# HarvardX PH125.9X Data Science 'CYO' Capstone Project on Wine Quality

*David Wong*

*June, 2020*

## Contents

# Section 1: Introduction

## 1.1 Background

This is the second of the two Capstone projects that learners need to complete in the final module (PH125.9X) of the HarvardX Data Science Professional Certification series.

Learners are allowed to pick a subject and data of their choice and develop a machine learning product using the techniques learned throughout the series and following the guidelines provided by the course provider.

The data chosen for this project is the "Wine Quality Data Set" provided by the UCI Machine Learning Repository in the following link:

https://archive.ics.uci.edu/ml/datasets/Wine+Quality.

## 1.2 About the data set

There are two data files from the source, one for the red wines and the other for the white wines.

Both data files have the same format and same variables.

There are 11 input variables based on physicochemical tests that determine the properties of the wine: 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

There is 1 output variable based on sensory data: 12 - quality (score between 0 (the worst) and 10 (the best))

## 1.3 Objectives

The key objectives of the project are:

- explore different machine learning algorthms and techniques particularly those related to classification or regression tasks;
- choose the ones that are relevant and suitable to the data and scenario at hand;
- build and test the models chosen;
- evaluate the performances of the models based on how accurate the predictions on the wine quality are; and
- identify the "best" model that achieves the highest "overall accuracy".

## 1.4 Key steps

The key steps throughout are:

- Data Preparation - where data files will be downloaded from the source.
- Data Exploration - where a combination of simple scripts, plots and visualisation will be used to explore the data behaviour and provide facts and statistics to support the rationale used behind model building.
- Model Building - where the original data set will be split into training set and test set for model building and testing respectively; training data set will be used together with the functions provided by the "Caret" package for training and cross-validations. Three models will be used: k-nearest neighbours (KNN), random forest (RF) and support vector machine (SVM).

- Model Testing - where models trained/built will be tested using the test data set; the performances of the models will be evaluated based on the "overall accuracy" of the predictions on wine quality.

Note: The same steps will be applied twice on both the red wine data set and the white wine data set respectively.

## 1.5 R script

The code snipplets presented in this report are extracted partially from the R script submitted separately as part of the project submission.

Consolidation of these code snipplets does not make it whole, therefore, please always refer to the R script for a successful full run.

A word of caution: the whole script usually takes around 90 minutes to complete running, but it may take longer (up to 2 hours 30 minutes) depending on the resource availability during execution and the overall hardware configuration of the machine. The following is the specification of the machine used for reference:

- CPU: 1.6 GHz Dual-Core Intel Corei5
- Memory: 16GB 2133 MHz LPDDR3

# Section 2: Methods / Analysis

## 2.1 Data Preparation

The first step is to download both the csv files for red wine data and white wine data from the UCI website and store them in the respective dataframes.

```
# Download Red Wine data file from UCI and load into data frame
redwine_url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.
redwine_raw <- read.csv(redwine_url, header = TRUE, sep = ";")
redwine <- redwine_raw

# Download White Wine data file from UCI and load into data frame
whitewine_url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-wh
whitewine_raw <- read.csv(whitewine_url, header = TRUE, sep = ";")
whitewine <- whitewine_raw
```

## 2.2 Data Exploration

### 2.2.1 General data inspection

According to UCI on their website, there are 11 input variables with numeric data type and 1 output variable in integer in both the red wine file and the white wine file; there should be 1599 rows for the red wine file and 4898 rows for the white wine file.

The following codes confirm that the data frames are loaded completely as the summary data matches with the information provided by UCI.

```
# Confirm summary data matches with information from UCI website
str(redwine)
```

```
## '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 ...
```

```
str(whitewine)
```

```
## 'data.frame':    4898 obs. of  12 variables:
##  $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
##  $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
##  $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
##  $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
```

```
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...
##  $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
##  $ total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
##  $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
##  $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
##  $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

The data looks generally tidy after visual inspection with the codes below.

```
# Visually insepct the data
head(redwine)
```

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

```
head(whitewine)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.0             0.27        0.36           20.7     0.045
## 2           6.3             0.30        0.34            1.6     0.049
## 3           8.1             0.28        0.40            6.9     0.050
## 4           7.2             0.23        0.32            8.5     0.058
## 5           7.2             0.23        0.32            8.5     0.058
## 6           8.1             0.28        0.40            6.9     0.050
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  45                  170  1.0010 3.00      0.45     8.8
## 2                  14                  132  0.9940 3.30      0.49     9.5
## 3                  30                   97  0.9951 3.26      0.44    10.1
## 4                  47                  186  0.9956 3.19      0.40     9.9
## 5                  47                  186  0.9956 3.19      0.40     9.9
## 6                  30                   97  0.9951 3.26      0.44    10.1
##   quality
```

```
## 1       6
## 2       6
## 3       6
## 4       6
## 5       6
## 6       6
```

It's confirmed that there is no "NA" in the data with the codes below and hence the data does not require further cleansing or transformation.

```
# Check for "NA"s
sum(is.na(redwine))
```

```
## [1] 0
```

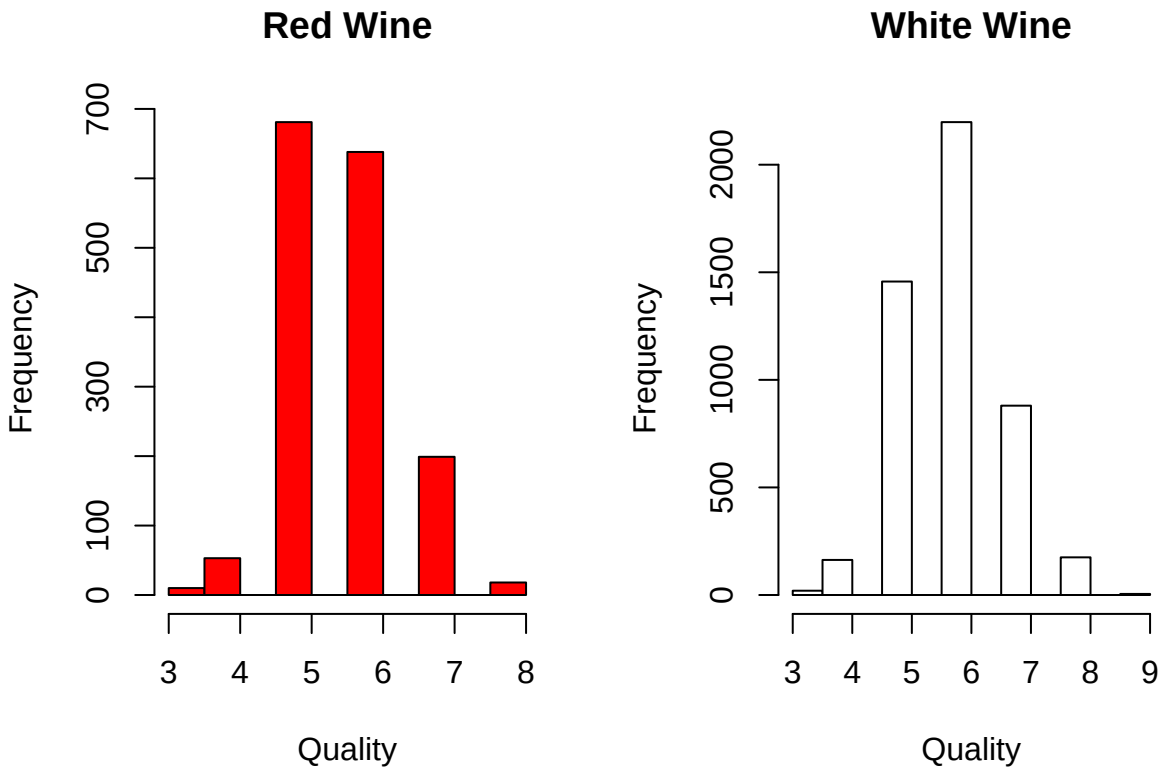
```
sum(is.na(whitewine))
```

```
## [1] 0
```

### 2.2.2 Class Imbalance

The histograms and the tables below show that there is a large class imbalance, there seems to be a lot more "average" wines (with quality graded with 5 and 6) than the "bad" wines (with quality value below 5) or the "good wines" (with quality value above 7).

```
# Analyse distribution of quality values using histogram
par(mfrow=c(1,2))
hist(redwine$quality, main="Red Wine", xlab = "Quality", col = "red")
hist(whitewine$quality, main="White Wine", xlab = "Quality", col = "white")
```




```
par(mfrow=c(1,1))
```

The tables below give you the specific counts that further confirm the class imbalance. It is particularly obvious for the case of white wines where there are only 5 out of 4898 having a quality rating of 9.

```
# Analyse distribution of quality values using table to show the actual figures
table(redwine$quality)
```

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

```r
table(whitewine$quality)
```

```
##
##    3    4    5    6    7    8    9
##   20  163 1457 2198  880  175    5
```

### 2.2.3 Visualise quality vs each input variable

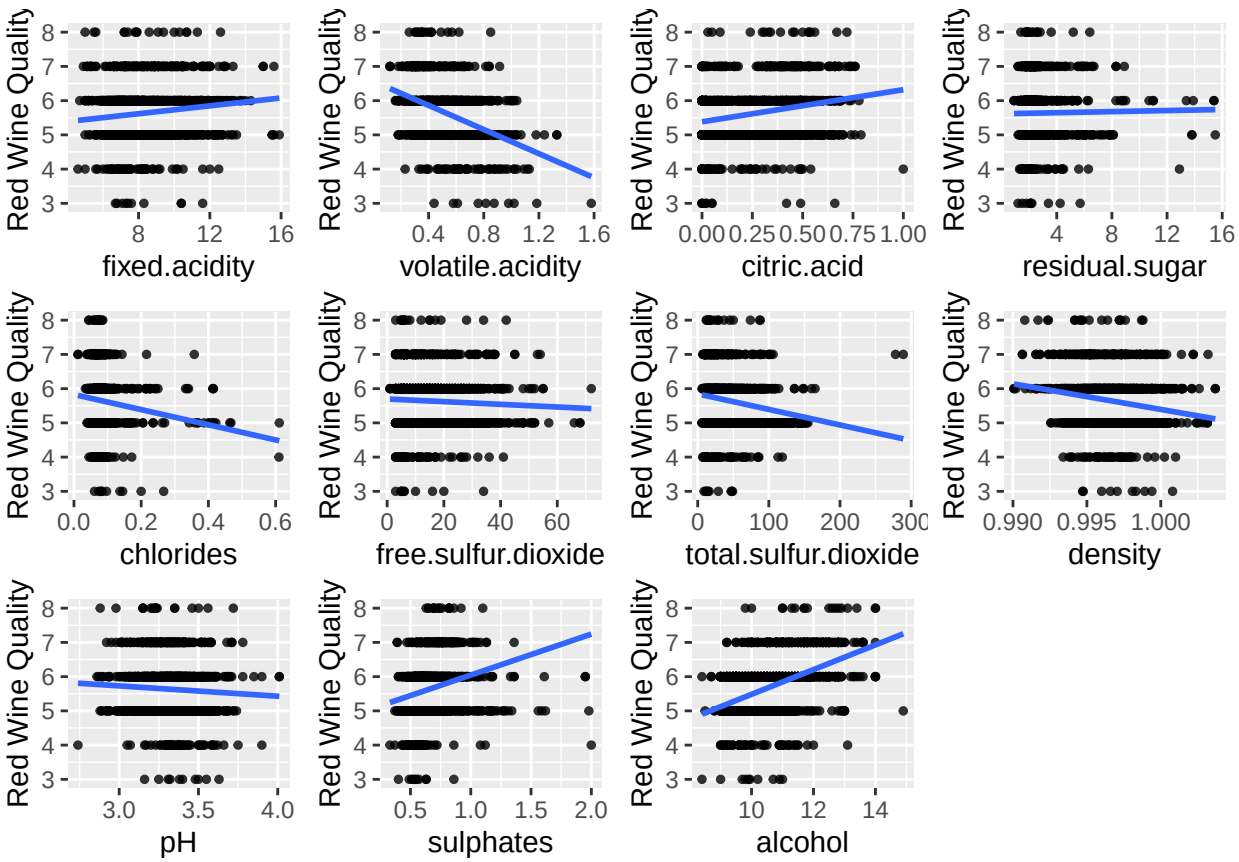
The following two sets of plots plot quality against each of the 11 input variables in the respective red and white wine data sets. The line on each plot shows the linear regression of quality as a function of the respective input variable.

Let's start with the Red Wine plots.

Judging the slope of the lines, volatile.acidity, sulphates, and alcohol have the strongest relationship with quality; whereas the lines for Free.sulfur.dioxide and residual.sugar are almost flat which imply that they have the weakest relationships with quality.

Through visual inspection, some of the input variables appear to have a few outliers.

```
# Plot Quality against each of the 11 input variables
# On Red Wine
redwineplots <- list()  # Create a list to store plots
for (i in 1:11) {
  p1 <- eval(substitute(
    ggplot(data=redwine,aes(x=redwine[ ,i], y=redwine[,"quality"])) +
      geom_point(alpha = 0.8, size =1) +
      geom_smooth(method = "lm", se = FALSE, size = 1) +
      labs(y="Red Wine Quality", x=names(redwine)[i])
    ,list(i = i)))
  redwineplots[[i]] <- p1  # add each plot into plot list
}
# Present plots
do.call(grid.arrange, c(redwineplots, ncol = 4))
```

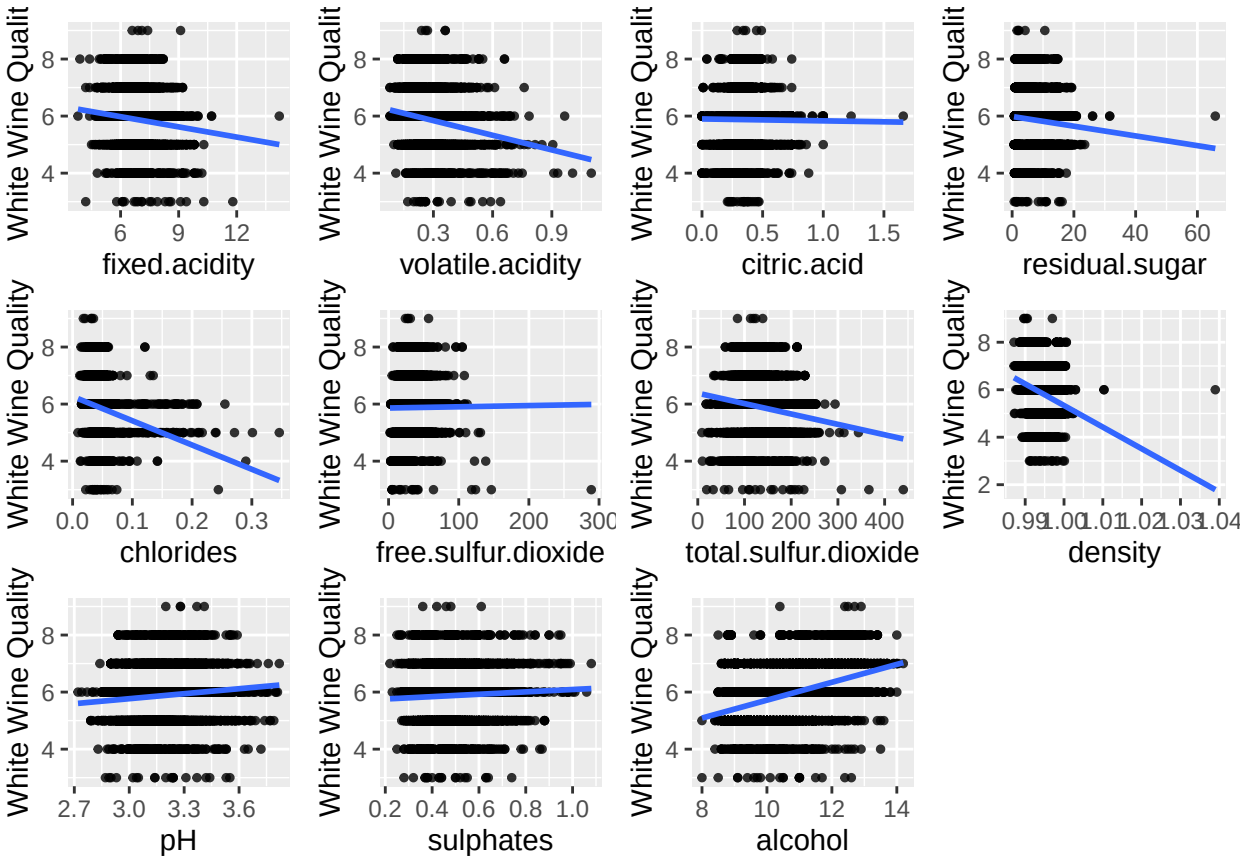


Moving on with the White Wine plots below.

Alcohol and density seem to have the strongest relationships with quality; whereas citric.acid, free.sulfur.dioxide, and sulphates appear to have the weakest relationships with quality.

There is a presence of outliers in most of the input variables, particularly obvious with residual.sugar and free.sulphur.dioxide.

```
# On White Wine
whitewineplots <- list()  # Create a list to store plots
for (i in 1:11) {
  p1 <- eval(substitute(
    ggplot(data=whitewine,aes(x=whitewine[ ,i], y=whitewine[,"quality"])) +
      geom_point(alpha = 0.8, size =1) +
      geom_smooth(method = "lm", se = FALSE, size = 1) +
      labs(y="White Wine Quality", x=names(whitewine)[i])
    ,list(i = i)))
  whitewineplots[[i]] <- p1  # add each plot into plot list
}
# Present plots
do.call(grid.arrange, c(whitewineplots, ncol = 4))
```



### 2.2.4 Correlation Matrix

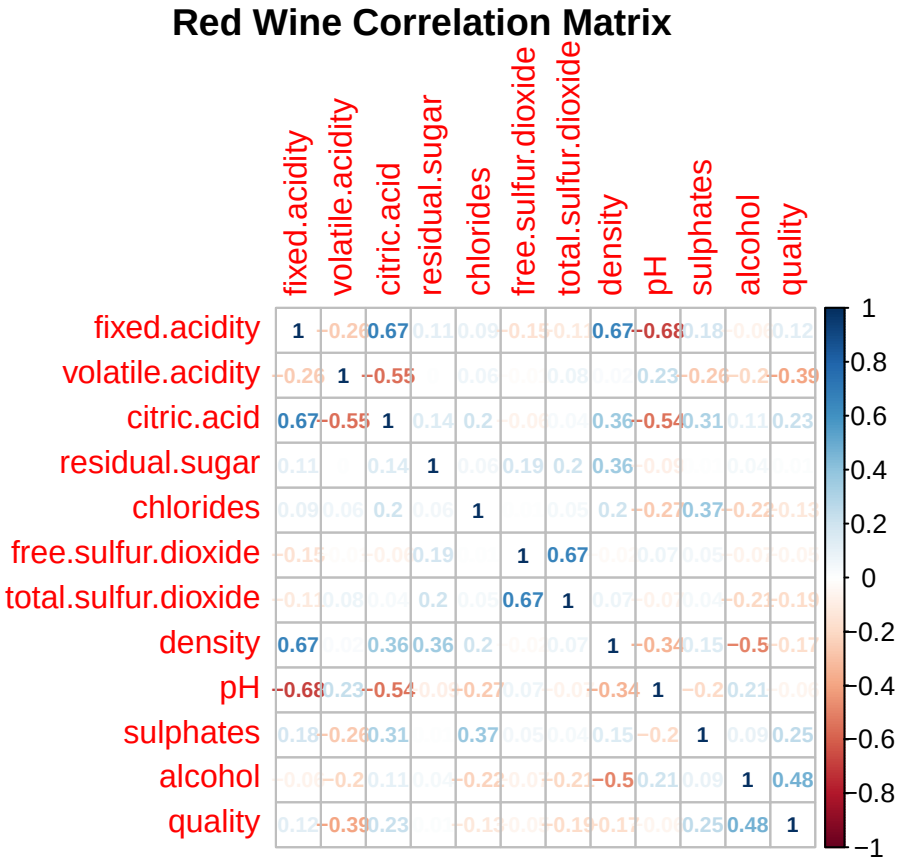
The following two plots show the correlation matrices of red wine and white wine data sets.

We are particularly interested in the relationships with quality.

As far as red wine data is concerned, only alcohol shows obvious relationship with quality even though the value is not particularly high, only at 0.48.

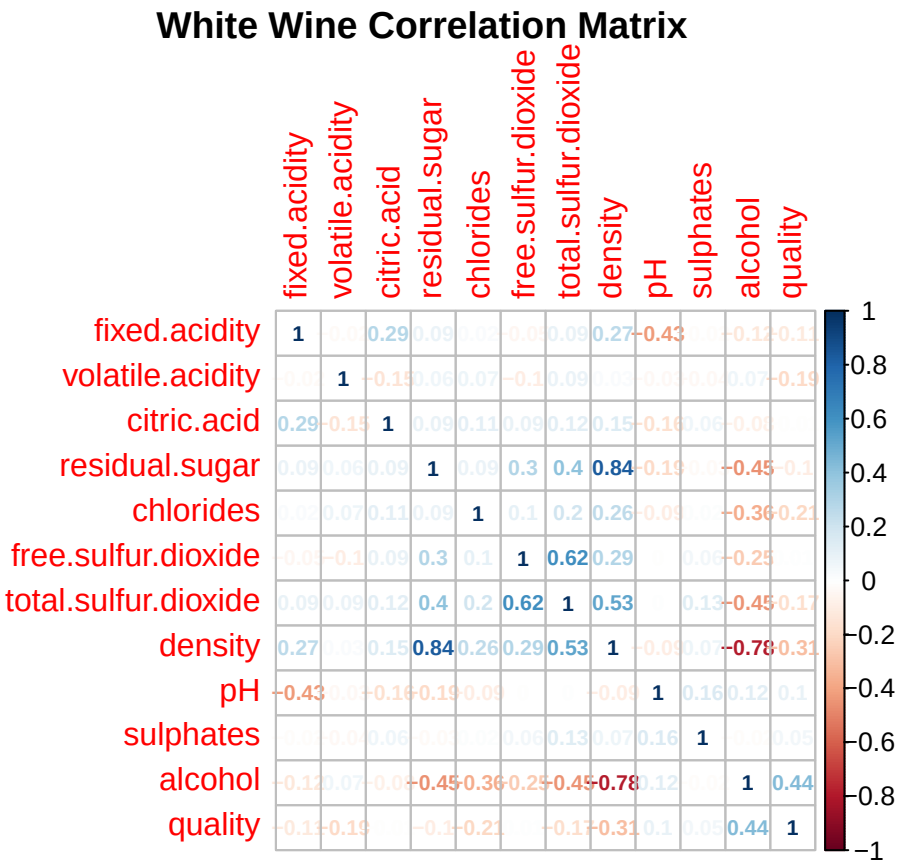
Other correlations that show strong values can be easily understood, for example, pH value is negatively correlated with fixed.acidity as the more acidic the wine is, the lower the pH value. Correlations as such are not of a concern.

```
# Plot correlation matrix using number method to display correlation coefficient
cor_redwine <- cor(redwine)
corrplot(cor_redwine, number.cex=0.7, method = 'number', title="Red Wine Correlation Matrix", mar=c(0,0
```




White wine data shows something slightly different. Density has a 0.84 correlation with residual.sugar and a -0.8 vs alcohol whereas alcohol has the stronger positive correlation value of 0.44 with quality. This could be a concern if we were planning to use linear models.

```
cor_whitewine <- cor(whitewine)
corrplot(cor_whitewine, number.cex=0.7, method = 'number', title = "White Wine Correlation Matrix", mar
```




Based on the findings from the above, non-linear classification models will be more appropriate than regression.

As we intend to build the same models for both red and white wine data, and there is not much commonality between both data sets in terms of the input variables that have weak correlations with quality, we therefore do not perform any feature selection by removing any of those.

For the sake of completeness and potential future comparison against the UCI study, we do not remove the outliers, either. Instead, we will use models that are more robust against the outliers such as Random Forest; and for the case of KNN, we will use standardisation to have the features transformed such that they all have a common range. Furthermore, as the dataset has a large number of samples, the remaining outliers will not affect the mean and SD to a significant extent such that they need to be removed.

In short, we are going to use k-nearest neighbours (KNN), random forest (RF) and support vector machine (SVM) for model building.

## 2.3 Model Building

Caret package and its related functions will be used for the sake of simplicity and convenience. For example, by using the tuneGrid argument, we can pass a grid of the hyperparameters we want to use into the train function; the expand.grid function simplies the creation by combining the hyperparameter values into every possible combination.

To achieve better accuracy, we are sticking with the rule of thumb of splitting our data samples into 80% training data as majority and 20% test data; in addition to that, we are going to use repeated cross-validation 5 times using 5 folds. For KNN, we will use the preProcess argument to centre and scale the input variations as part of the standardisation task.

### 2.3.1 Model Building on Red Wine data set

The following codes split the red wine data into training and test set in the ratio of 80:20.

```
# Prepare training and test data sets
redwine$quality <- as.factor(redwine$quality)
set.seed(2006, sample.kind="Rounding") #Use Year - Month - day of today's date, 2020-06-08, the first d
train_index_redwine <- createDataPartition(redwine$quality, p = 0.8, list = F)
train_set_redwine <- redwine[train_index_redwine,]
test_set_redwine <- redwine[-train_index_redwine,]
```

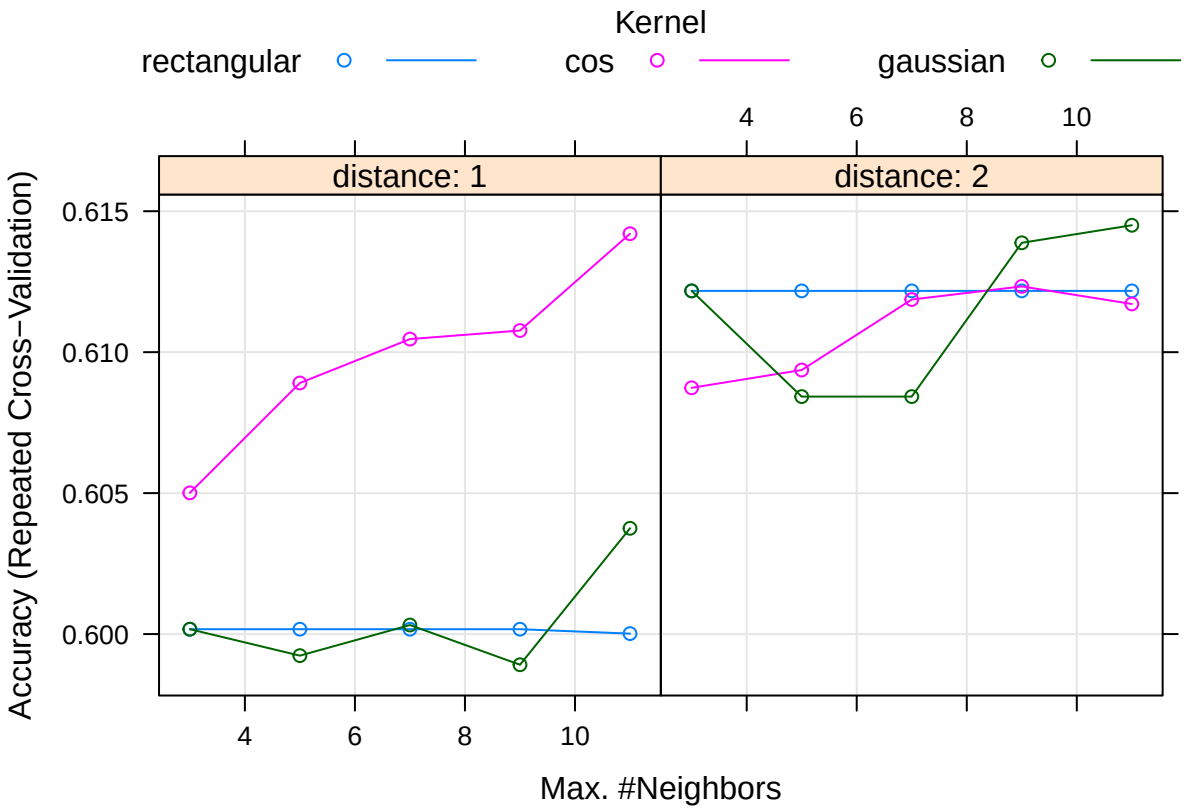
#### 2.3.1.1 k-nearest neighbours on Red Wine data

5 kmax, 2 distances (1 for Manahattan and 2 for Euclidian) and 3 kernel values will be used.

```
# Model 1 - k-nearest neighbours on Red Wine
# 5 kmax, 2 distances, 3 kernels

# Create a log to record execution time
run_log <- tibble(Activity = "KNN Red Start:", Time = Sys.time())

control <- trainControl(method = "repeatedcv", number = 5, repeats = 5)
grid_kknn <- expand.grid(kmax = c(3, 5, 7, 9, 11), distance = c(1, 2),
                         kernel = c("rectangular", "cos", "gaussian"))
train_kknn_redwine <- train(quality ~ ., data = train_set_redwine, method = "kknn",
                    trControl = control, tuneGrid = grid_kknn,
                    preProcess = c("center", "scale"))
plot(train_kknn_redwine)
```




```
train_kknn_redwine$bestTune
```

```
##    kmax distance   kernel
## 30   11        2 gaussian
```

```
# Record execution end-time
run_log <- bind_rows(run_log,tibble(Activity = "KNN Red End:", Time = Sys.time()))
```

As per result above, the gaussian kernel using Euclidian distance and k value of 11 works best for Red Wine data.

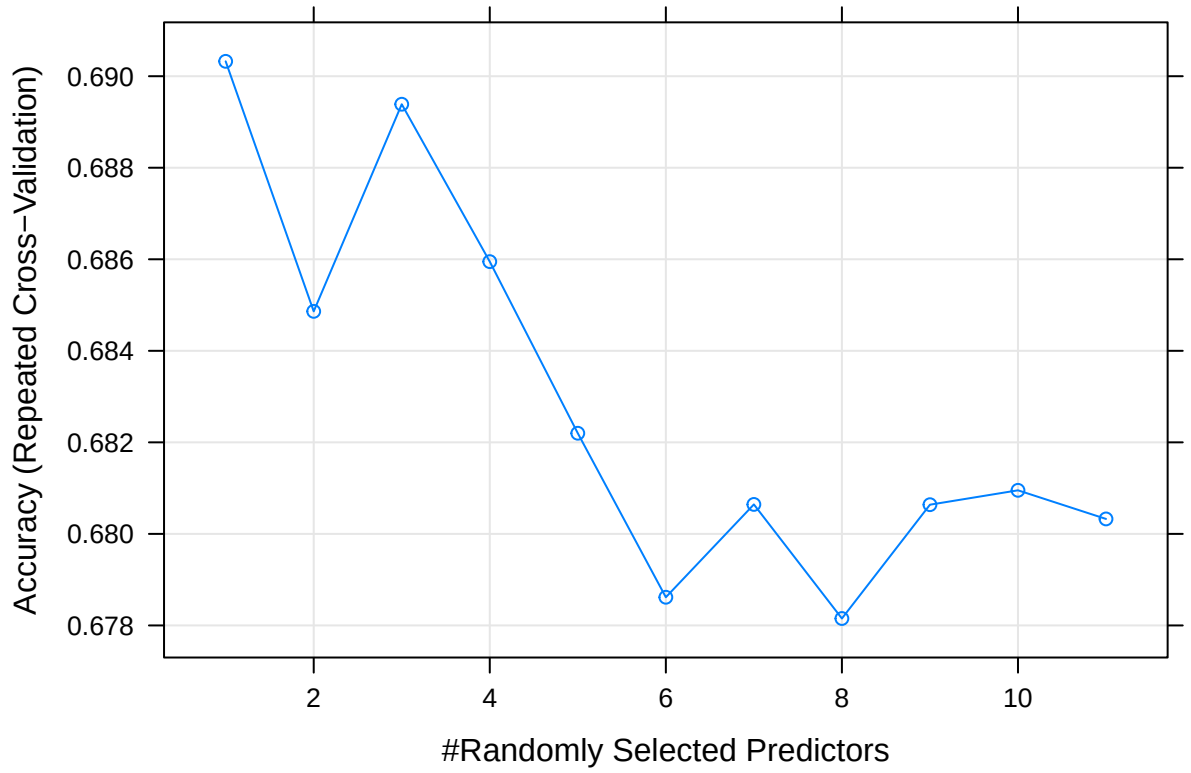
#### 2.3.1.2 Random Forest on Red Wine data

Mtry is the number of variables randomly sampled as candidates at each split and it is the only hyperparameter available for tuning. The values from 1 to 11 are used in the tuneGrid argument.

```
# Model 2 - Random Forest on Red Wine
# mtry from 1 through 11

# Record execution start-time
run_log <- bind_rows(run_log,tibble(Activity = "RF Red Start:", Time = Sys.time()))

control <- trainControl(method = "repeatedcv", number = 5, repeats = 5)
grid_rf <- expand.grid(mtry = 1:11)
train_rf_redwine <- train(quality ~ ., data = train_set_redwine, method = "rf",
                  trControl = control, tuneGrid = grid_rf,
                  preProcess = c("center", "scale"))
plot(train_rf_redwine)
```




```
train_rf_redwine$bestTune
```

```
##   mtry
## 1    1
```

```
# Record execution end-time
run_log <- bind_rows(run_log,tibble(Activity = "RF Red End:", Time = Sys.time()))
```

As per result above, mtry = 1 is the best value for Red Wine data.

#### 2.3.1.3 Support Vector Machine (SVM) on Red Wine data

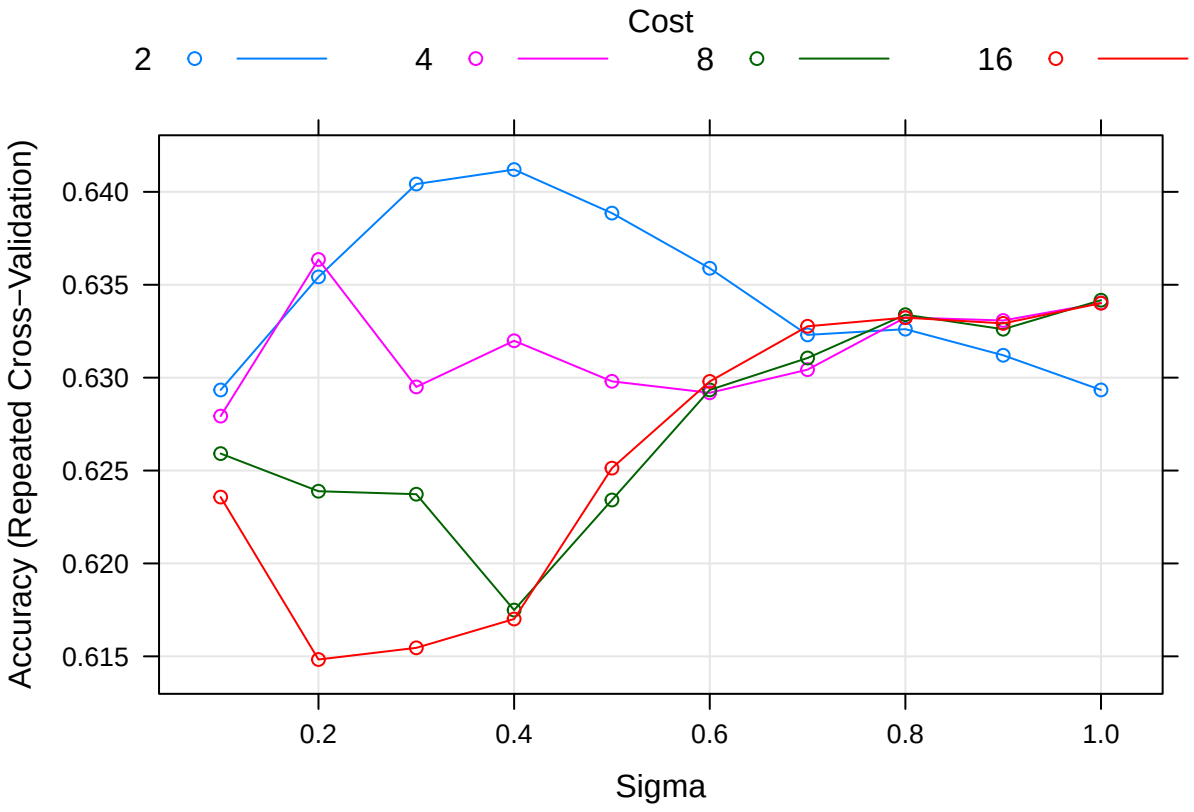
The Radial Basis Function (svmRadial) will be used as the kernel.

After a few trial-and-errors, 10 sigma values between 0.1 and 1; 4 values between 2 and 16 will be used for cost.

```
# Model 3 - Support Vector Machine on Red Wine
# The Radial Basis Function (svmRadial) used
# 10 sigma values between 0.1 and 1
# Cost: 4 values between 2 and 16

# Record execution start-time
run_log <- bind_rows(run_log,tibble(Activity = "SVM Red Start:", Time = Sys.time()))

grid_svm <- expand.grid(C = 2^(1:4), sigma = seq(0.1, 1, length = 10))
train_svm_redwine <- train(quality ~ ., data = train_set_redwine, method = "svmRadial",
                   trControl = control, tuneGrid = grid_svm,
                   preProcess = c("center", "scale"))
plot(train_svm_redwine)
```




```
train_svm_redwine$bestTune
```

```
##   sigma C
## 4   0.4 2
```

```r
# Record execution end-time
run_log <- bind_rows(run_log,tibble(Activity = "SVM Red End:", Time = Sys.time()))
```

As per result above, a cost of 2 and a sigma of 0.4 works most accurately for Red Wine data.

### 2.3.2 Model Building on White Wine data set

The following codes split the white wine data into training and test set in the ratio of 80:20.

```
# Prepare training and test data sets
whitewine$quality <- as.factor(whitewine$quality)
set.seed(2006, sample.kind="Rounding") #Use Year - Month - day of today's date, 2020-06-08, the first d
train_index_whitewine <- createDataPartition(whitewine$quality, p = 0.8, list = F)
train_set_whitewine <- whitewine[train_index_whitewine,]
test_set_whitewine <- whitewine[-train_index_whitewine,]
```

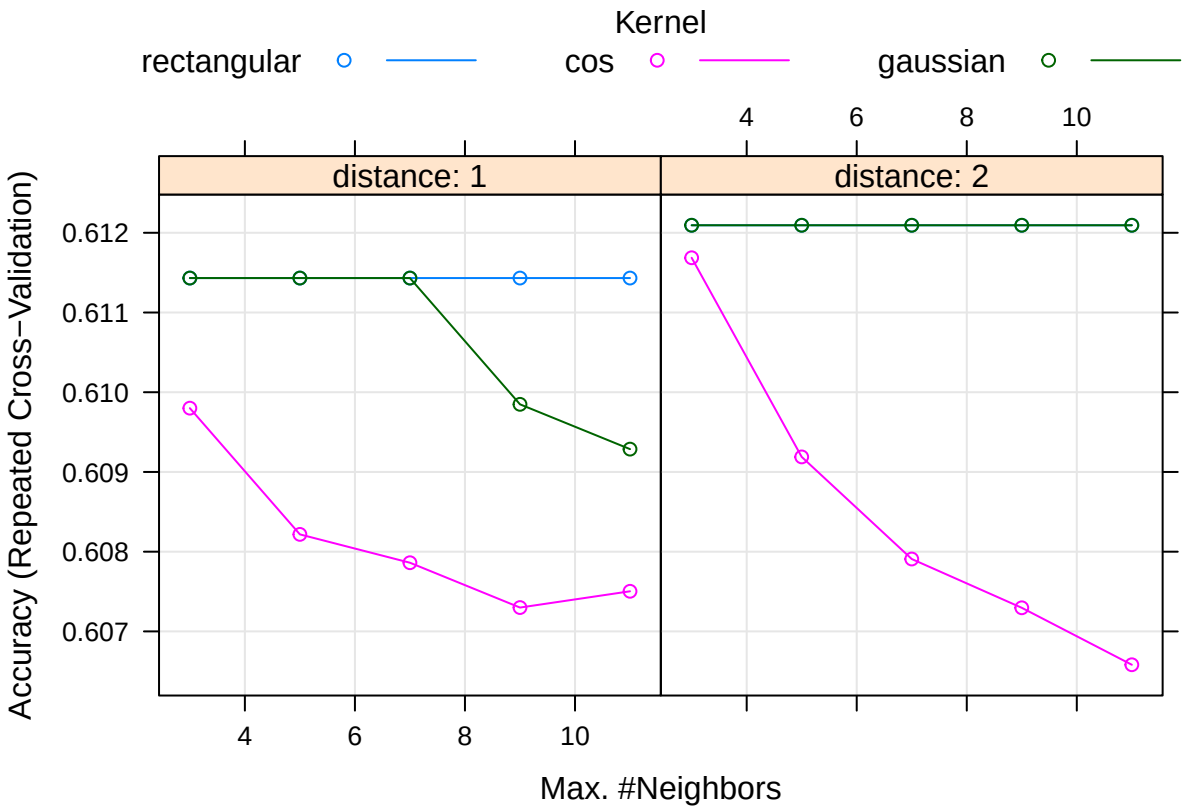
#### 2.3.2.1 k-nearest neighbours on White Wine data

5 kmax, 2 distances (1 for Manahattan and 2 for Euclidian) and 3 kernel values will be used.

```
# Model 1 - k-nearest neighbours on White Wine
# 5 kmax, 2 distances, 3 kernels

# Record execution start-time
run_log <- bind_rows(run_log,tibble(Activity = "KNN White Start:", Time = Sys.time()))

control <- trainControl(method = "repeatedcv", number = 5, repeats = 5)
grid_kknn <- expand.grid(kmax = c(3, 5, 7, 9, 11), distance = c(1, 2),
                         kernel = c("rectangular", "cos", "gaussian"))
train_kknn_whitewine <- train(quality ~ ., data = train_set_whitewine, method = "kknn",
                              trControl = control, tuneGrid = grid_kknn,
                              preProcess = c("center", "scale"))
plot(train_kknn_whitewine)
```




```
train_kknn_whitewine$bestTune
```

```
##    kmax distance      kernel
## 28   11        2 rectangular
```

```
# Record execution end-time
run_log <- bind_rows(run_log,tibble(Activity = "KNN White End:", Time = Sys.time()))
```

As per result above, the rectangular kernel using Euclidian distance and k value of 11 works best for White Wine data.

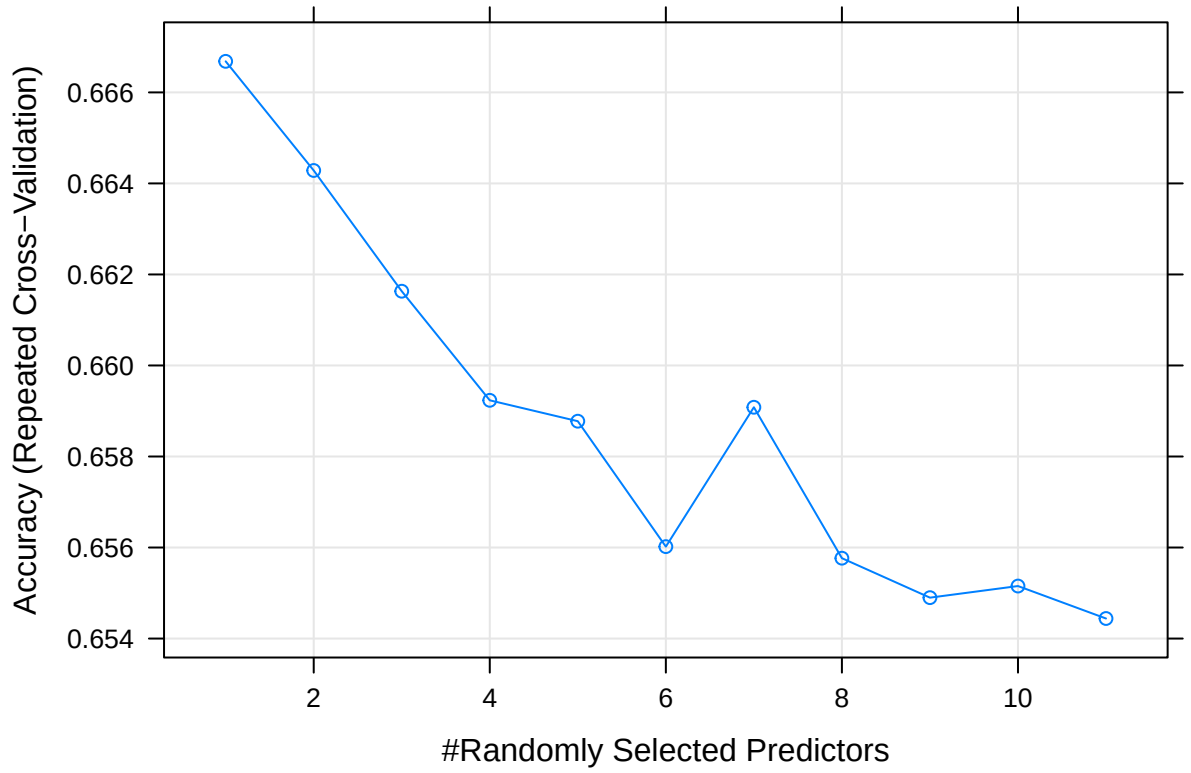
#### 2.3.2.2 Random Forest on White Wine data

Mtry is the number of variables randomly sampled as candidates at each split and it is the only hyperparameter available for tuning. The values from 1 to 11 are used in the tuneGrid argument.

```
# Model 2 - Random Forest on White Wine
# mtry from 1 through 11

# Record execution start-time
run_log <- bind_rows(run_log,tibble(Activity = "RF White Start:", Time = Sys.time()))

control <- trainControl(method = "repeatedcv", number = 5, repeats = 5)
grid_rf <- expand.grid(mtry = 1:11)
train_rf_whitewine <- train(quality ~ ., data = train_set_whitewine, method = "rf",
                            trControl = control, tuneGrid = grid_rf,
                            preProcess = c("center", "scale"))
plot(train_rf_whitewine)
```




```
train_rf_whitewine$bestTune
```

```
##   mtry
## 1    1
```

```
# Record execution end-time
run_log <- bind_rows(run_log,tibble(Activity = "RF White End:", Time = Sys.time()))
```

As per result above, mtry = 1 is the best value for White Wine data.

#### 2.3.2.3 Support Vector Machine (SVM) on White Wine data

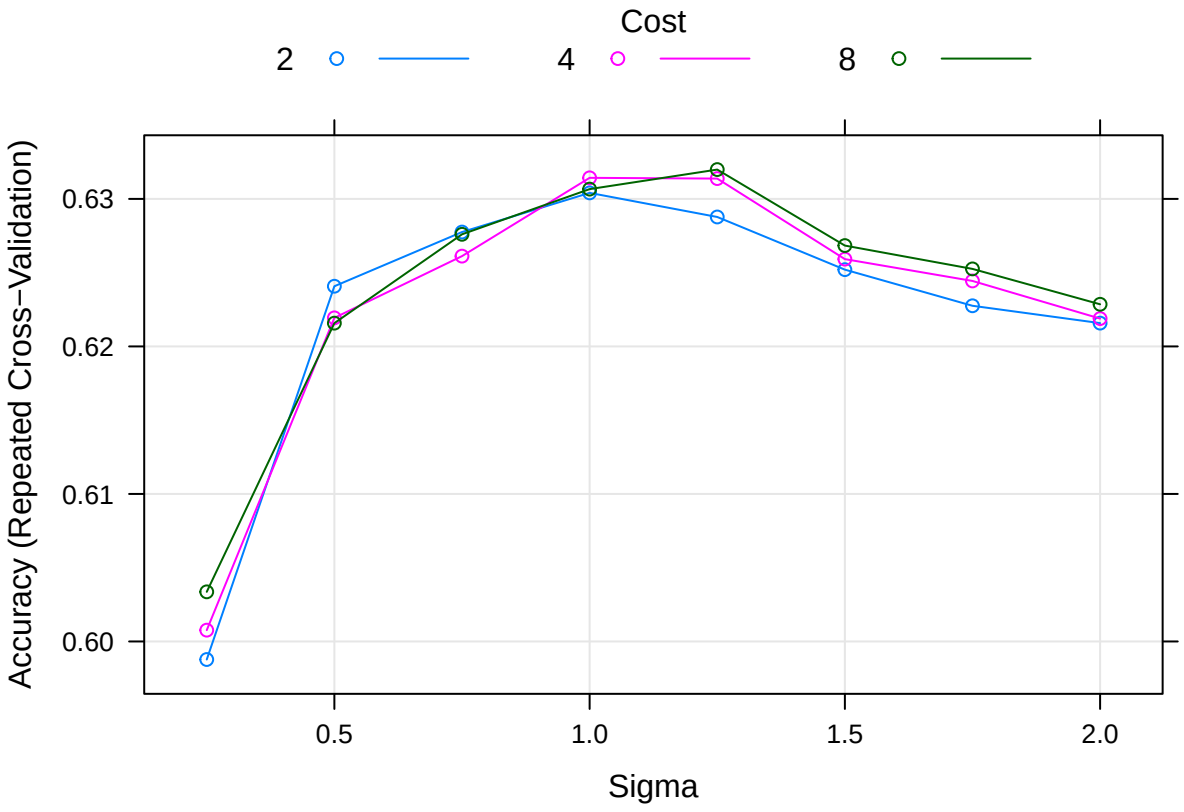
The Radial Basis Function (svmRadial) will be used as the kernel.

After a few trial-and-errors, 8 sigma values between 0.25 and 2; 3 values between 2 and 8 will be used for cost.

```
# Model 3 - Support Vector Machine on White Wine
# The Radial Basis Function (svmRadial) used
# 8 sigma values between 0.25 and 2
# Cost: 3 values between 2 and 8

# Record execution start-time
run_log <- bind_rows(run_log,tibble(Activity = "SVM White Start:", Time = Sys.time()))

grid_svm <- expand.grid(C = 2^(1:3), sigma = seq(0.25, 2, length = 8))
train_svm_whitewine <- train(quality ~ ., data = train_set_whitewine, method = "svmRadial",
                    trControl = control, tuneGrid = grid_svm,
                    preProcess = c("center", "scale"))
plot(train_svm_whitewine)
```




```
train_svm_whitewine$bestTune
```

```
##    sigma C
## 21  1.25 8
```

```
# Record execution end-time
run_log <- bind_rows(run_log,tibble(Activity = "SVM White End:", Time = Sys.time()))
```

As per result above, a cost of 8 and a sigma of 1.25 works most accurately for White Wine data.

# Section 3: Model Testing and Results

In this section, for each of the trained models, we will use predict() function to run on the respective test data sets. confusionMatrix() will be used to present the results and the overall accuracy values will be consolidated into a summary tibble for final presentation of results.

## 3.1 KNN on Red Wine Data

```
# Model 1 - k-nearest neighbours (Red)
predict_kknn_redwine <- predict(train_kknn_redwine, test_set_redwine)
cm_kknn_redwine <- confusionMatrix(predict_kknn_redwine, test_set_redwine$quality)
accuracy_kknn_redwine <- cm_kknn_redwine$overall['Accuracy']

# Show result
cm_kknn_redwine
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  0  0  3  6  0  0
##          5  2  7 96 25  5  0
##          6  0  3 32 81 13  0
##          7  0  0  5 15 19  2
##          8  0  0  0  0  2  1
##
## Overall Statistics
##
##                Accuracy : 0.6215
##                  95% CI : (0.5655, 0.6751)
##     No Information Rate : 0.429
##     P-Value [Acc > NIR] : 4.378e-12
##
##                   Kappa : 0.4061
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000  0.00000   0.7059   0.6378  0.48718 0.333333
## Specificity          1.000000  0.97068   0.7845   0.7474  0.92086 0.993631
## Pos Pred Value            NaN  0.00000   0.7111   0.6279  0.46341 0.333333
## Neg Pred Value       0.993691  0.96753   0.7802   0.7553  0.92754 0.993631
## Prevalence           0.006309  0.03155   0.4290   0.4006  0.12303 0.009464
## Detection Rate       0.000000  0.00000   0.3028   0.2555  0.05994 0.003155
## Detection Prevalence 0.000000  0.02839   0.4259   0.4069  0.12934 0.009464
## Balanced Accuracy    0.500000  0.48534   0.7452   0.6926  0.70402 0.663482
```

```
accuracy_kknn_redwine
```

```
##  Accuracy
## 0.6214511
```

Overall accuracy for KNN on Red Wine data is 0.6214511.

### 3.2 KNN on White Wine Data

```
# Model 1 - k-nearest neighbours (White)
predict_kknn_whitewine <- predict(train_kknn_whitewine, test_set_whitewine)
cm_kknn_whitewine <- confusionMatrix(predict_kknn_whitewine, test_set_whitewine$quality)
accuracy_kknn_whitewine <- cm_kknn_whitewine$overall['Accuracy']

# Show result
cm_kknn_whitewine
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   1   0   1   0   0   0
##          4   0  11   8   2   1   0   0
##          5   2  13 196  67   7   1   0
##          6   2   5  77 304  52   9   1
##          7   0   2   9  58 112  10   0
##          8   0   0   1   7   4  15   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6524
##                  95% CI : (0.6216, 0.6822)
##     No Information Rate : 0.4489
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4808
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000  0.34375   0.6735   0.6925   0.6364  0.42857
## Specificity          0.997947  0.98837   0.8690   0.7291   0.9015  0.98727
## Pos Pred Value       0.000000  0.50000   0.6853   0.6756   0.5864  0.55556
## Neg Pred Value       0.995902  0.97803   0.8627   0.7443   0.9187  0.97897
## Prevalence           0.004090  0.03272   0.2975   0.4489   0.1800  0.03579
## Detection Rate       0.000000  0.01125   0.2004   0.3108   0.1145  0.01534
## Detection Prevalence 0.002045  0.02249   0.2924   0.4601   0.1953  0.02761
## Balanced Accuracy    0.498973  0.66606   0.7713   0.7108   0.7689  0.70792
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.998978
## Prevalence           0.001022
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

```
accuracy_kknn_whitewine
```

```
##  Accuracy
## 0.6523517
```

Overall accuracy for KNN on White Wine data is 0.6523517.

### 3.3 Random Forest on Red Wine Data

```
# Model 2 - Random Forest (Red)
predict_rf_redwine <- predict(train_rf_redwine, test_set_redwine)
cm_rf_redwine <- confusionMatrix(predict_rf_redwine, test_set_redwine$quality)
accuracy_rf_redwine <- cm_rf_redwine$overall['Accuracy']

# Show result
cm_rf_redwine
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   0   0   0   0
##          5   1   8 108  35   3   0
##          6   1   2  26  86  19   2
##          7   0   0   2   6  15   0
##          8   0   0   0   0   2   1
##
## Overall Statistics
##
##                Accuracy : 0.6625
##                  95% CI : (0.6075, 0.7144)
##     No Information Rate : 0.429
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.446
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000  0.00000   0.7941   0.6772  0.38462 0.333333
## Specificity          1.000000  1.00000   0.7403   0.7368  0.97122 0.993631
## Pos Pred Value            NaN      NaN   0.6968   0.6324  0.65217 0.333333
## Neg Pred Value       0.993691  0.96845   0.8272   0.7735  0.91837 0.993631
## Prevalence           0.006309  0.03155   0.4290   0.4006  0.12303 0.009464
## Detection Rate       0.000000  0.00000   0.3407   0.2713  0.04732 0.003155
## Detection Prevalence 0.000000  0.00000   0.4890   0.4290  0.07256 0.009464
## Balanced Accuracy    0.500000  0.50000   0.7672   0.7070  0.67792 0.663482
```

```
accuracy_rf_redwine
```

```
##  Accuracy
## 0.6624606
```

Overall accuracy for RF on Red Wine data is 0.6624606.

### 3.4 Random Forest on White Wine Data

```
# Model 2 - Random Forest (White)
predict_rf_whitewine <- predict(train_rf_whitewine, test_set_whitewine)
cm_rf_whitewine <- confusionMatrix(predict_rf_whitewine, test_set_whitewine$quality)
accuracy_rf_whitewine <- cm_rf_whitewine$overall['Accuracy']

# Show result
cm_rf_whitewine
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   0   5   2   1   0   0   0
##          5   2  16 200  50   2   1   0
##          6   2  11  89 367  73  14   1
##          7   0   0   0  20 101   6   0
##          8   0   0   0   1   0  14   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.7025
##                  95% CI : (0.6727, 0.731)
##     No Information Rate : 0.4489
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5334
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000 0.156250   0.6873   0.8360   0.5739  0.40000
## Specificity           1.00000 0.996829   0.8967   0.6475   0.9676  0.99894
## Pos Pred Value            NaN 0.625000   0.7380   0.6589   0.7953  0.93333
## Neg Pred Value        0.99591 0.972165   0.8713   0.8290   0.9119  0.97819
## Prevalence            0.00409 0.032720   0.2975   0.4489   0.1800  0.03579
## Detection Rate        0.00000 0.005112   0.2045   0.3753   0.1033  0.01431
## Detection Prevalence  0.00000 0.008180   0.2771   0.5695   0.1299  0.01534
## Balanced Accuracy     0.50000 0.576539   0.7920   0.7417   0.7707  0.69947
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.998978
## Prevalence           0.001022
## Detection Rate       0.000000
## Detection Prevalence 0.000000
## Balanced Accuracy    0.500000
```

```
accuracy_rf_whitewine
```

```
## Accuracy
## 0.702454
```

Overall accuracy for RF on White Wine data is 0.702454.

### 3.5 SVM on Red Wine Data

```
# Model 3 - SVM (Red)
predict_svm_redwine <- predict(train_svm_redwine, test_set_redwine)
cm_svm_redwine <- confusionMatrix(predict_svm_redwine, test_set_redwine$quality)

accuracy_svm_redwine <- cm_svm_redwine$overall['Accuracy']

# Show result
cm_svm_redwine
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   0   0   0   0
##          4   0   0   0   1   0   0
##          5   1   7 110  43   4   0
##          6   1   3  24  76  17   2
##          7   0   0   2   7  16   1
##          8   0   0   0   0   2   0
##
## Overall Statistics
##
##                Accuracy : 0.6372
##                  95% CI : (0.5816, 0.6902)
##     No Information Rate : 0.429
##     P-Value [Acc > NIR] : 7.119e-14
##
##                   Kappa : 0.4063
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity          0.000000 0.000000   0.8088   0.5984  0.41026 0.000000
## Specificity          1.000000 0.996743   0.6961   0.7526  0.96403 0.993631
## Pos Pred Value            NaN 0.000000   0.6667   0.6179  0.61538 0.000000
## Neg Pred Value       0.993691 0.968354   0.8289   0.7371  0.92096 0.990476
## Prevalence           0.006309 0.031546   0.4290   0.4006  0.12303 0.009464
## Detection Rate       0.000000 0.000000   0.3470   0.2397  0.05047 0.000000
## Detection Prevalence 0.000000 0.003155   0.5205   0.3880  0.08202 0.006309
## Balanced Accuracy    0.500000 0.498371   0.7525   0.6755  0.68714 0.496815
```

```
accuracy_svm_redwine
```

```
## Accuracy
## 0.637224
```

Overall accuracy for SVM on Red Wine data is 0.637224.

### 3.6 SVM on White Wine Data

```
# Model 3 - SVM (White)
predict_svm_whitewine <- predict(train_svm_whitewine, test_set_whitewine)
cm_svm_whitewine <- confusionMatrix(predict_svm_whitewine, test_set_whitewine$quality)

accuracy_svm_whitewine <- cm_svm_whitewine$overall['Accuracy']

# Show result
cm_svm_whitewine
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   3   4   5   6   7   8   9
##          3   0   0   0   0   0   0   0
##          4   0   5   0   1   0   0   0
##          5   0   9 177  39   2   0   0
##          6   4  18 112 372  78  16   1
##          7   0   0   2  26  95   5   0
##          8   0   0   0   1   1  14   0
##          9   0   0   0   0   0   0   0
##
## Overall Statistics
##
##                Accuracy : 0.6779
##                  95% CI : (0.6476, 0.7071)
##     No Information Rate : 0.4489
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4894
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 3 Class: 4 Class: 5 Class: 6 Class: 7 Class: 8
## Sensitivity           0.00000 0.156250   0.6082   0.8474  0.53977  0.40000
## Specificity           1.00000 0.998943   0.9272   0.5751  0.95885  0.99788
## Pos Pred Value            NaN 0.833333   0.7797   0.6190  0.74219  0.87500
## Neg Pred Value        0.99591 0.972222   0.8482   0.8223  0.90471  0.97817
## Prevalence            0.00409 0.032720   0.2975   0.4489  0.17996  0.03579
## Detection Rate        0.00000 0.005112   0.1810   0.3804  0.09714  0.01431
## Detection Prevalence  0.00000 0.006135   0.2321   0.6145  0.13088  0.01636
## Balanced Accuracy     0.50000 0.577596   0.7677   0.7113  0.74931  0.69894
##                      Class: 9
## Sensitivity          0.000000
## Specificity          1.000000
## Pos Pred Value            NaN
## Neg Pred Value       0.998978
## Prevalence           0.001022
## Detection Rate       0.000000
## Detection Prevalence 0.000000
```

```
## Balanced Accuracy    0.500000
```

```
accuracy_svm_whitewine
```

```
##  Accuracy
## 0.6779141
```

Overall accuracy for SVM on White Wine data is 0.6779141.

## 3.7 Final Results: a consolidated view

The following table consolidates and summarises all the overall accuarcy results of the respective tested models.

Random Forest models produce the best overall accuracy values on both the red and white wine data sets.

The accuracy value on White Wine data is higher than the same on Red Wine data, that is likely because the sample data size of the White Wine data set is much larger than the same of the Red Wine data set and therefore more training data was available and hence better accuracy.

| Model | Accuracy on Red Wine | Accuracy on White Wine |
|---|---|---|
| KNN | 0.6214511 | 0.6523517 |
| RF | 0.6624606 | 0.7024540 |
| SVM | 0.6372240 | 0.6779141 |

## Section 4: Conclusion

While it has been the right decision to choose non-linear classification models for training and testing as they are more appropriate than regression, there are a few areas worth fine-tuning:

- Feature selection: some input variables such as free.sulfur.dioxide for both red and white wine data and residual.sugar for the case of red wine data are less important and could have been removed in order to reduce the footprint of the dimensionality and hence improve the training time and efficiency.
- Removal of outliers: although reasons were given not to remove the outliers in Section 2, it was more of an assumption that such outliers would not affect the overall model building and testing, some due diligence could have been done.

Whether the selected models can be used for practical use is still undetermined for a few reasons:

- overall accuracy which hovers around high of 0.6 and low of 0.7 is still considered not high enough to give confidence from the commercial point of view;
- accuracy values on good and bad quality wines are low, therefore there is no practical usage of the model if it is only good at predicting average or mediocre wines.

Some future considerations could be:

- from wine producer's perspective - include non-physiochemical attributes such as origin of grapes, cost of production into the overall data set, so that machine learning models can be built to support more commercial use-cases such as deciding what grapes produce the best-performance wines, i.e. best quality with lowest cost; or which origin should the wine producers focus on increasing grape production based on the past data, etc.
- from consumers' perspective - include information on brand and year of production, so that machine learning models can be used to build a recommendation system that predicts consumers' preference and recommends wines that have similar properties.

*** THE END ***