# HarvardX (PH125.9x) - Red Wine Quality

Konstantinos Liakopoulos

12/8/2021

## Contents



## Overview

This report is related to the HarvardX: PH125.9x Data Science: Capstone.

The objective of this analysis is to find the factors that separates good from bad wine and build a model that can predict the quality of the wine.

It consists of six parts:

It starts with this **Overview** that summarizes this analysis document contents

The **Introduction** sections provides the goal of the analysis and the dataset that will be used.

In **Data analysis** section an exploratory data analysis of the dataset is performed

In **Methods and Analysis** section, machine learning algorithms are developed that predicts and separate the best wine based on their quality from the rest

**Results** section presents the modeling results and discusses the model performance.

**Conclusion** is the summary of what was learned from this analysis and if its goal was achieved.

# Introduction

I am Greek, so naturally I am also fan of good red wine.In Greek mythology, Dionysus is the god of wine and a major figure of the Olympian pantheon. The divine functions of both wine and Dionysus are often connected. In this analysis i will try to find the factors that separates good from bad (and mediocre wine).

In order to develop a red wine quality model, I will use data from Kraggle https://www.kaggle.com/uciml/red-wine-quality-cortez-et-al-2009 that original donated to UCI Machine learning Repository by P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. For more details, consult the following link https://archive.ics.uci.edu/ml/datasets/wine+quality Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

Import libraries and seed

```
if (!require("rpart")) install.packages("pacman")
if (!require("corrplot")) install.packages("corrplot")
if (!require("dplyr")) install.packages("dplyr")
if (!require("caret")) install.packages("caret", dependencies = TRUE)
if (!require("ggplot2")) install.packages("ggplot2")
if (!require("rpart.plot")) install.packages("rpart.plot")
if (!require("corrplot")) install.packages("corrplot")
if (!require("pROC")) install.packages("pROC")
if (!require("tidyr")) install.packages("tidyr")
if (!require("e1071")) install.packages('e1071', dependencies=TRUE)
if (!require("randomForest")) install.packages("randomForest")


library(dplyr)
library(rpart)
library(caret)
library(rpart.plot)
library(ggplot2)
library(corrplot)
library(pROC)
library(tidyr)
library(e1071)
library(randomForest)


set.seed(1, sample.kind="Rounding")
```

# Data Analysis

Use the read.csv function to load the data. Afterwards loop through all columns and check if something is missing.

```
wine<-read.csv("winequality-red.csv")


missing<- wine %>%
  is.na() %>%
```

```
  colSums()
data.frame(missing)
```

```
##                       missing
## fixed.acidity               0
## volatile.acidity            0
## citric.acid                 0
## residual.sugar              0
## chlorides                   0
## free.sulfur.dioxide         0
## total.sulfur.dioxide        0
## density                     0
## pH                          0
## sulphates                   0
## alcohol                     0
## quality                     0
```

Data are good and there are no missing values. It is time to explore the data.

```
str(wine)
```

```
## 'data.frame':    1599 obs. of  12 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
```

The data frame has 1599 observations and 12 variables:

1 - fixed acidity

2 - volatile acidity

3 - citric acid

4 - residual sugar

5 - chlorides

6 - free sulfur dioxide

7 - total sulfur dioxide

8 - density

9 - pH

10 - sulphates

11 - alcohol

12 - quality (score between 0 and 10)

Lets examine the data to get more information about its distribution and research if it has outliers or other problems

```
summary(wine$quality)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   5.000   6.000   5.636   6.000   8.000
```

```
summary(wine)
```

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

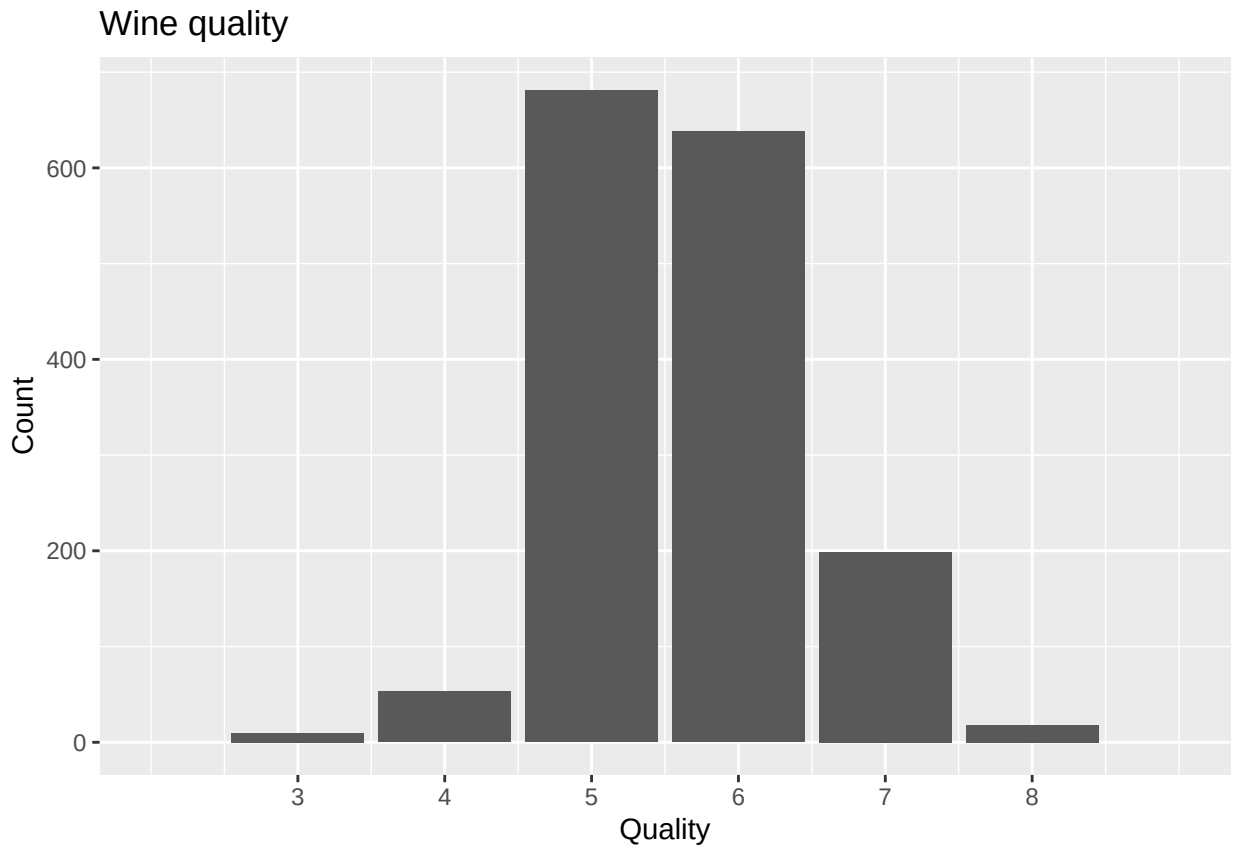
```
table(wine$quality)
```

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

```
prop.table(table(wine$quality))
```

```
##
##           3           4           5           6           7           8
## 0.006253909 0.033145716 0.425891182 0.398999375 0.124452783 0.011257036
```

```
ggplot(data.frame(wine), aes(x=quality)) +
  geom_bar()+
  scale_x_continuous(limits = c(2, max(wine$quality)+1), breaks = round(seq(3,8)))+
 xlab("Quality") +
  ylab("Count") +
  ggtitle("Wine quality")
```




The data seems to follow a normal bell shaped distribution centered around five and then six. Most wines are average. Moreover we have very few values at both ends of the range of quality.

```
mean(wine$quality)
```

```
## [1] 5.636023
```

```
sd(wine$quality)
```

```
## [1] 0.8075694
```

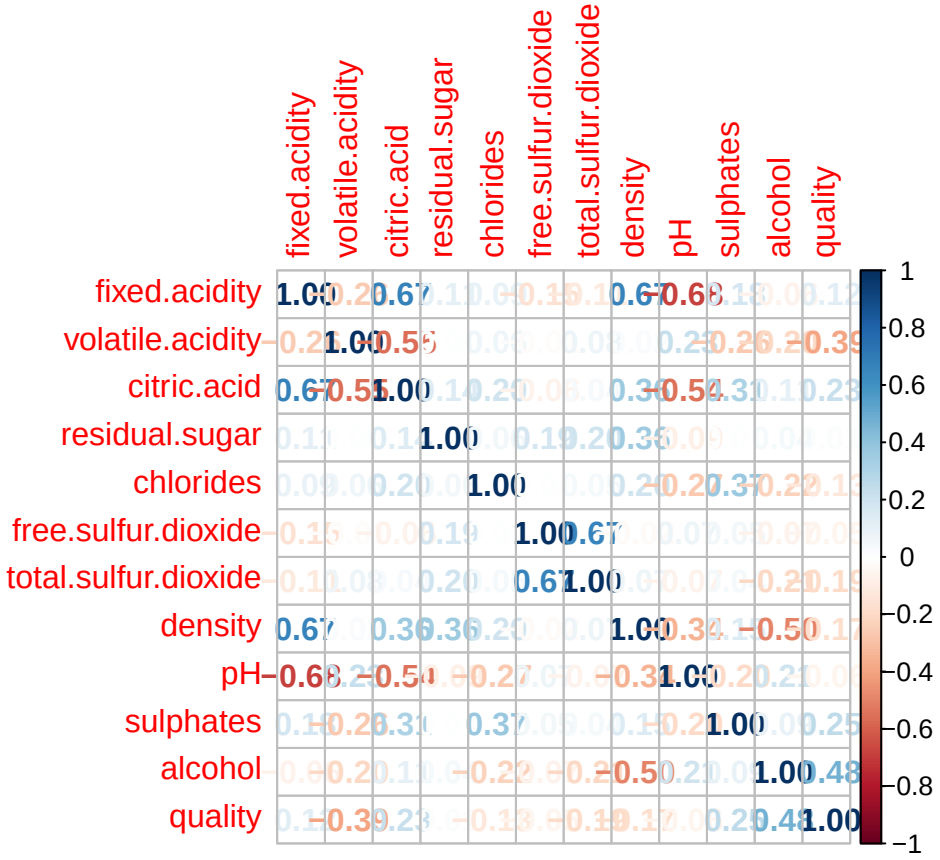
```
quantile(as.integer(wine$quality), .95)
```

```
## 95%
##   7
```

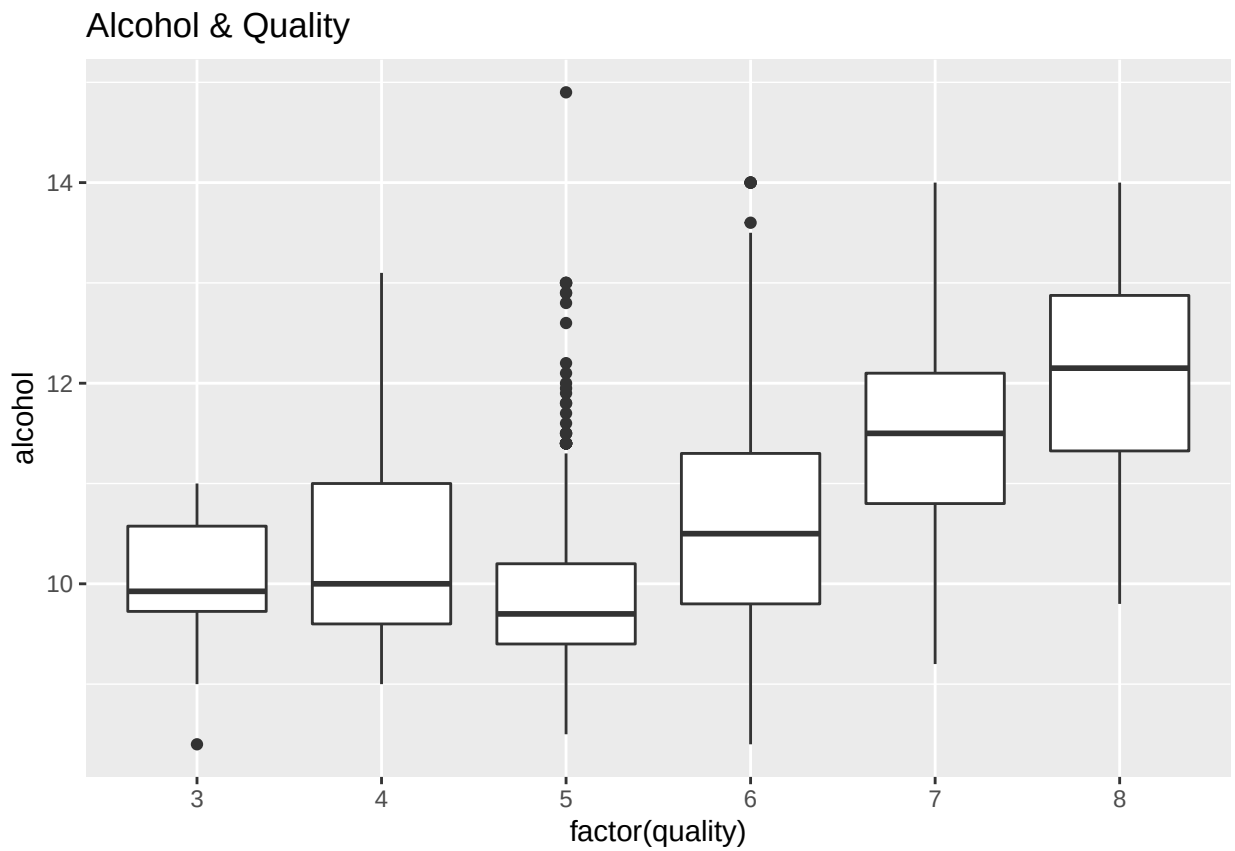
By searching to find a good cutoff to separate the bad (and average wines) from the really good ones, wine quality of 7 is the best candidate.

A graphical display of a correlation matrix shows that alcohol, sulphates and volatile.acitidy has the biggest correlation with wine quality.

```
corrplot(cor(wine), method="number")
```




```
wine%>%ggplot(aes(factor(quality), alcohol, group=quality))+geom_boxplot()+ggtitle("Alcohol & Quality")
```




Boxplot shows that alchocol, the variable that affects mostly quality, has many outliers at the quality value of 5 and this can affect the process of seperating the good from the bad wines.

A factor is created named status with two values good or bad based on wine quality. This is the output value that will be predicted.

```
wine$status = factor(ifelse(wine$quality >=  7, "good", "bad"))
wine<-mutate(wine%>%dplyr::select(-quality))
```

```
table(wine$status)
```

```
##
##  bad good
## 1382  217
```

```
prop.table(table(wine$status))
```

```
##
##       bad      good
## 0.8642902 0.1357098
```

13.5% of the wines are of good quality.

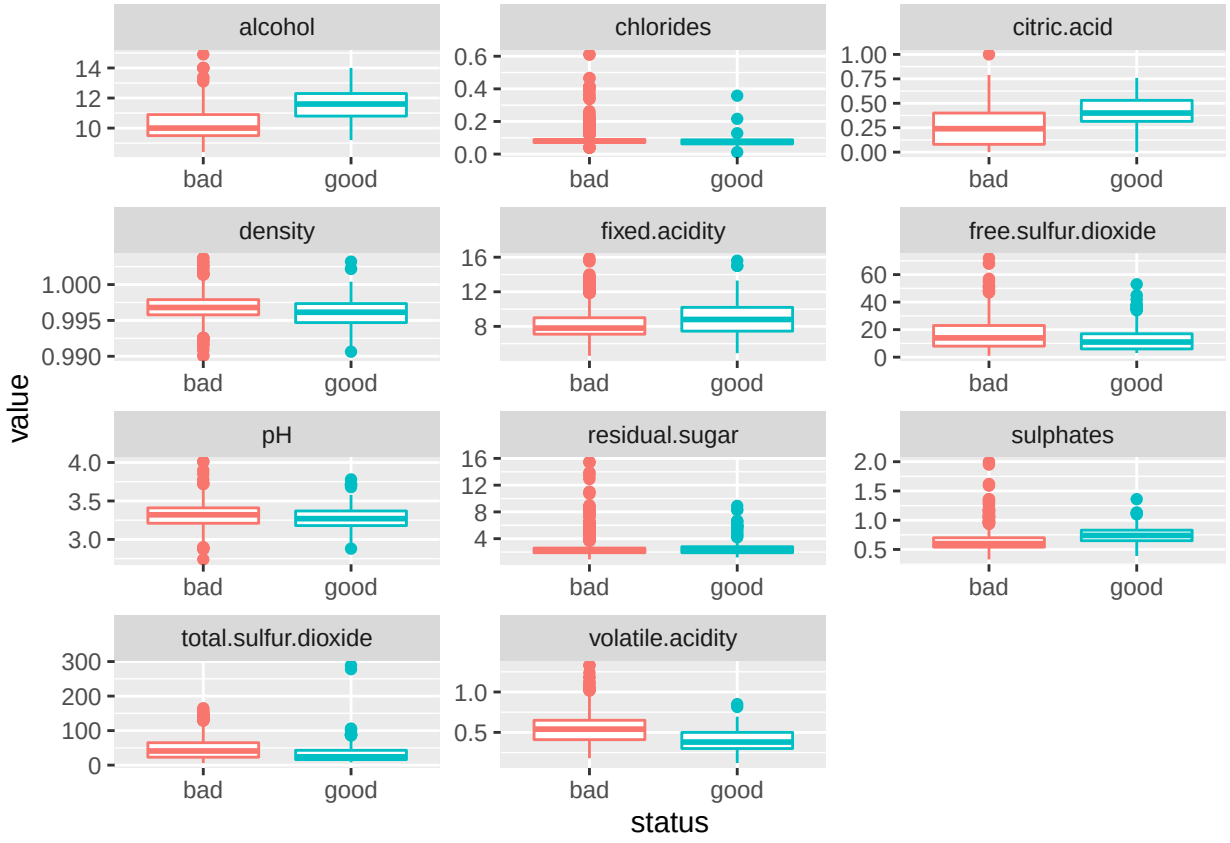
## Methods Analysis

The dataset needs to be partitioned to set up a train dataset and a test dataset.

```
test_index <- createDataPartition(wine$status, times = 1, p = 0.75, list = FALSE)

wine_train<-wine[test_index ,]

wine_test<-wine[-test_index ,]
```

A box plot of each of the predictors visually evaluates the train data.

```
wine_train%>%
  gather(-status, key = "var", value = "value") %>%
  ggplot(aes(x = status, y = value, color = status)) +
  geom_boxplot() +
  facet_wrap(~ var, scales = "free", ncol = 3)+
  theme(legend.position="none")
```




## Regression Tree Model

I begin by training a regression model tree using the default values.

```
rpart_model<-train(status~.,data=wine_train, method = "rpart")
rpart_model
```

```
## CART
##
## 1200 samples
##   11 predictor
##    2 classes: 'bad', 'good'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.02760736  0.8753666  0.3646685
##   0.03374233  0.8770775  0.3681717
##   0.07975460  0.8706099  0.2649773
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03374233.
```

The optimal model that was selected seems to has good accuracy but has low Kappa. Kappa static adjusts accuracy by accounting the possibility of the agreement occurring by chance. This is important for this dataset as it has class imbalance. The good class appears far less frequently than bad class.

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

Figure 1: kappa statistic formula

At the formula Pr(a) refers to the actual agreement between the predictions and the actual values. Pr(e) refers to the expected agreement between the predictions and the actual values.

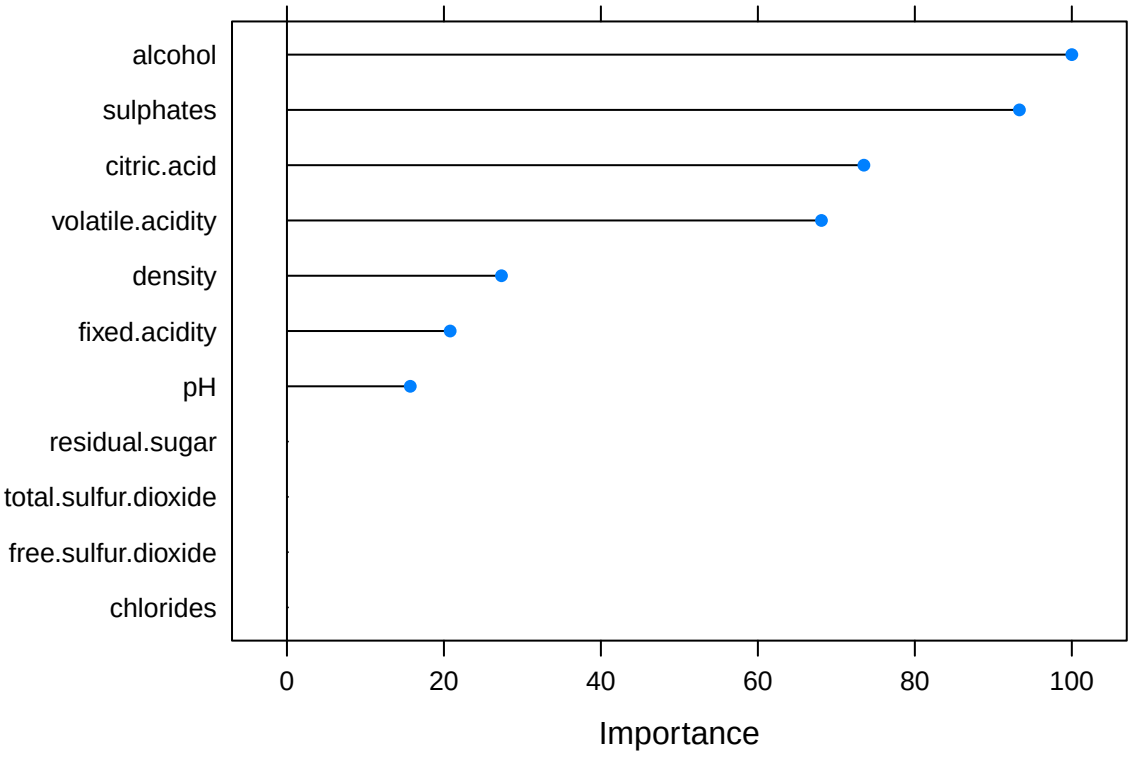
The calculation of variables by train of the model produced an interesting result. Although alcohol and sulphates are evaluated by the model as the important factors and as predicted by initial data analysis, citric.acid steals third place from volatile.acidity.

```
varImp(rpart_model)
```

```
## rpart variable importance
##
##                      Overall
## alcohol               100.00
## sulphates              93.32
## citric.acid            73.51
## volatile.acidity       68.10
## density                27.33
## fixed.acidity          20.80
## pH                     15.71
## free.sulfur.dioxide     0.00
## chlorides               0.00
```

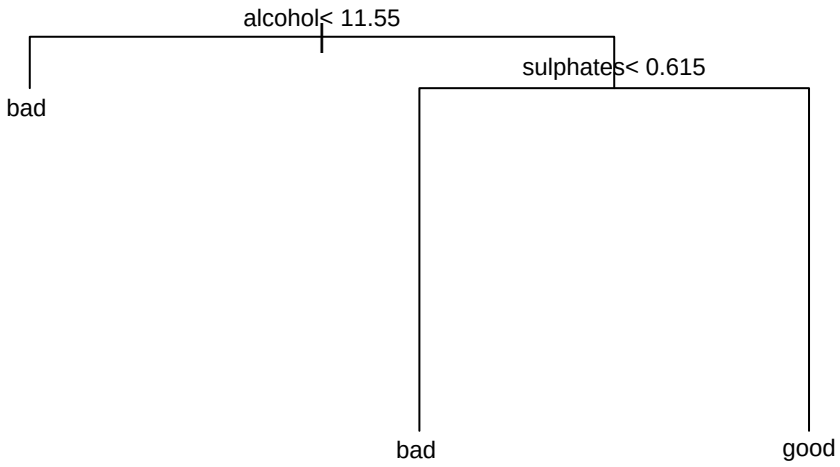
```
## total.sulfur.dioxide    0.00
## residual.sugar          0.00
```

```
plot(varImp(rpart_model))
```




It is time to evaluate the model's performance.

```
rpart_predictor<-predict(rpart_model, wine_test)

plot(rpart_model$finalModel, margin=0.1)
text(rpart_model$finalModel, cex=0.75)
```




```
confusionMatrix(rpart_predictor,wine_test$status)
```

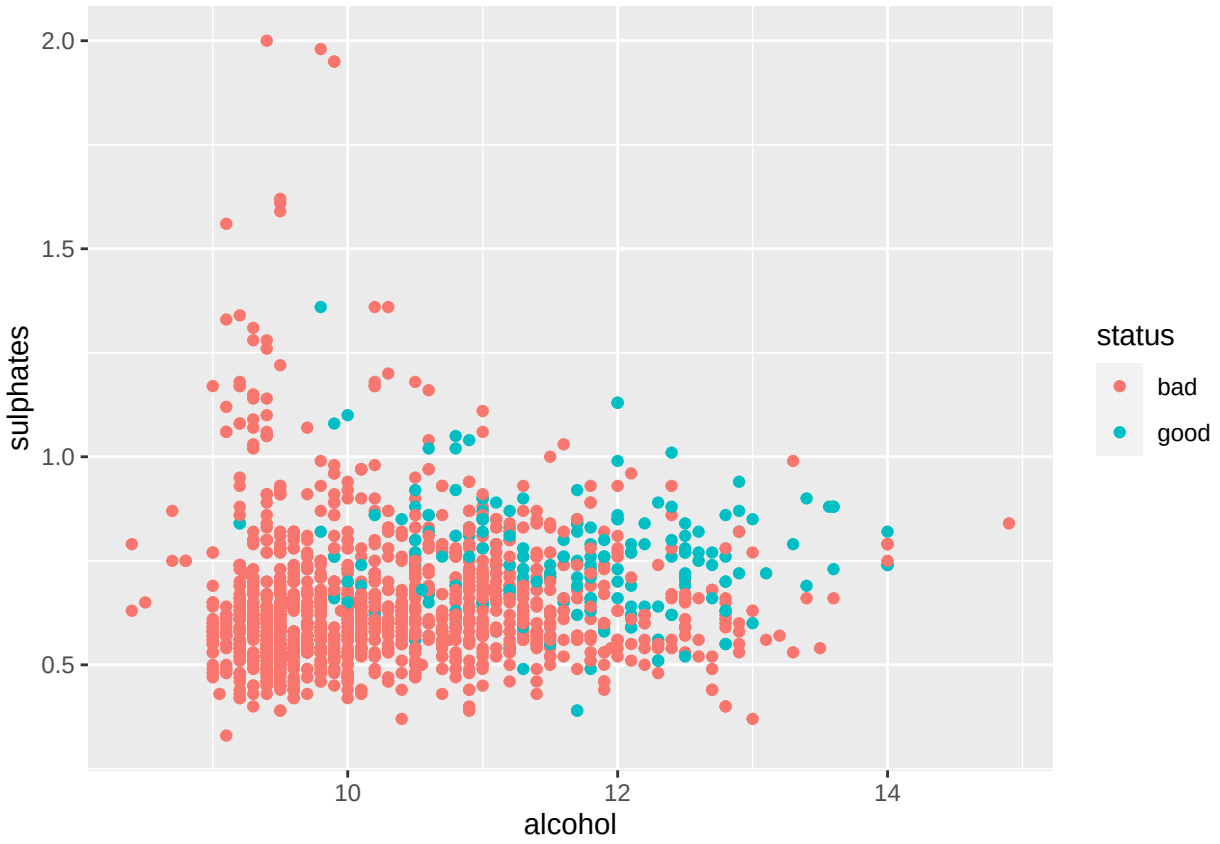
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  327   36
##       good  18   18
##
##                Accuracy : 0.8647
##                  95% CI : (0.8271, 0.8967)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.5362
##
##                   Kappa : 0.3272
##
##  Mcnemar's Test P-Value : 0.0207
##
##             Sensitivity : 0.9478
##             Specificity : 0.3333
##          Pos Pred Value : 0.9008
##          Neg Pred Value : 0.5000
##              Prevalence : 0.8647
##          Detection Rate : 0.8195
##    Detection Prevalence : 0.9098
```

```
##        Balanced Accuracy : 0.6406
##
##         'Positive' Class : bad
##
```

The evaluation shows that the model achieved high sensitivity but low specificity. Though it achieved a general good accuracy, the prediction of the good wine is far from adequate. Moreover it is observable from the tree plot that the model used only two variables. High percentage of alcohol seems to be very important for a good wine so it should not be abused.

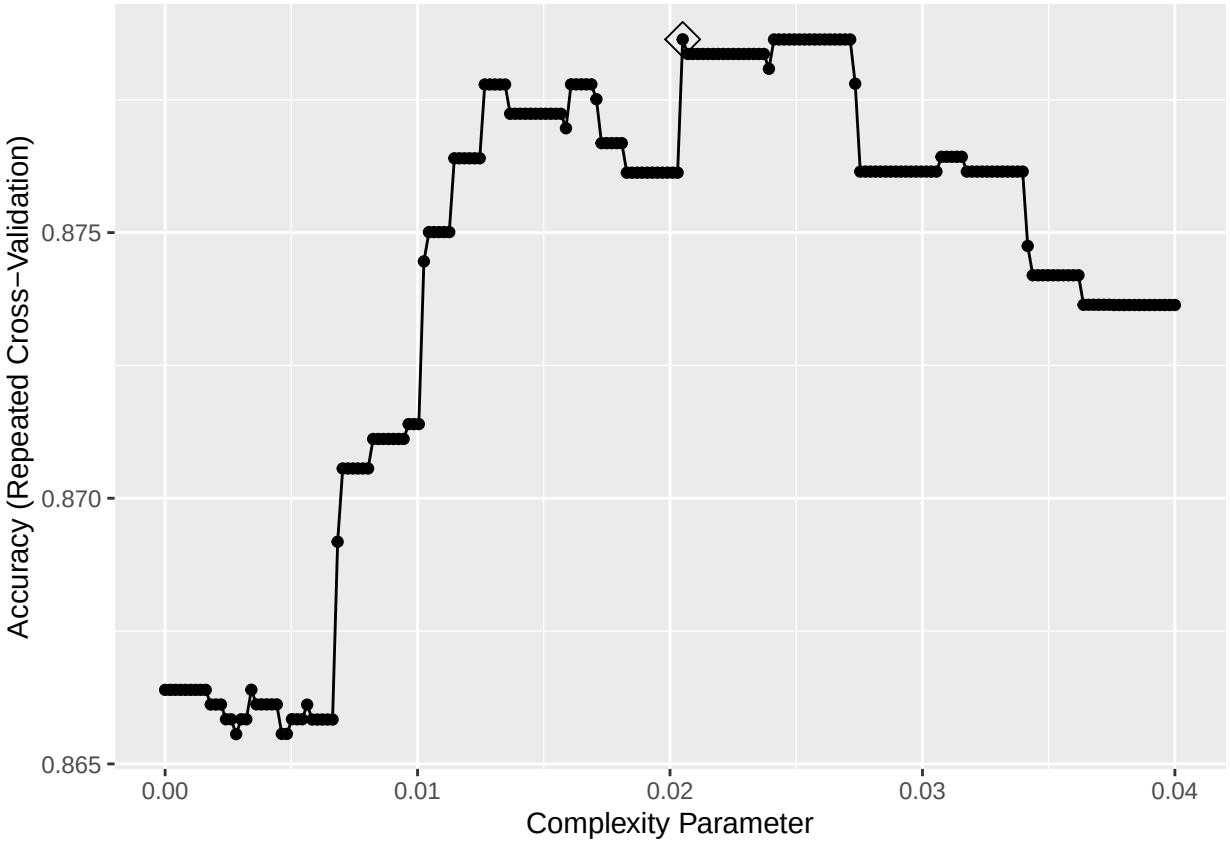
These two variables by themselves are not good enough to separate food from bad wine.

```
ggplot(wine, aes(alcohol, sulphates, color = status)) + geom_point()
```

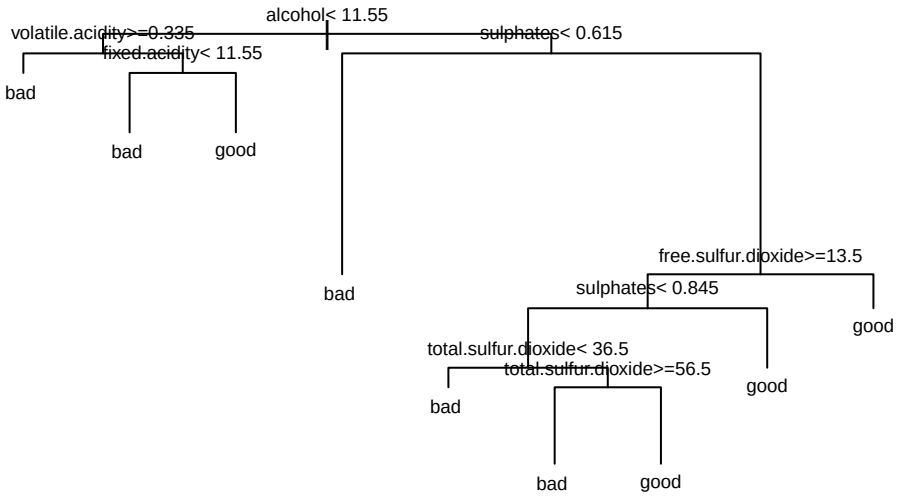



Maybe the model can perform better if I tune the cp parameter. I will use also repeated k-Fold cross-validation for the model. Repeated k-fold cross-validation will improve the estimated performance by repeating the cross-validation procedure multiple times and reporting the mean result across all folds from all runs.

```
tunegrider = data.frame(cp=seq(0,0.04, len=200))
train_control <- trainControl(method="repeatedcv", number=10, repeats=3)
rpart_model<-train(status~.,data=wine_train, method = "rpart",
  tuneGrid=tunegrider,trControl=train_control)

ggplot(rpart_model,highlight = TRUE)
```




```
plot(rpart_model$finalModel, margin=0.05)
text(rpart_model$finalModel, cex=0.55)
```




```
rpart_predictor<-predict(rpart_model, wine_test)

rpart_model$bestTune
```

```
##             cp
## 103 0.02050251
```

```
rpart_model$finalModel
```

```
## n= 1200
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1200 163 bad (0.86416667 0.13583333)
##     2) alcohol< 11.55 1012  78 bad (0.92292490 0.07707510)
##       4) volatile.acidity>=0.335 897  48 bad (0.94648829 0.05351171) *
##       5) volatile.acidity< 0.335 115  30 bad (0.73913043 0.26086957)
##        10) fixed.acidity< 11.55 100  19 bad (0.81000000 0.19000000) *
##        11) fixed.acidity>=11.55 15   4 good (0.26666667 0.73333333) *
##     3) alcohol>=11.55 188  85 bad (0.54787234 0.45212766)
##       6) sulphates< 0.615 62   9 bad (0.85483871 0.14516129) *
##       7) sulphates>=0.615 126  50 good (0.39682540 0.60317460)
##        14) free.sulfur.dioxide>=13.5 56  26 bad (0.53571429 0.46428571)
```

```
##          28) sulphates< 0.845 45  17 bad (0.62222222 0.37777778)
##            56) total.sulfur.dioxide< 36.5 8   0 bad (1.00000000 0.00000000) *
##            57) total.sulfur.dioxide>=36.5 37  17 bad (0.54054054 0.45945946)
##             114) total.sulfur.dioxide>=56.5 24   6 bad (0.75000000 0.25000000) *
##             115) total.sulfur.dioxide< 56.5 13   2 good (0.15384615 0.84615385) *
##          29) sulphates>=0.845 11   2 good (0.18181818 0.81818182) *
##        15) free.sulfur.dioxide< 13.5 70  20 good (0.28571429 0.71428571) *
```

```
confusionMatrix(rpart_predictor,wine_test$status)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  331   36
##       good  14   18
##
##                Accuracy : 0.8747
##                  95% CI : (0.8381, 0.9055)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.308924
##
##                   Kappa : 0.3535
##
##  Mcnemar's Test P-Value : 0.002979
##
##             Sensitivity : 0.9594
##             Specificity : 0.3333
##          Pos Pred Value : 0.9019
##          Neg Pred Value : 0.5625
##              Prevalence : 0.8647
##          Detection Rate : 0.8296
##    Detection Prevalence : 0.9198
##       Balanced Accuracy : 0.6464
##
##        'Positive' Class : bad
##
```

The model now used more predictors and achieved better results. Nevertheless the specificity is very low. We will try another algorithm.

## k-Nearest Neighbors Model

This time a K-nn algorithm will be used as it is simple and effective. I will tune the k parameter and use also repeated k-Fold cross-validation for the model.

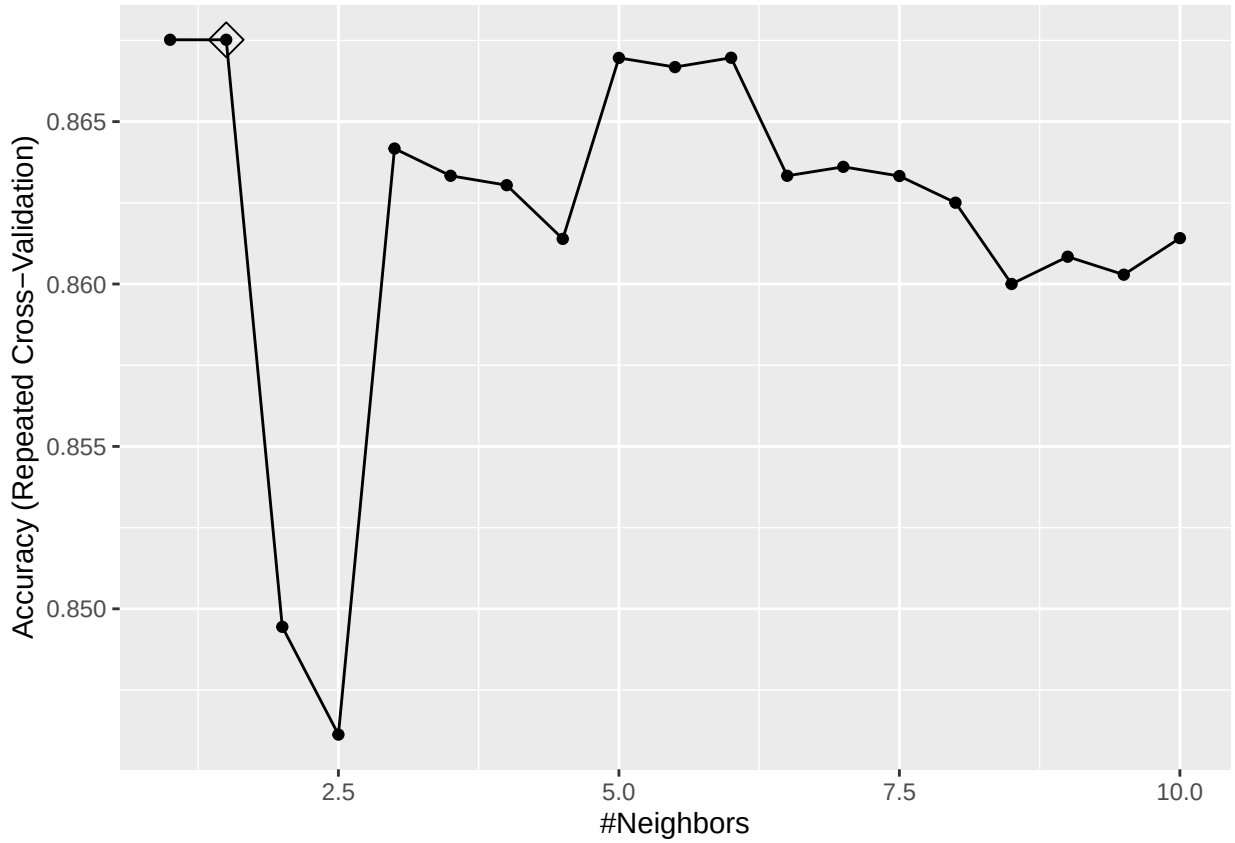
```
train_control <- trainControl(method="repeatedcv", number=10, repeats=3)
knn_model<-train(status~.,data=wine_train, method = "knn",trControl=train_control,
tuneGrid=data.frame(k=seq(1,10,0.5)))
knn_predictor<-predict(knn_model, wine_test)

knn_model$bestTune
```

```
##     k
## 2 1.5
```

```
knn_model$finalMode
```

```
## 1.5-nearest neighbor model
## Training set outcome distribution:
##
##  bad good
## 1037  163
```

```
ggplot(knn_model, highlight = TRUE)
```




```
confusionMatrix(knn_predictor,wine_test$status)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  320   30
##       good  25   24
##
##                Accuracy : 0.8622
##                  95% CI : (0.8244, 0.8944)
```

```
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.5934
##
##                   Kappa : 0.3871
##
##  Mcnemar's Test P-Value : 0.5896
##
##             Sensitivity : 0.9275
##             Specificity : 0.4444
##          Pos Pred Value : 0.9143
##          Neg Pred Value : 0.4898
##              Prevalence : 0.8647
##          Detection Rate : 0.8020
##    Detection Prevalence : 0.8772
##       Balanced Accuracy : 0.6860
##
##        'Positive' Class : bad
##
```

The model needs more tuning. Because K-nn is depended on measurement scale of the predictors, predictors that have greater ranges impact more. To eliminate this effect i will normalize the data.

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

wine_n <- as.data.frame(lapply(wine[1:11], normalize))


wine_n$status<-wine$status


summary(wine_n)
```

```
##  fixed.acidity    volatile.acidity  citric.acid    residual.sugar
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.000   Min.   :0.00000
##  1st Qu.:0.2212   1st Qu.:0.1849   1st Qu.:0.090   1st Qu.:0.06849
##  Median :0.2920   Median :0.2740   Median :0.260   Median :0.08904
##  Mean   :0.3292   Mean   :0.2793   Mean   :0.271   Mean   :0.11225
##  3rd Qu.:0.4071   3rd Qu.:0.3562   3rd Qu.:0.420   3rd Qu.:0.11644
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.000   Max.   :1.00000
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :0.00000   Min.   :0.00000     Min.   :0.00000      Min.   :0.0000
##  1st Qu.:0.09683   1st Qu.:0.08451     1st Qu.:0.05654      1st Qu.:0.4060
##  Median :0.11185   Median :0.18310     Median :0.11307      Median :0.4905
##  Mean   :0.12599   Mean   :0.20951     Mean   :0.14300      Mean   :0.4902
##  3rd Qu.:0.13022   3rd Qu.:0.28169     3rd Qu.:0.19788      3rd Qu.:0.5701
##  Max.   :1.00000   Max.   :1.00000     Max.   :1.00000      Max.   :1.0000
##        pH           sulphates         alcohol        status
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   bad :1382
##  1st Qu.:0.3701   1st Qu.:0.1317   1st Qu.:0.1692   good: 217
##  Median :0.4488   Median :0.1737   Median :0.2769
##  Mean   :0.4497   Mean   :0.1965   Mean   :0.3112
```

```
##  3rd Qu.:0.5197   3rd Qu.:0.2395   3rd Qu.:0.4154
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
```

Now that the data are normalized a new train and test dataset are created to evalute the k-nn model.

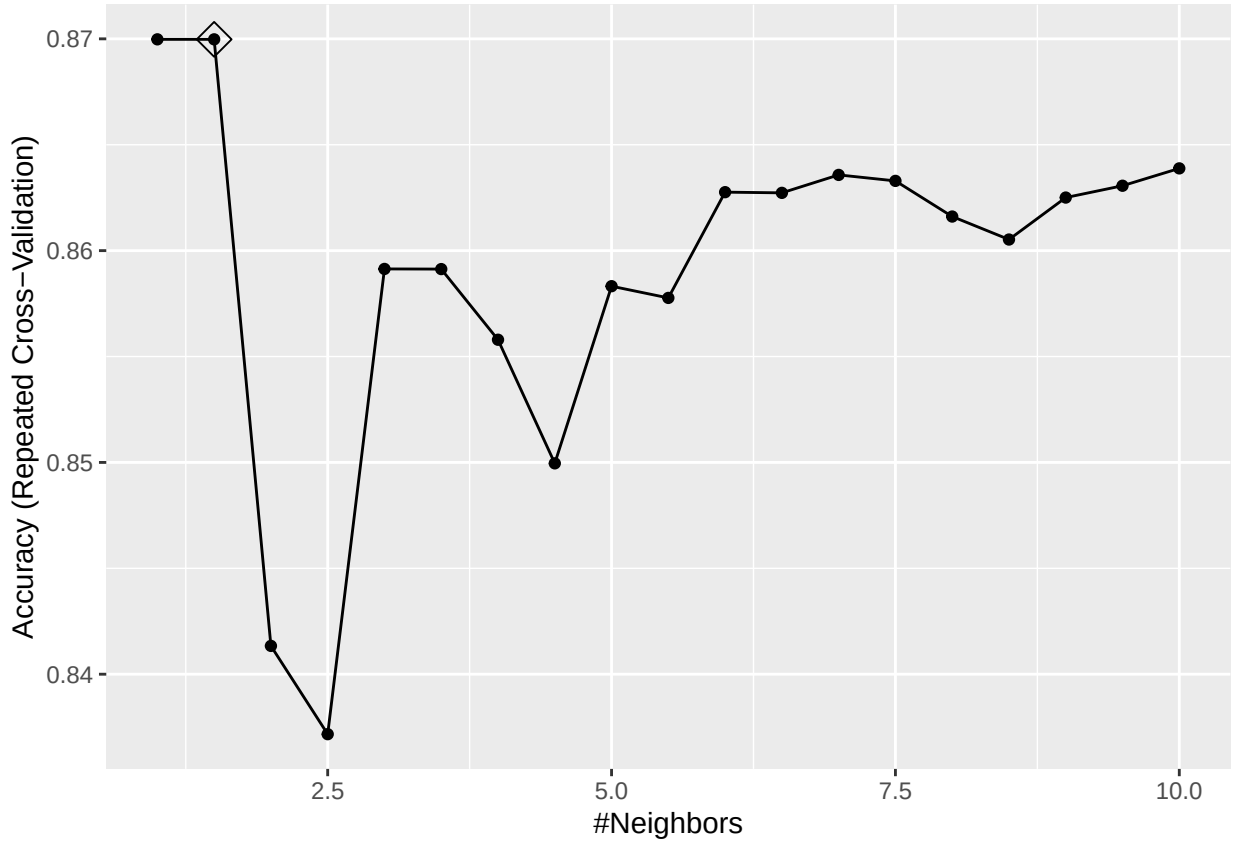
```
test_index_n <- createDataPartition(wine$status, times = 1, p = 0.75, list = FALSE)

wine_train_n <-wine[test_index_n  ,]

wine_test_n <-wine[-test_index_n  ,]


train_control <- trainControl(method="repeatedcv", number=10, repeats=3)
knn_model<-train(status~.,data=wine_train_n, method = "knn",
trControl=train_control, tuneGrid=data.frame(k=seq(1,10,0.5)))
knn_predictor<-predict(knn_model, wine_test_n)

knn_model$bestTune
```

```
##     k
## 2 1.5
```

```
knn_model$finalMode
```

```
## 1.5-nearest neighbor model
## Training set outcome distribution:
##
##  bad good
## 1037  163
```

```
ggplot(knn_model, highlight = TRUE)
```




```
confusionMatrix(knn_predictor,wine_test_n$status)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  311   25
##       good  34   29
##
##                Accuracy : 0.8521
##                  95% CI : (0.8134, 0.8855)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.7914
##
##                   Kappa : 0.4097
##
##  Mcnemar's Test P-Value : 0.2976
##
##             Sensitivity : 0.9014
##             Specificity : 0.5370
##          Pos Pred Value : 0.9256
##          Neg Pred Value : 0.4603
##              Prevalence : 0.8647
##          Detection Rate : 0.7794
##    Detection Prevalence : 0.8421
```

```
##       Balanced Accuracy : 0.7192
##
##        'Positive' Class : bad
##
```

The result is a good accuracy combined with the best specificity and balanced accuracy so far. Separating good from average/bad wines is not an easy job as it seems.

## Random Forest Model

Finally, i will use Random Forest as it is a good all-purpose model, that performs very good on most of the problems. I will tune the model using ntree, mtry and metric parameter. Ntree is the number of trees to grow and because the problem appears a little difficult , a value more than default of 500 will be used. Kappa will be used as training metric as there is a problem with low specificity and Kappa in the models's results till now. Mry parameter is the number of variables randomly sampled as candidates at each split.

```
rfTunegrider = data.frame(mtry=seq(2,6, 0.5))
train_control <- trainControl(method="repeatedcv", number=10, repeats=3)
random_model<-train(status~.,data=wine_train, method = "rf",
metric="Kappa", tuneGrid=rfTunegrider,trControl=train_control,ntree = 1000)

random_predictor<-predict(random_model, wine_test)
result<-confusionMatrix(random_predictor,wine_test$status)

result
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction bad good
##       bad  336   30
##       good   9   24
##
##                Accuracy : 0.9023
##                  95% CI : (0.8688, 0.9296)
##     No Information Rate : 0.8647
##     P-Value [Acc > NIR] : 0.014108
##
##                   Kappa : 0.5004
##
##  Mcnemar's Test P-Value : 0.001362
##
##             Sensitivity : 0.9739
##             Specificity : 0.4444
##          Pos Pred Value : 0.9180
##          Neg Pred Value : 0.7273
##              Prevalence : 0.8647
##          Detection Rate : 0.8421
##    Detection Prevalence : 0.9173
##       Balanced Accuracy : 0.7092
##
##        'Positive' Class : bad
##
```

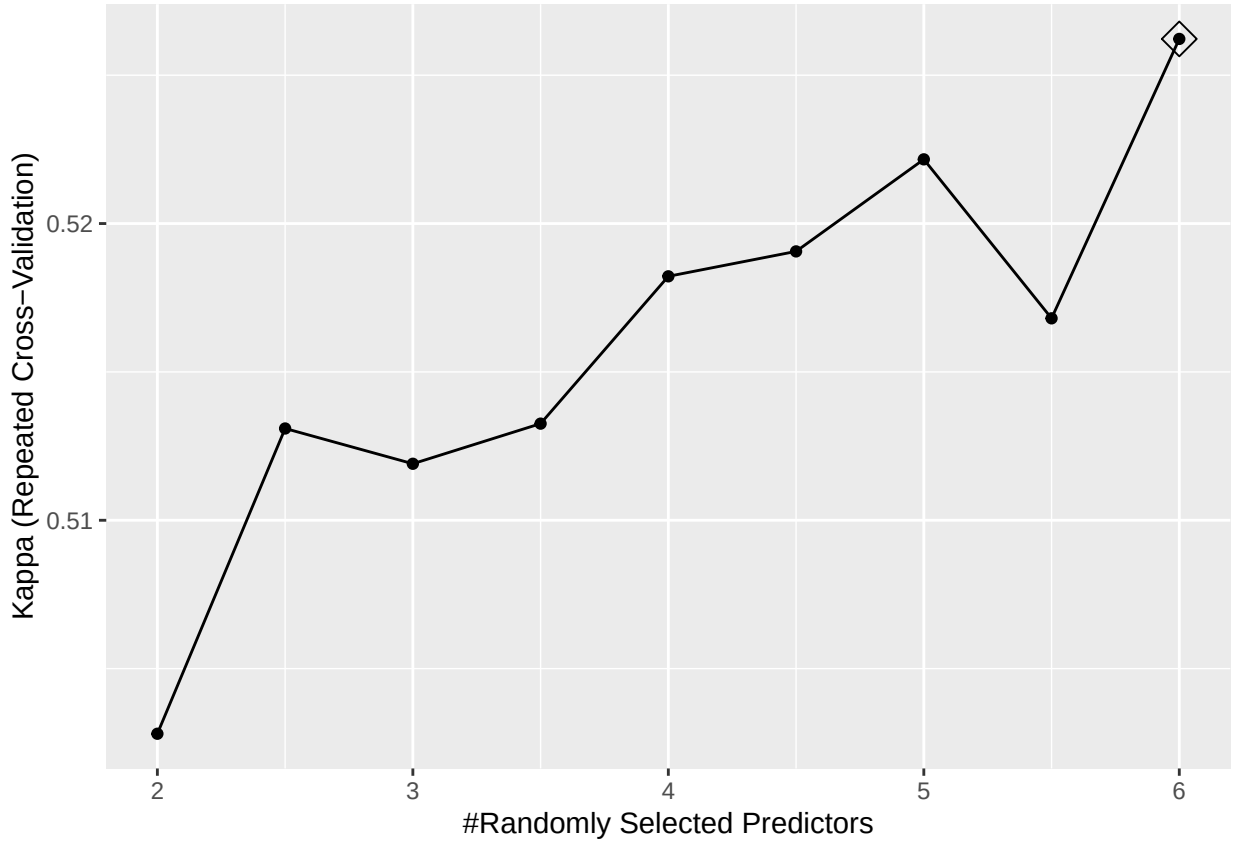
```
random_model$bestTune
```

```
##   mtry
## 9    6
```

```
random_model$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 1000, mtry = min(param$mtry,      ncol(x)))
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 8.92%
## Confusion matrix:
##       bad good class.error
## bad  1008   29  0.02796528
## good   78   85  0.47852761
```

```
ggplot(random_model,highlight = TRUE)
```




Random forest achieved the best accuracy of all the models in this analysis combined with relatively satisfactory specificity and a good balanced accuracy. As i sought a balance between precision and recall a high f-score suggests success of the model.

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

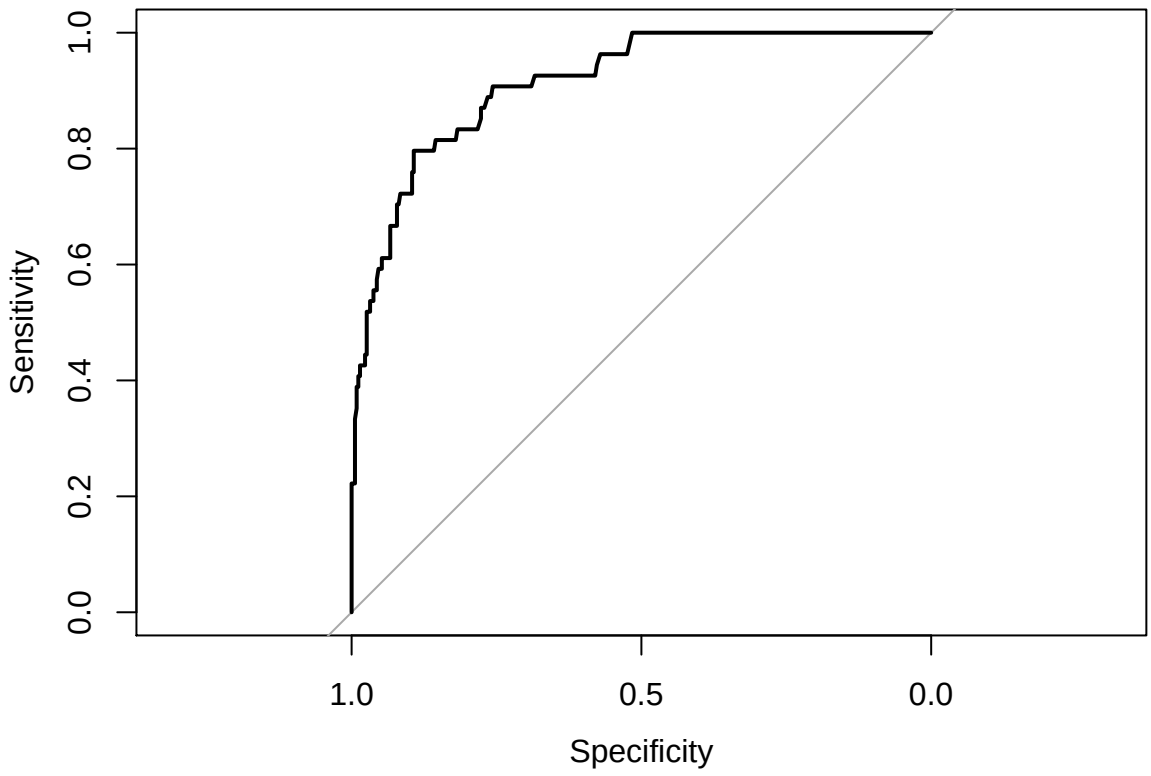
Figure 2: f-score formula

```
result$byClass["F1"]
```

```
##        F1
## 0.9451477
```

A ROC curve helps visualize the trade off between sensitivity and specificity at various threshold settings and better evaluate the model.

```
evalResult.rf <- predict(random_model, wine_test, type = "prob")


random_roc<-roc(wine_test$status,evalResult.rf[,2])

plot(random_roc)
```




AUC (Area Under Curve ) that measures the entire two-dimensional area underneath the entire ROC curve suggests that the classifier has excellent predictive value.

```
auc(random_roc)
```

```
## Area under the curve: 0.9139
```

# Results

The best candidate for separating good wines for the rest was the wine quality of 7. After research, it was found that the best predictors for wine quality is alcohol, sulphates, and volatile acidity. Three models were built to predict wine quality, a regression tree model, a k-nearest neighbours' model and a random forest model.

Regression tree model achieved an accuracy of 0.8747 with a sensitivity of 0.9594 but also with a low specificity of 0.3333 and Kappa of 0.3535.

A fine tuned k-nearest neighbours' model attained an accuracy of 0.8521, and a more balanced sensitivity of 0.9014 and specificity of 0.5370. The kappa value was 0.4097.

Ultimately, a random forest model was tuned and achieved an accuracy of 0.9023, sensitivity of 0.9739 and specificity of 0.4444. Kappa value reached a decent value of 0.5004. Furthermore, test accuracy was calculated with an f1-score of 0.9451477.

# Conclusion

The best wine is subjective, it is the one that makes you happy. According to this analysis though, the factors that makes a wine stand out is the high alcohol content (more that 11.5%) and the influence of its sulphates, volatile acidity and citric acid.

Its relatively difficult though to separate the finest wine from the rest, as wines quality follows a normal distribution and most of them are average. Regression tree model was inadequate especially in founding the finest of the wines suffering from low specificity.

The k-nearest neighbors algorithm based model, especially its tuned version solved this problem, but without achieving excellent accuracy. Random forest model with an accuracy of 0.9023 and f1-score of 0.9451477 ensures that next time we will be able to distinguish the good Portugal red wine.