, y_train, epochs = 300, batch_size=64,
          validation_data = list(x_valid, y_valid),
          callback=callback)
   )
```

```{r}
   plot(history)
```

```{r}
accuracy <- function(pred, truth) {
  mean(drop(as.numeric(pred)) == drop(truth)) }
modelnn %>% predict(x_valid) %>% k_argmax() %>% accuracy(wwvd.nn$quality)
```

Best Model (Random Forest) Ran on Test Dataset

Now we will run the best model on the test data. We will use the Random Forest model since it produced the highest accuracy on the validation data.

```{r}
wwtest<-read.csv('winequality-white_test_wquality.csv')
#wwtest<-subset(wwtest, select=-index)
wwtest$qlevels <- factor(ifelse(wwtest$quality <= 5, "No", "Yes"))
head(wwtest)
```

```{r}
pred.rf2_test <- predict(rf2.wwtrd, wwtest, type="class")
print(addmargins(table(pred.rf2_test , wwtest$qlevels)))
```

```{r}
print((223+592)/978)
```

```{r}
results <-as.integer(pred.rf2_test)
results1 <-as.data.frame(results)
test_with_results <-cbind(wwtest, results1)
```

```
test_with_results$results <- factor(ifelse(test_with_results$results == 1, "No", "Yes"))

head(test_with_results, 50)

write.csv(test_with_results,"test_file_w_final_results.csv", row.names = FALSE)

```
```
```