1st Objective (partitioning clustering)

Load required libraries:

```
> library(tidyverse)
> library(NbClust)
> library(fpc)
> library(MASS)
> library(flexclust)
>
```

Data exploration, preparation and processing.
Load the Whitewine dataset and check the structure of each variable:

```
> white_wine <- readxl::read_xlsx("whitewine.xlsx", sheet = 1)
> str(white_wine)
Classes ‘tbl_df’, ‘tbl’ and '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             : num  6 6 6 6 6 6 6 6 6 6 ...
>
```

All 12 variables appear to be numeric, although we need quality to be categorical. This will be done later.

Let’s summarize the whitewine dataset to see some basic stats:

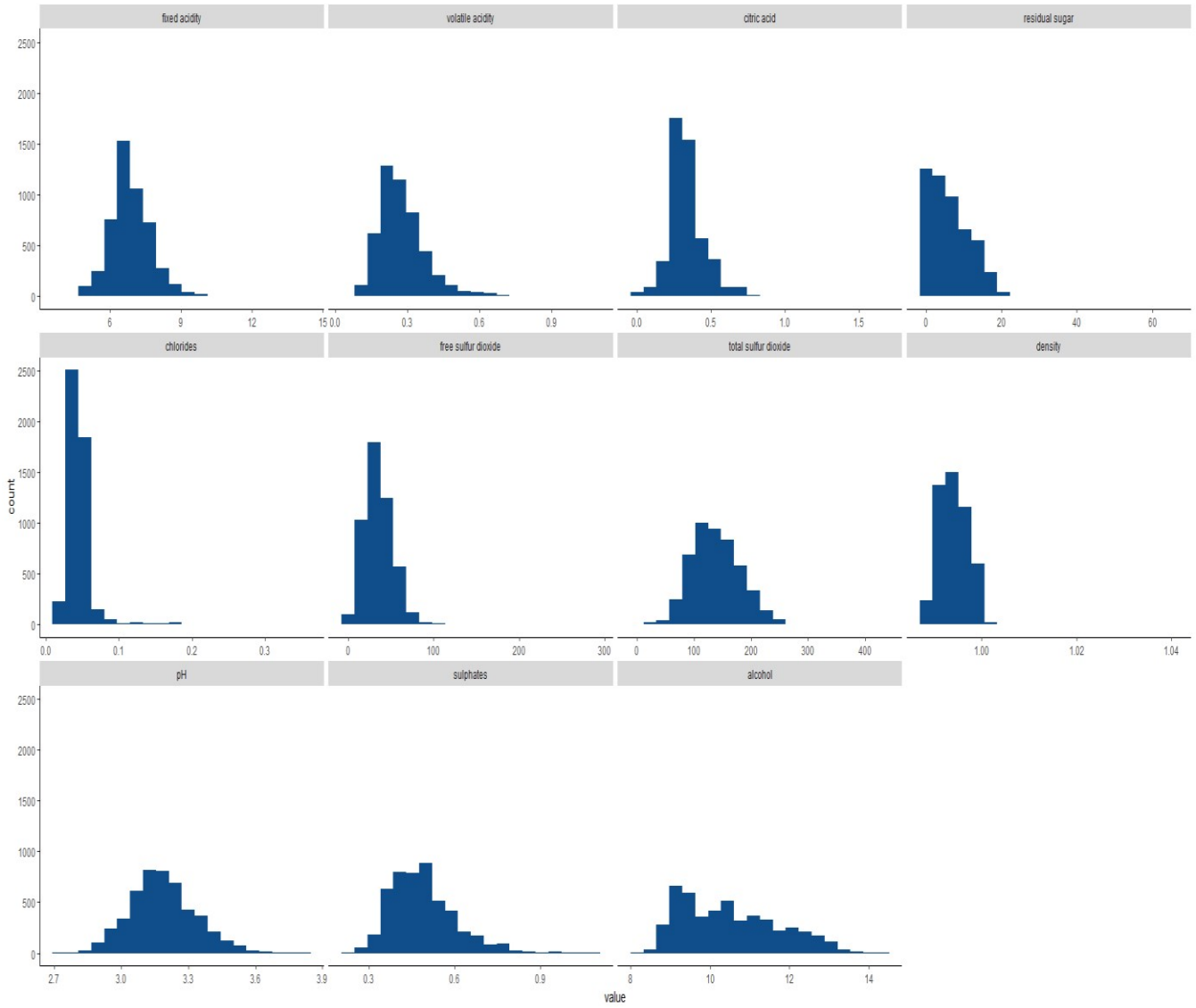
```
> summary(white_wine)
 fixed acidity    volatile acidity  citric acid     residual sugar     chlorides
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200   Median :0.04300
 Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391   Mean   :0.04577
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900   3rd Qu.:0.05000
 Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800   Max.   :0.34600
 free sulfur dioxide total sulfur dioxide    density             pH          sulphates
 Min.   :  2.00      Min.   :  9.0        Min.   :0.9871   Min.   :2.720   Min.   :0.2200
 1st Qu.: 23.00      1st Qu.:108.0        1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100
 Median : 34.00      Median :134.0        Median :0.9937   Median :3.180   Median :0.4700
 Mean   : 35.31      Mean   :138.4        Mean   :0.9940   Mean   :3.188   Mean   :0.4898
 3rd Qu.: 46.00      3rd Qu.:167.0        3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500
 Max.   :289.00      Max.   :440.0        Max.   :1.0390   Max.   :3.820   Max.   :1.0800
    alcohol         quality
 Min.   : 8.00   Min.   :3.000
 1st Qu.: 9.50   1st Qu.:5.000
 Median :10.40   Median :6.000
 Mean   :10.51   Mean   :5.878
 3rd Qu.:11.40   3rd Qu.:6.000
 Max.   :14.20   Max.   :9.000
> |
```

There happen to be no missing value.

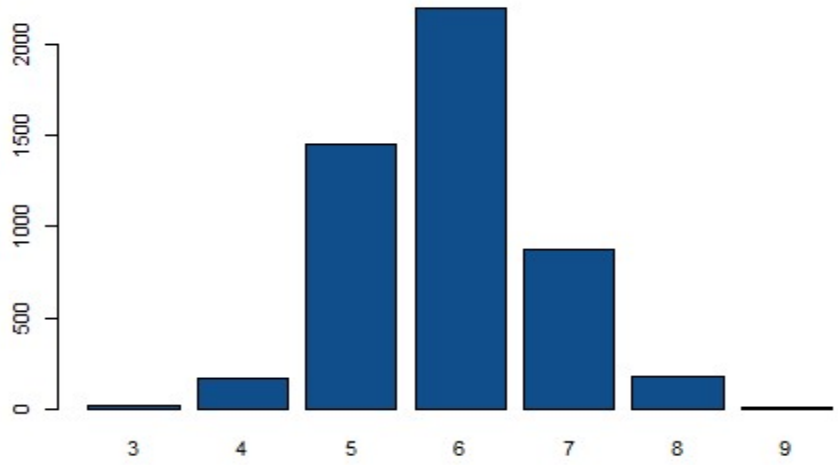
```
> anyNA(white_wine)
[1] FALSE
> |
```

Next is to plot the histogram of the first 11 variables to see how the data is being distributed:

```
> ggplot(data = reshape2::melt(white_wine[, -12]), mapping = aes(x = value)) +
+   geom_histogram(bins = 20, fill = "dodgerblue4") + facet_wrap(~variable, scales = 'free_x')+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
```



Bar plot of the quality

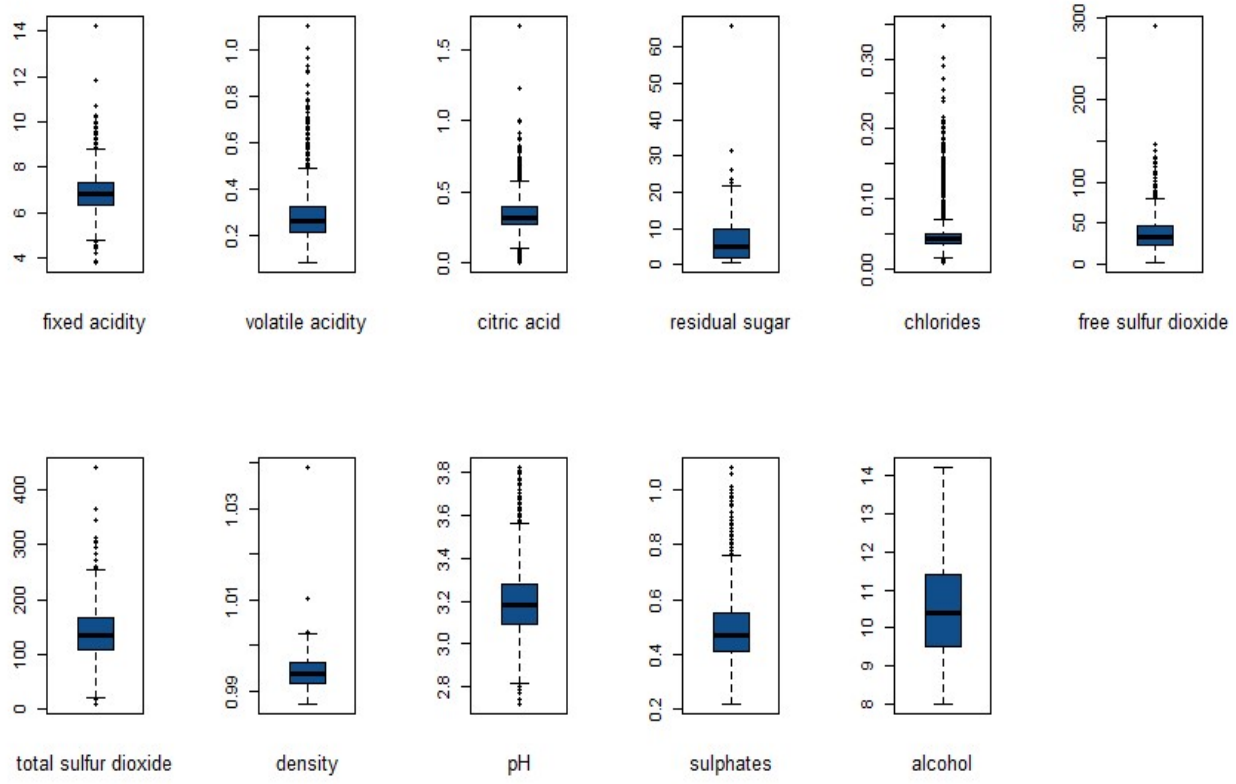



```
> unique(white_wine$quality)
[1] 6 5 7 8 4 3 9
> length(unique(white_wine$quality))
[1] 7
>
> table(white_wine$quality)

   3    4    5    6    7    8    9
  20  163 1457 2198  880  175    5
>
> round(100*prop.table(table(white_wine$quality)), 2)

    3     4     5     6     7     8     9
 0.41  3.33 29.75 44.88 17.97  3.57  0.10
> paste0(round(100*prop.table(table(white_wine$quality)), 2), "%")
[1] "0.41%"  "3.33%"  "29.75%" "44.88%" "17.97%" "3.57%"  "0.1%"
>
```

There are 7 classes in the quality variable (ranging from 3 to 9) with most of the wine tested between quality 5 and 7. Although with some outliers, some variables such as Fixed acidity, total Sulphur dioxide, PH appear to be normally distributed. The fact that most are in favour of normal distribution but none of the properties appear to be uniformly distributed will form a basis for scaling decision. Hence, standardizing the 11 variables is appropriate for this analysis. Moreover, each variable in our dataset are of different standard unit. Meanwhile, we still need to deal with the outliers present in the dataset.

Dealing with outliers:
Let’s visualize using box to see how the outliers plays out:

```
> par( mfrow = c( 2, 6 ) )
> boxplot(white_wine$`fixed acidity`, col= "dodgerblue4", pch=19)
> mtext("fixed acidity", cex=0.8, side=1, line=2)
> boxplot(white_wine$`volatile acidity`, col= "dodgerblue4", pch=19)
> mtext("volatile acidity", cex=0.8, side=1, line=2)
> boxplot(white_wine$`citric acid`, col= "dodgerblue4", pch=19)
> mtext("citric acid", cex=0.8, side=1, line=2)
> boxplot(white_wine$`residual sugar`, col= "dodgerblue4", pch=19)
> mtext("residual sugar", cex=0.8, side=1, line=2)
> boxplot(white_wine$chlorides, col= "dodgerblue4", pch=19)
> mtext("chlorides", cex=0.8, side=1, line=2)
> boxplot(white_wine$`free sulfur dioxide`, col= "dodgerblue4", pch=19)
> mtext("free sulfur dioxide", cex=0.8, side=1, line=2)
> boxplot(white_wine$`total sulfur dioxide`, col= "dodgerblue4", pch=19)
> mtext("total sulfur dioxide", cex=0.8, side=1, line=2)
> boxplot(white_wine$density, col= "dodgerblue4", pch=19)
> mtext("density", cex=0.8, side=1, line=2)
> boxplot(white_wine$pH, col= "dodgerblue4", pch=19)
> mtext("pH", cex=0.8, side=1, line=2)
> boxplot(white_wine$sulphates, col= "dodgerblue4", pch=19)
> mtext("sulphates", cex=0.8, side=1, line=2)
> boxplot(white_wine$alcohol, col= "dodgerblue4", pch=19)
> mtext("alcohol", cex=0.8, side=1, line=2)
```



Fixed acidity, volatile acidity, citric acid, total Sulphur dioxide, PH and sulphates and have outliers. If those outliers are eliminated distribution of the variables may be taken to be symmetric.

Residual sugar has a positively skewed distribution; even after eliminating the outliers distribution will possibly remain skewed.
Some of the variables like sulphur dioxide and density, have a few outliers but these are very different from the rest. Alcohol has an irregular shaped distribution but it does not have pronounced outliers.

We can further investigate the presence of outliers leveraging summary function and describe function from psych library:

```
> library("psych")
>
> psych::describe(white_wine[, -12])
                      vars    n   mean    sd median trimmed   mad  min    max  range skew kurtosis   se
fixed acidity            1 4898   6.85  0.84   6.80    6.82  0.74 3.80  14.20  10.40 0.65     2.17 0.01
volatile acidity         2 4898   0.28  0.10   0.26    0.27  0.09 0.08   1.10   1.02 1.58     5.08 0.00
citric acid              3 4898   0.33  0.12   0.32    0.33  0.09 0.00   1.66   1.66 1.28     6.16 0.00
residual sugar           4 4898   6.39  5.07   5.20    5.80  5.34 0.60  65.80  65.20 1.08     3.46 0.07
chlorides                5 4898   0.05  0.02   0.04    0.04  0.01 0.01   0.35   0.34 5.02    37.51 0.00
free sulfur dioxide      6 4898  35.31 17.01  34.00   34.36 16.31 2.00 289.00 287.00 1.41    11.45 0.24
total sulfur dioxide     7 4898 138.36 42.50 134.00  136.96 43.00 9.00 440.00 431.00 0.39     0.57 0.61
density                  8 4898   0.99  0.00   0.99    0.99  0.00 0.99   1.04   0.05 0.98     9.78 0.00
pH                       9 4898   3.19  0.15   3.18    3.18  0.15 2.72   3.82   1.10 0.46     0.53 0.00
sulphates               10 4898   0.49  0.11   0.47    0.48  0.10 0.22   1.08   0.86 0.98     1.59 0.00
alcohol                 11 4898  10.51  1.23  10.40   10.43  1.48 8.00  14.20   6.20 0.49    -0.70 0.02
> summary(white_wine[, -12])
 fixed acidity  volatile acidity  citric acid   residual sugar   chlorides    free sulfur dioxide total sulfur dioxide
 Min.   : 3.8   Min.   :0.08     Min.   :0.00   Min.   : 1     Min.   :0.01   Min.   :  2         Min.   :  9
 1st Qu.: 6.3   1st Qu.:0.21     1st Qu.:0.27   1st Qu.: 2     1st Qu.:0.04   1st Qu.: 23         1st Qu.:108
 Median : 6.8   Median :0.26     Median :0.32   Median : 5     Median :0.04   Median : 34         Median :134
 Mean   : 6.9   Mean   :0.28     Mean   :0.33   Mean   : 6     Mean   :0.05   Mean   : 35         Mean   :138
 3rd Qu.: 7.3   3rd Qu.:0.32     3rd Qu.:0.39   3rd Qu.:10     3rd Qu.:0.05   3rd Qu.: 46         3rd Qu.:167
 Max.   :14.2   Max.   :1.10     Max.   :1.66   Max.   :66     Max.   :0.35   Max.   :289         Max.   :440
    density           pH        sulphates       alcohol
 Min.   :0.99   Min.   :2.7   Min.   :0.22   Min.   : 8.0
 1st Qu.:0.99   1st Qu.:3.1   1st Qu.:0.41   1st Qu.: 9.5
 Median :0.99   Median :3.2   Median :0.47   Median :10.4
 Mean   :0.99   Mean   :3.2   Mean   :0.49   Mean   :10.5
 3rd Qu.:1.00   3rd Qu.:3.3   3rd Qu.:0.55   3rd Qu.:11.4
 Max.   :1.04   Max.   :3.8   Max.   :1.08   Max.   :14.2
>
```

It can be observed that range is much larger than the IQR, and mean is larger than the median. These observations supports our suspicion that there are outliers which must be taken care of in the data set and before analysis.

Next we look at the bivariate analysis:
1. Correlation coefficients and
2. Pairwise scatterplot

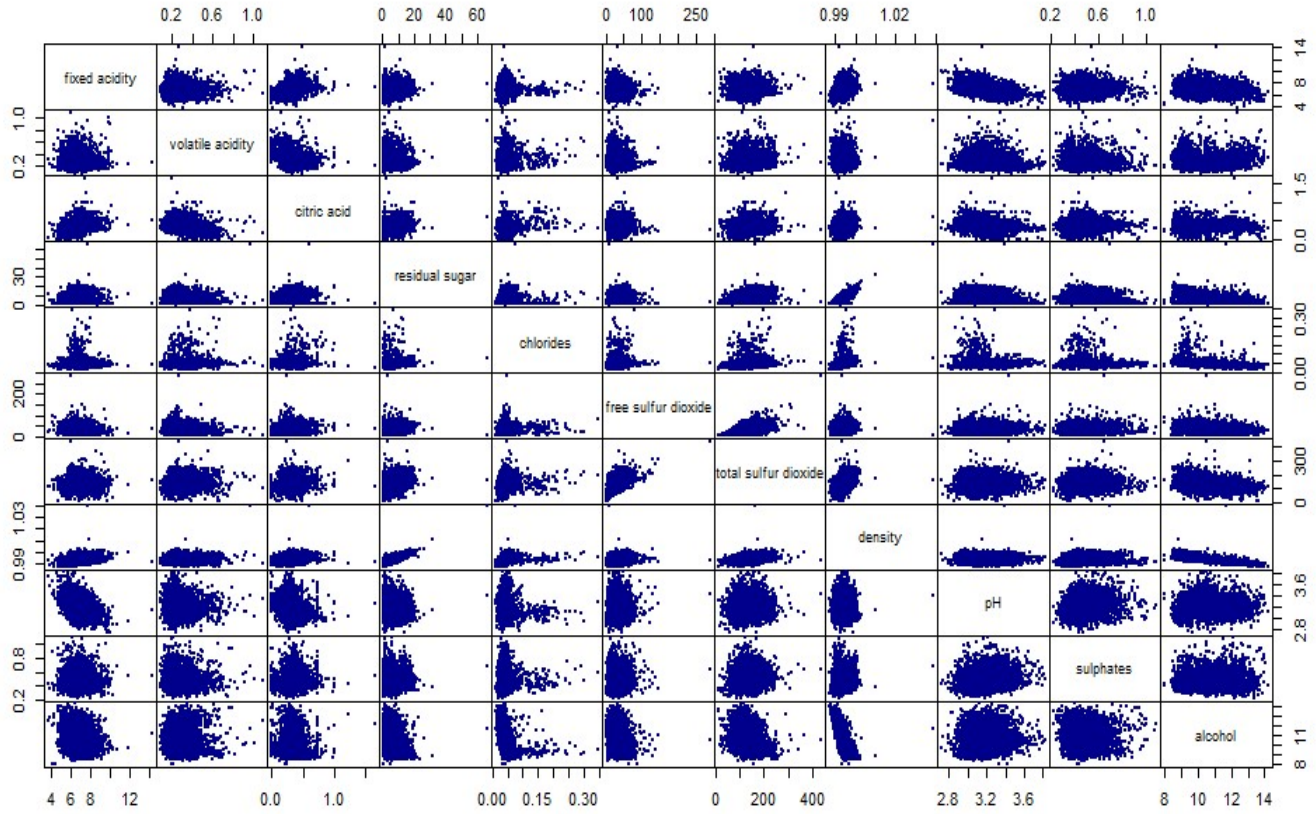
Pearson’s correlation coefficients

```
> round(cor(white_wine[, -12], method = "pearson"), 2)
                      fixed acidity volatile acidity citric acid residual sugar chlorides
fixed acidity                  1.00            -0.02        0.29          0.09      0.02
volatile acidity              -0.02             1.00       -0.15          0.06      0.07
citric acid                    0.29            -0.15        1.00          0.09      0.11
residual sugar                 0.09             0.06        0.09          1.00      0.09
chlorides                      0.02             0.07        0.11          0.09      1.00
free sulfur dioxide           -0.05            -0.10        0.09          0.30      0.10
total sulfur dioxide           0.09             0.09        0.12          0.40      0.20
density                        0.27             0.03        0.15          0.84      0.26
pH                            -0.43            -0.03       -0.16         -0.19     -0.09
sulphates                     -0.02            -0.04        0.06         -0.03      0.02
alcohol                       -0.12             0.07       -0.08         -0.45     -0.36
                      free sulfur dioxide total sulfur dioxide density    pH sulphates alcohol
fixed acidity                       -0.05                 0.09    0.27 -0.43     -0.02   -0.12
volatile acidity                    -0.10                 0.09    0.03 -0.03     -0.04    0.07
citric acid                          0.09                 0.12    0.15 -0.16      0.06   -0.08
residual sugar                       0.30                 0.40    0.84 -0.19     -0.03   -0.45
chlorides                            0.10                 0.20    0.26 -0.09      0.02   -0.36
free sulfur dioxide                  1.00                 0.62    0.29  0.00      0.06   -0.25
total sulfur dioxide                 0.62                 1.00    0.53  0.00      0.13   -0.45
density                              0.29                 0.53    1.00 -0.09      0.07   -0.78
pH                                   0.00                 0.00   -0.09  1.00      0.16    0.12
sulphates                            0.06                 0.13    0.07  0.16      1.00   -0.02
alcohol                             -0.25                -0.45   -0.78  0.12     -0.02    1.00
```

The chemical properties with high correlation are highlighted in colors. I have considered correlation of 40% and above as high.

Pairwise scatterplot

```
> pairs(cor(white_wine[, -12]), gap=0, pch=19, cex=0.4, col="darkblue")
```



In lieu of the above, I have decided to apply the boxplot outlier detection method which imply that if a variable has a value more than Q3 + 1.5IQR and if has a value less than Q1 - 1.5IQR, it will be considered outlier.

Removing outliers:
First we write a function that represent outliers as NA and summarize:

```
> remove_outliers <- function(x) {
+   qnt <- quantile(x, probs=c(.25, .75))
+   checker <- 1.5 * IQR(x)
+   x[x < (qnt[[1]] - checker)] <- NA
+   x[x > (qnt[[2]] + checker)] <- NA
+   x
+ }
> white_wine_11 <- as.data.frame(lapply(white_wine[, -12], remove_outliers))
> nrow(white_wine_11)
[1] 4898
> summary(white_wine_11)
 fixed.acidity volatile.acidity  citric.acid  residual.sugar   chlorides   free.sulfur.dioxide
 Min.   :5     Min.   :0        Min.   :0     Min.   : 0.6   Min.   :0     Min.   : 2
 1st Qu.:6     1st Qu.:0        1st Qu.:0     1st Qu.: 1.7   1st Qu.:0     1st Qu.:23
 Median :7     Median :0        Median :0     Median : 5.2   Median :0     Median :33
 Mean   :7     Mean   :0        Mean   :0     Mean   : 6.4   Mean   :0     Mean   :35
 3rd Qu.:7     3rd Qu.:0        3rd Qu.:0     3rd Qu.: 9.8   3rd Qu.:0     3rd Qu.:45
 Max.   :9     Max.   :0        Max.   :1     Max.   :22.0   Max.   :0     Max.   :80
 NA's   :119   NA's   :186      NA's   :270   NA's   :7      NA's   :208   NA's   :50
 total.sulfur.dioxide    density         pH       sulphates      alcohol
 Min.   : 21          Min.   :1   Min.   :3   Min.   :0     Min.   : 8.0
 1st Qu.:108          1st Qu.:1   1st Qu.:3   1st Qu.:0     1st Qu.: 9.5
 Median :134          Median :1   Median :3   Median :0     Median :10.4
 Mean   :138          Mean   :1   Mean   :3   Mean   :0     Mean   :10.5
 3rd Qu.:167          3rd Qu.:1   3rd Qu.:3   3rd Qu.:1     3rd Qu.:11.4
 Max.   :255          Max.   :1   Max.   :4   Max.   :1     Max.   :14.2
 NA's   :19           NA's   :5   NA's   :75   NA's   :124
>
```

Next I add the quality column and check the result:

```
> white_wine_new <- cbind(white_wine_11, white_wine[ , 12])
> nrow(white_wine_new)
[1] 4898
> summary(white_wine_new)
 fixed.acidity volatile.acidity  citric.acid  residual.sugar   chlorides   free.sulfur.dioxide
 Min.   :5     Min.   :0        Min.   :0     Min.   : 0.6   Min.   :0     Min.   : 2
 1st Qu.:6     1st Qu.:0        1st Qu.:0     1st Qu.: 1.7   1st Qu.:0     1st Qu.:23
 Median :7     Median :0        Median :0     Median : 5.2   Median :0     Median :33
 Mean   :7     Mean   :0        Mean   :0     Mean   : 6.4   Mean   :0     Mean   :35
 3rd Qu.:7     3rd Qu.:0        3rd Qu.:0     3rd Qu.: 9.8   3rd Qu.:0     3rd Qu.:45
 Max.   :9     Max.   :0        Max.   :1     Max.   :22.0   Max.   :0     Max.   :80
 NA's   :119   NA's   :186      NA's   :270   NA's   :7      NA's   :208   NA's   :50
 total.sulfur.dioxide    density         pH       sulphates      alcohol        quality
 Min.   : 21          Min.   :1   Min.   :3   Min.   :0     Min.   : 8.0   Min.   :3.0
 1st Qu.:108          1st Qu.:1   1st Qu.:3   1st Qu.:0     1st Qu.: 9.5   1st Qu.:5.0
 Median :134          Median :1   Median :3   Median :0     Median :10.4   Median :6.0
 Mean   :138          Mean   :1   Mean   :3   Mean   :0     Mean   :10.5   Mean   :5.9
 3rd Qu.:167          3rd Qu.:1   3rd Qu.:3   3rd Qu.:1     3rd Qu.:11.4   3rd Qu.:6.0
 Max.   :255          Max.   :1   Max.   :4   Max.   :1     Max.   :14.2   Max.   :9.0
 NA's   :19           NA's   :5   NA's   :75   NA's   :124
>
```

Finally, we drop the NA's and see the result:

```
> white_wine_newer <- white_wine_new %>% drop_na()
> nrow(white_wine_newer)
[1] 4015
> summary(white_wine_newer)
 fixed.acidity volatile.acidity  citric.acid   residual.sugar   chlorides     free.sulfur.dioxide
 Min.   :4.8   Min.   :0.08     Min.   :0.10   Min.   : 0.6   Min.   :0.015   Min.   : 2
 1st Qu.:6.3   1st Qu.:0.21     1st Qu.:0.27   1st Qu.: 1.8   1st Qu.:0.035   1st Qu.:24
 Median :6.8   Median :0.26     Median :0.31   Median : 5.2   Median :0.042   Median :34
 Mean   :6.8   Mean   :0.26     Mean   :0.32   Mean   : 6.4   Mean   :0.042   Mean   :35
 3rd Qu.:7.3   3rd Qu.:0.31     3rd Qu.:0.37   3rd Qu.: 9.7   3rd Qu.:0.049   3rd Qu.:45
 Max.   :8.8   Max.   :0.48     Max.   :0.57   Max.   :22.0   Max.   :0.071   Max.   :80
 total.sulfur.dioxide    density           pH        sulphates       alcohol        quality
 Min.   : 21          Min.   :0.99   Min.   :2.8   Min.   :0.22   Min.   : 8.4   Min.   :3.0
 1st Qu.:107          1st Qu.:0.99   1st Qu.:3.1   1st Qu.:0.41   1st Qu.: 9.5   1st Qu.:5.0
 Median :132          Median :0.99   Median :3.2   Median :0.47   Median :10.5   Median :6.0
 Mean   :137          Mean   :0.99   Mean   :3.2   Mean   :0.48   Mean   :10.6   Mean   :5.9
 3rd Qu.:166          3rd Qu.:1.00   3rd Qu.:3.3   3rd Qu.:0.54   3rd Qu.:11.4   3rd Qu.:6.0
 Max.   :255          Max.   :1.00   Max.   :3.6   Max.   :0.76   Max.   :14.2   Max.   :9.0
>
```

The application of the whisker outlier removal rule has reduced our dataset from the original 4898 to 4015. Now we are left with 4015 dataset.

Assign the data from column 1-11 (features) to variable white_wine_input, and the class to variable white_wine_output.

```
> white_wine_input <- white_wine_newer[, -12]
> names(white_wine_input)
 [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"          "residual.sugar"
 [5] "chlorides"            "free.sulfur.dioxide"  "total.sulfur.dioxide" "density"
 [9] "pH"                   "sulphates"            "alcohol"
> length(white_wine_input)
[1] 11
> white_wine_output <- as.factor(white_wine_newer$quality)
> str(white_wine_output)
 Factor w/ 7 levels "3","4","5","6",..: 4 4 4 4 4 4 4 4 4 4 ...
>
```

Next is to scale the white_wine_input to have mean = 0, standard deviation = 1 using the built in R scale function and assign the value to white_wine_train. . This will enable us to create a meaningful distance matrix.

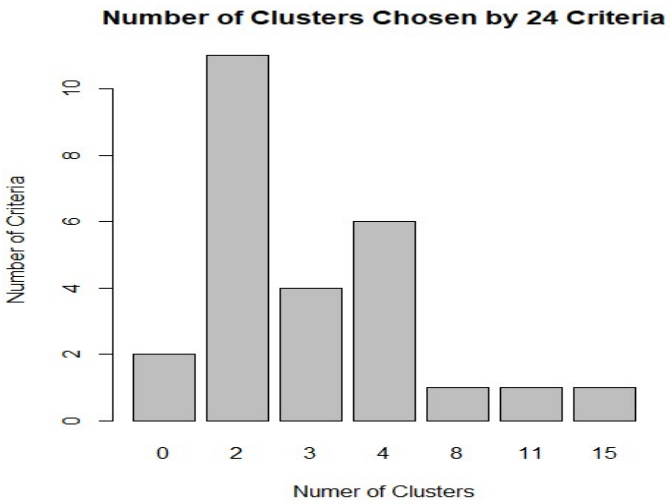
```
> white_wine_train <- scale(white_wine_input)
> summary(white_wine_train)
 fixed.acidity    volatile.acidity  citric.acid     residual.sugar    chlorides      free.sulfur.dioxide
 Min.   :-2.73   Min.   :-2.42    Min.   :-2.60   Min.   :-1.2   Min.   :-2.75   Min.   :-2.18
 1st Qu.:-0.70   1st Qu.:-0.71    1st Qu.:-0.63   1st Qu.:-0.9   1st Qu.:-0.74   1st Qu.:-0.72
 Median :-0.02   Median :-0.05    Median :-0.17   Median :-0.2   Median :-0.03   Median :-0.05
 Mean   : 0.00   Mean   : 0.00    Mean   : 0.00   Mean   : 0.0   Mean   : 0.00   Mean   : 0.00
 3rd Qu.: 0.66   3rd Qu.: 0.60    3rd Qu.: 0.53   3rd Qu.: 0.7   3rd Qu.: 0.67   3rd Qu.: 0.68
 Max.   : 2.70   Max.   : 2.90    Max.   : 2.85   Max.   : 3.2   Max.   : 2.89   Max.   : 3.02
 total.sulfur.dioxide    density           pH          sulphates        alcohol
 Min.   :-2.82        Min.   :-2.34   Min.   :-2.66   Min.   :-2.64   Min.   :-1.80
 1st Qu.:-0.73        1st Qu.:-0.78   1st Qu.:-0.70   1st Qu.:-0.72   1st Qu.:-0.89
 Median :-0.12        Median :-0.11   Median :-0.05   Median :-0.11   Median :-0.07
 Mean   : 0.00        Mean   : 0.00   Mean   : 0.00   Mean   : 0.00   Mean   : 0.00
 3rd Qu.: 0.71        3rd Qu.: 0.71   3rd Qu.: 0.67   3rd Qu.: 0.59   3rd Qu.: 0.67
 Max.   : 2.87        Max.   : 2.81   Max.   : 2.70   Max.   : 2.82   Max.   : 2.98
>
```

As expected, all the variable has mean of zero.

Model Fitting:

To determine the best number of clusters k for K-Means leveraging the NbClust function:

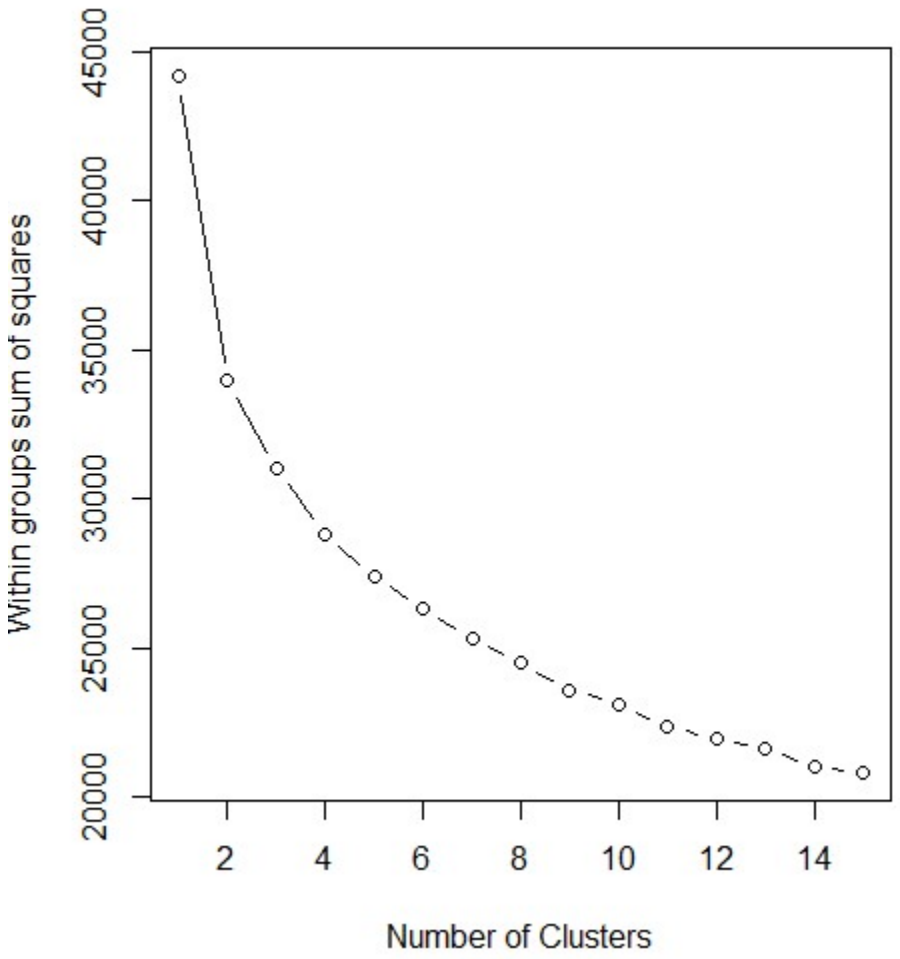
```
> nc <- NbClust(white_wine_train,
+               min.nc=2, max.nc=15,
+               method="kmeans")
*** : The Hubert index is a graphical method of determining the number of clusters.
               In the plot of Hubert index, we seek a significant knee that corresponds to a
               significant increase of the value of the measure i.e the significant peak in Hubert
               index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
               In the plot of D index, we seek a significant knee (the significant peak in Dindex
               second differences plot) that corresponds to a significant increase of the value of
               the measure.

*******************************************************************
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 4 proposed 3 as the best number of clusters
* 6 proposed 4 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 1 proposed 11 as the best number of clusters
* 1 proposed 15 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2


*******************************************************************
>
> table(nc$Best.n[1,])

 0  2  3  4  8 11 15
 2 11  4  6  1  1  1
>
> barplot(table(nc$Best.n[1,]),
+         xlab="Numer of Clusters",
+         ylab="Number of Criteria",
+         main="Number of Clusters Chosen by 24 Criteria")
>
```



According to the graph, the NbClust proposed that the best number of clusters is 2 follow by 4.

Let's consider another method to determine the best number of clusters is using sum of square error, sse:

```
> wss <- 0
> for (i in 1:15){
+   wss[i] <-
+     sum(kmeans(white_wine_train, centers=i)$withinss)
+ }
> plot(1:15,
+      wss,
+      type="b",     ### "b" for both
+      xlab="Number of Clusters",
+      ylab="Within groups sum of squares")
>
```



The elbow of the Within groups sum of squares (wss) indicates k = 2 is the best k. Both two methods suggest k = 2 is best choice for us. Although the original data set contains 7 classes. Therefore, we will investigate from k = 2 to k= 9 and evaluate performance in other to determine the best two clusters.

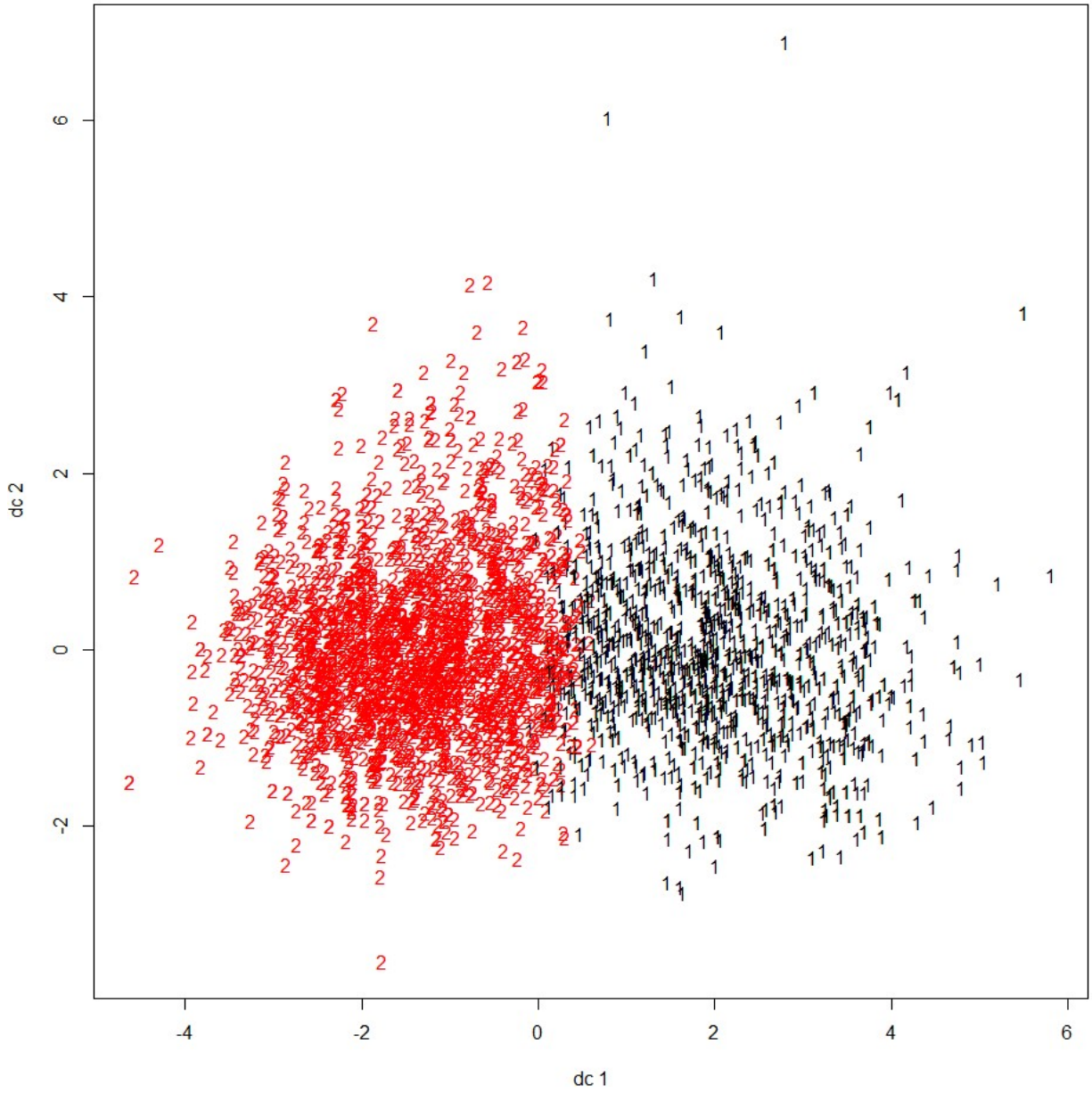
Fit the model

We now fit white_wine data to K-Means with k = 2 and print result:

```
> set.seed(3211)
> fit_km_2 <- kmeans(white_wine_train, 2)
> fit_km_2
K-means clustering with 2 clusters of sizes 1595, 2420

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         0.129            0.044       0.083           0.87      0.62                0.61
2        -0.085           -0.029      -0.055          -0.57     -0.41               -0.40
  total.sulfur.dioxide density     pH sulphates alcohol
1                 0.78    0.99 -0.135     0.120   -0.83
2                -0.52   -0.65  0.089    -0.079    0.55

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   1    2    2    1    1    2    1    1    2    2    2    2    2    2    2    2    1    2    2    2    1
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   2    1    2    2    1    2    2    2    1    2    2    1    1    1    1    1    1    1    1    1    1
  51   52   53   54   56   57   58   59   60   62   64   65   67   69   70   71   72   74   75   76   77
   2    2    2    2    2    1    1    2    2    1    2    2    2    2    1    1    1    2    2    1    2
  78   79   82   83   84   88   92   93   94   95   96   98  101  102  103  104  105  106  107  108  109
   2    1    1    1    1    1    1    2    2    2    1    2    1    2    2    1    1    2    1    1    2
1103 1104 1106 1107 1108 1109 1111 1112 1113 1114 1116 1117 1118 1119 1121 1122 1123 1127 1128 1129 1130
   2    2    2    2    2    2    2    2    1    2    2    2    1    2    1    2    2    2    2    2    2
1131 1132 1133 1134 1135 1136 1137 1138 1144 1145 1146 1149 1150 1151 1152 1155 1156 1157 1158 1160 1162
   1    2    1    2    2    1    2    2    2    2    1    2    1    1    1    1    2    1    1    1    1
1163 1165 1166 1167 1168 1169 1171 1173 1174 1175 1176 1177 1178 1180 1182 1183 1184 1185 1187 1188 1189
   1    2    2    2    2    2    1    2    2    1    1    1    2    2    2    2    2    2    1    2    2
1190 1191 1192 1194 1195 1196 1197 1198 1200 1201 1202 1203 1204 1205 1207 1208 1209 1210 1212 1213 1214
   2    2    2    1    1    1    2    1    2    2    2    1    2    2    1    1    2    2    1    2    2
1216 1217 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1230 1231 1232 1233 1234 1235 1236 1237 1238
   1    2    2    2    2    2    2    1    2    2    2    2    2    1    2    2    2    2    2    2    2
1239 1241 1242 1243 1244 1245 1247 1248 1249 1252 1253 1254 1259
   2    1    2    2    2    2    1    2    2    2    2    2    1
 [ reached getOption("max.print") -- omitted 3015 entries ]

Within cluster sum of squares by cluster:
[1] 12984 20962
 (between_SS / total_SS =  23.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

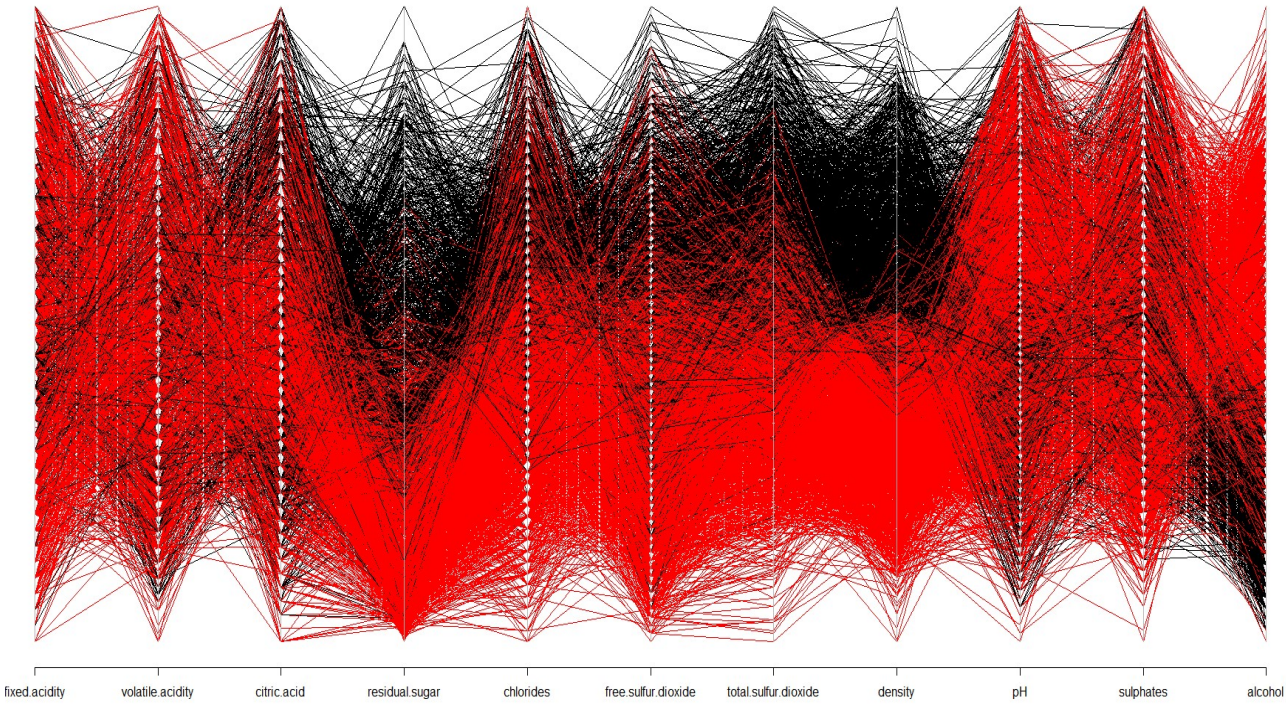
```
> plotcluster(white_wine_train, fit_km_2$cluster)
>
```



We can see the data is fairly clustered, there are few collapse between clusters.

Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km$cluster)
>
```



We can extract some insights from above graph as black cluster contains wine with lower alcohol, higher residual sugar, higher total sulfur dioxide, and higher density than white wine in red cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_2$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           6.9             0.27        0.33           10.7     0.049                  44
2       2           6.8             0.26        0.32            3.6     0.038                  29
  total.sulfur.dioxide density  pH sulphates alcohol
1                  169    1.00 3.2      0.49     9.6
2                  116    0.99 3.2      0.47    11.2
>
```

To check the consistency of fit_km_2 against column 12

```
> confuseTable_km <- table(white_wine_output, fit_km_2$cluster)
> confuseTable_km

white_wine_output    1    2
                3    3    5
                4   24   65
                5  655  451
                6  739 1127
                7  150  647
                8   24  121
                9    0    4
> |
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1 and 2 clusters.

```
> randIndex(confuseTable_km)
   ARI
0.027
> |
```

The agreement between the wine quality and cluster solution is 0.027. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is two. This maybe the possible reason for the low ARI.

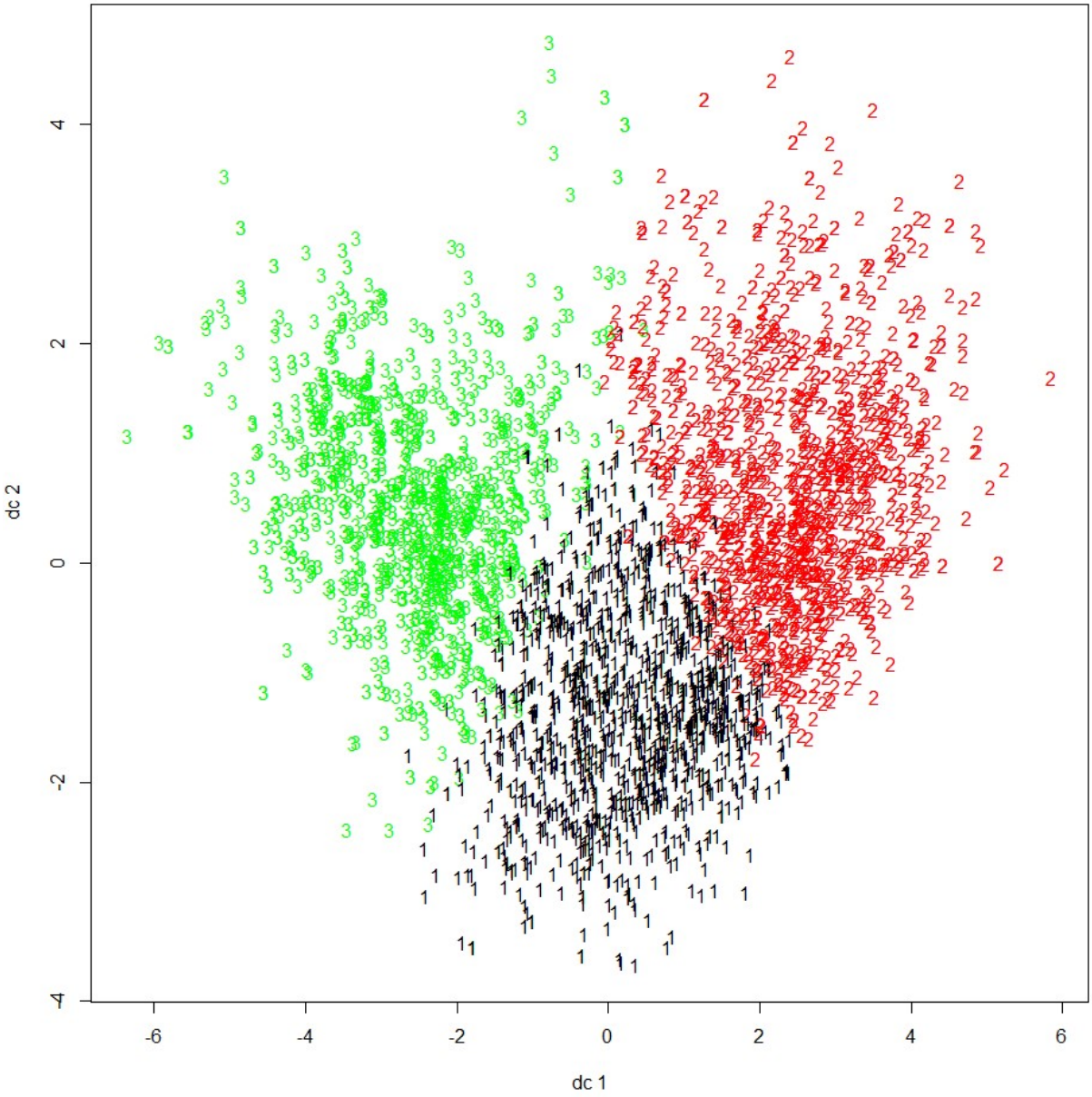
Fit the model for k = 3

We now fit white_wine data to K-Means with k = 3 and print the result:

```
> set.seed(3211)
> fit_km_3 <- kmeans(white_wine_train, 3)
> fit_km_3
K-means clustering with 3 clusters of sizes 1383, 1296, 1336

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         -0.10           -0.379    -0.15103          -0.60      0.24               -0.28
2         -0.05            0.314     0.00025          -0.48     -0.88               -0.43
3          0.15            0.088     0.15610           1.08      0.61                0.71
  total.sulfur.dioxide density    pH sulphates alcohol
1                -0.18   -0.23  0.42     0.303   -0.15
2                -0.69   -0.92 -0.19    -0.377    1.07
3                 0.85    1.13 -0.25     0.051   -0.88

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   3    1    1    3    3    1    1    3    1    1    2    1    1    1    1    2    1    2    1    1    3
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   1    1    2    2    3    1    1    1    3    1    2    3    3    3    1    1    1    3    3    3    1
  51   52   53   54   56   57   58   59   60   62   64   65   67   69   70   71   72   74   75   76   77
   1    1    1    1    1    3    3    2    1    3    2    1    2    1    3    3    3    1    1    3    2

Within cluster sum of squares by cluster:
[1] 10819 10023 10184
 (between_SS / total_SS =  29.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

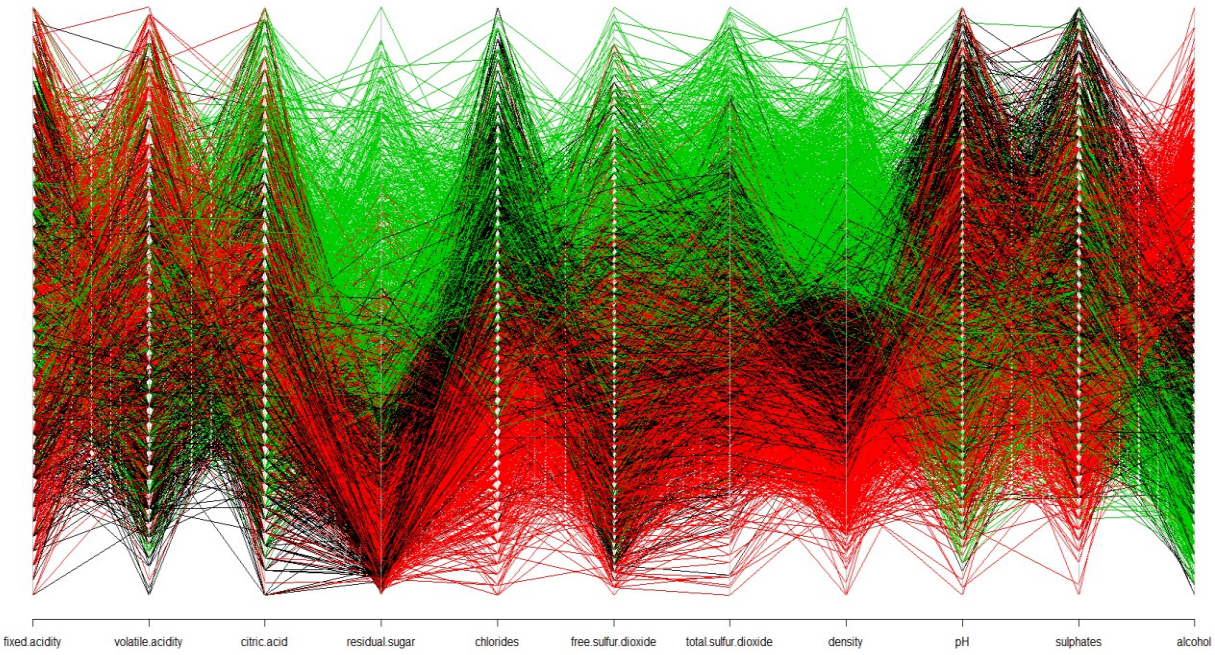
```
> plotcluster(white_wine_train, fit_km_3$cluster)
>
```



We can see the data is fairly clustered, there are few collapse between clusters.

Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km_3$cluster)
>
```



We can extract some insights from above graph as red cluster contains wine with lower density and lower chloride than other coloured cluster. The green cluster contains wine with higher residual sugar, total Sulphur dioxide and density than other coloured cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_3$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           6.7             0.24        0.31            3.4     0.045                  30
2       2           6.8             0.29        0.32            4.0     0.034                  28
3       3           6.9             0.27        0.34           11.7     0.048                  45
  total.sulfur.dioxide density  pH sulphates alcohol
1                  129    0.99 3.2      0.51    10.4
2                  109    0.99 3.2      0.44    11.9
3                  172    1.00 3.2      0.49     9.5
>
```

To check the consistency of fit_km_3 against column 12

```
> confuseTable_km_3 <- table(white_wine_output, fit_km_3$cluster)
> confuseTable_km_3

white_wine_output   1   2   3
                3   2   4   2
                4  38  31  20
                5 383 146 577
                6 693 578 595
                7 228 448 121
                8  39  85  21
                9   0   4   0
>
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1, 2 and 3 clusters.

```
> randIndex(confuseTable_km_3)
   ARI
0.044
>
```

The agreement between the wine quality and cluster solution is 0.044. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is three. This maybe the possible reason for the low ARI.

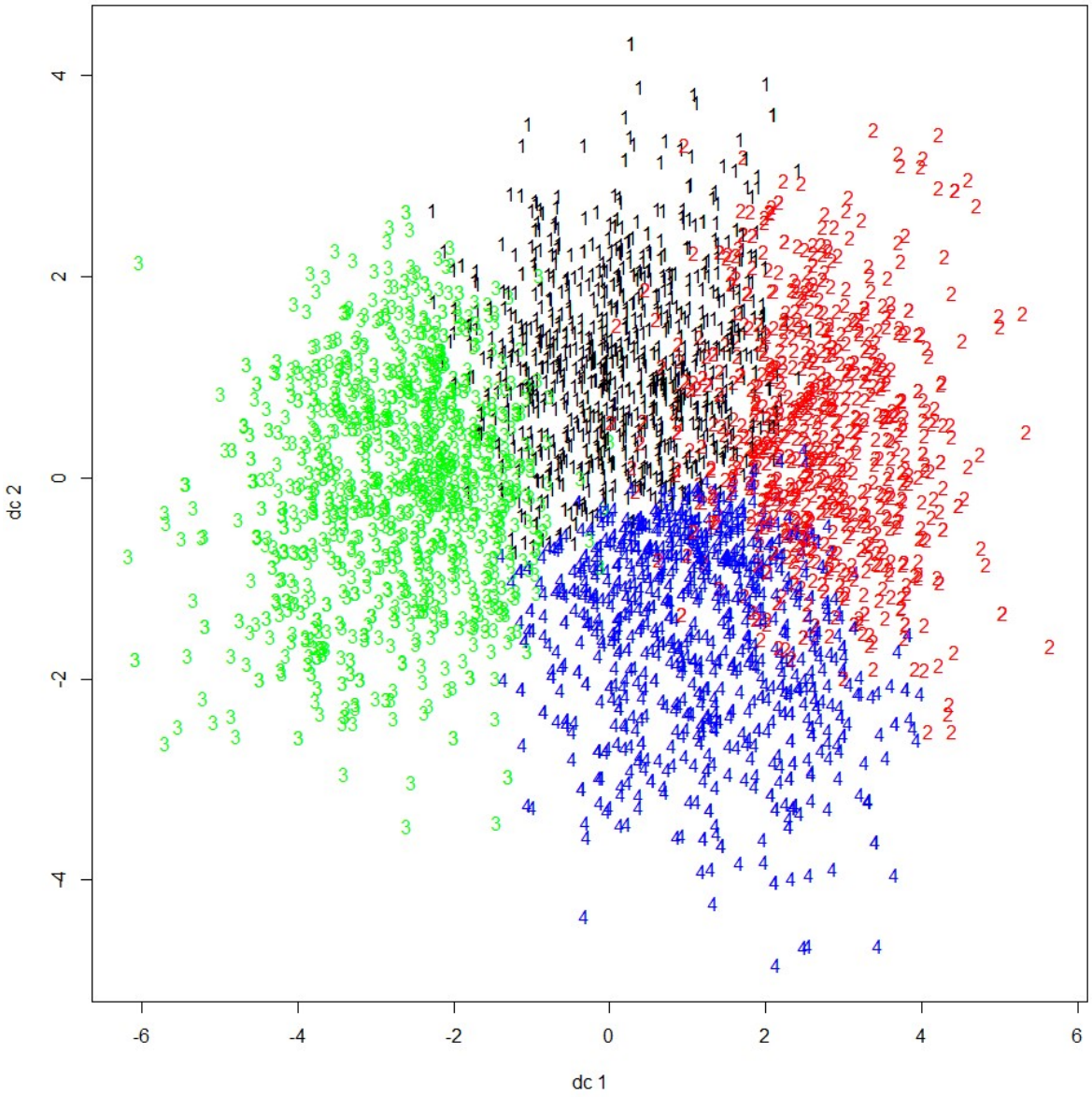
Fit the model for k = 4

We now fit white_wine data to K-Means with k = 4 and print result:

```
> set.seed(3211)
> fit_km_4 <- kmeans(white_wine_train, 4)
> fit_km_4
K-means clustering with 4 clusters of sizes 1017, 826, 1258, 914

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         -0.44           -0.353       -0.34          -0.52      0.24               -0.12
2         -0.69            0.457       -0.20          -0.55     -0.96               -0.39
3          0.13            0.082        0.16           1.11      0.64                0.74
4          0.93           -0.134        0.34          -0.45     -0.29               -0.54
  total.sulfur.dioxide density    pH sulphates alcohol
1               -0.034   -0.18  0.71      0.40   -0.19
2               -0.733   -1.11  0.31     -0.25    1.26
3                0.882    1.16 -0.26      0.07   -0.92
4               -0.514   -0.39 -0.71     -0.32    0.34

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   3    1    4    3    3    4    1    3    1    4    4    4    4    1    2    4    1    2    1    1    3
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   1    1    4    2    4    4    4    1    3    1    4    3    3    3    1    1    1    3    3    3    1
1103 1104 1106 1107 1108 1109 1111 1112 1113 1114 1116 1117 1118 1119 1121 1122 1123 1127 1128 1129 1130
   1    4    2    2    2    2    2    4    3    4    4    2    3    2    1    1    2    2    2    1    4
1131 1132 1133 1134 1135 1136 1137 1138 1144 1145 1146 1149 1150 1151 1152 1155 1156 1157 1158 1160 1162
   1    1    3    4    4    3    2    2    1    1    3    4    3    3    3    1    1    3    3    3    3
1163 1165 1166 1167 1168 1169 1171 1173 1174 1175 1176 1177 1178 1180 1182 1183 1184 1185 1187 1188 1189
   1    4    1    2    4    4    1    4    1    3    3    3    1    1    4    1    2    2    4    2    4
1190 1191 1192 1194 1195 1196 1197 1198 1200 1201 1202 1203 1204 1205 1207 1208 1209 1210 1212 1213 1214
   4    4    1    3    1    3    1    3    4    2    4    3    1    1    3    3    1    4    3    4    1
1216 1217 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1230 1231 1232 1233 1234 1235 1236 1237 1238
   3    4    4    4    1    2    4    3    2    2    2    4    4    3    4    4    4    1    4    2    4
1239 1241 1242 1243 1244 1245 1247 1248 1249 1252 1253 1254 1259
   2    1    1    4    2    1    4    4    4    1    1    4    3
 [ reached getOption("max.print") -- omitted 3015 entries ]

Within cluster sum of squares by cluster:
[1] 7387 5317 9318 6816
 (between_SS / total_SS =  34.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
>
```

Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

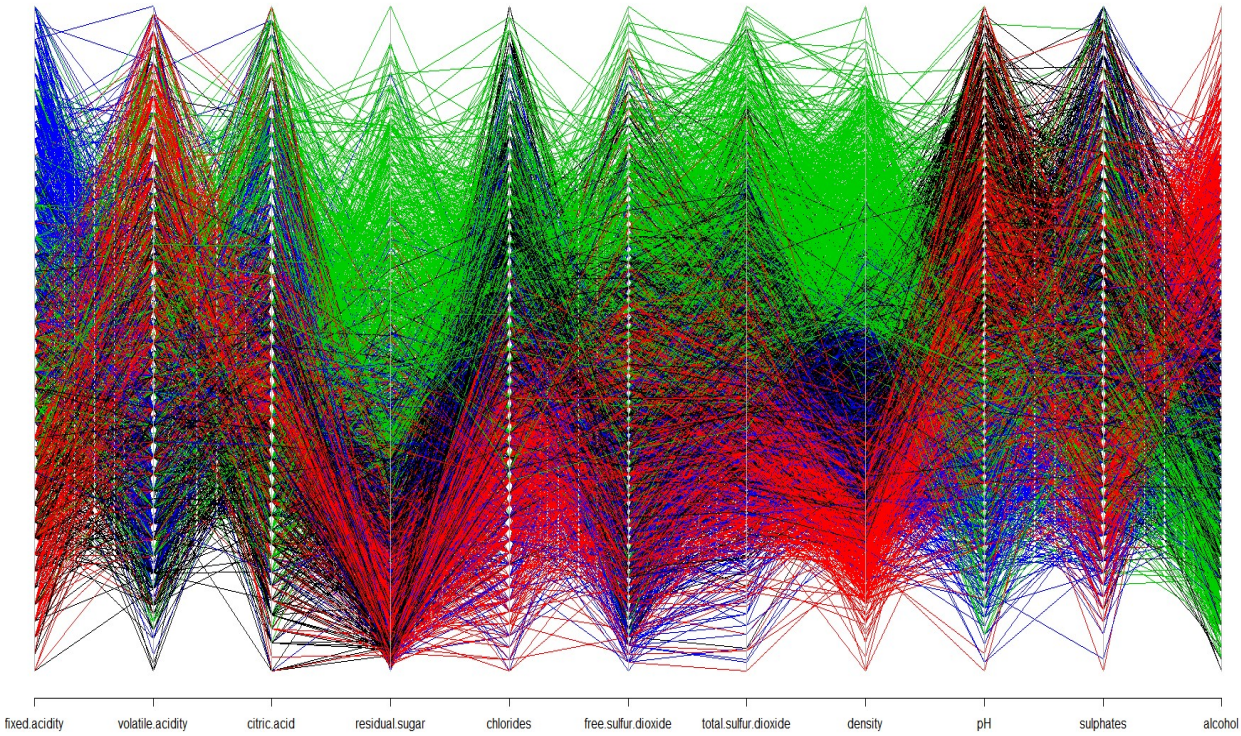
```
> plotcluster(white_wine_train, fit_km_4$cluster)
>
```



We can see the data is fairly clustered, there are few collapse between clusters.

Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km_4$cluster)
>
```



We can extract some insights from above graph as red cluster contains wine with lower density than other coloured cluster, green cluster contains wine with higher residual sugar, lower total sulphur dioxide and lower density than other coloured cluster; blue cluster contains wine with higher fixed acidity than other coloured cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_4$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           6.5              0.24        0.30           3.8     0.045                  33
2       2           6.3              0.30        0.31           3.6     0.033                  29
3       3           6.9              0.27        0.34          11.9     0.049                  46
4       4           7.5              0.25        0.35           4.2     0.039                  27
  total.sulfur.dioxide density  pH sulphates alcohol
1                  136    0.99 3.3      0.52    10.4
2                  107    0.99 3.2      0.46    12.1
3                  173    1.00 3.2      0.49     9.5
4                  116    0.99 3.1      0.45    11.0
>
```

To check the consistency of fit_km_4 against column 12

```
> confuseTable_km_4 <- table(white_wine_output, fit_km_4$cluster)
> confuseTable_km_4

white_wine_output   1   2   3   4
                3   2   1   2   3
                4  24  11  19  35
                5 253  55 557 241
                6 527 354 548 437
                7 182 335 112 168
                8  29  66  20  30
                9   0   4   0   0
>
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1, 2, 3 and 4 clusters.

```
> randIndex(confuseTable_km_4)
   ARI
0.043
>
```

The agreement between the wine quality and cluster solution is 0.043. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is four. This maybe the possible reason for the low ARI.

Fit the model for k = 5

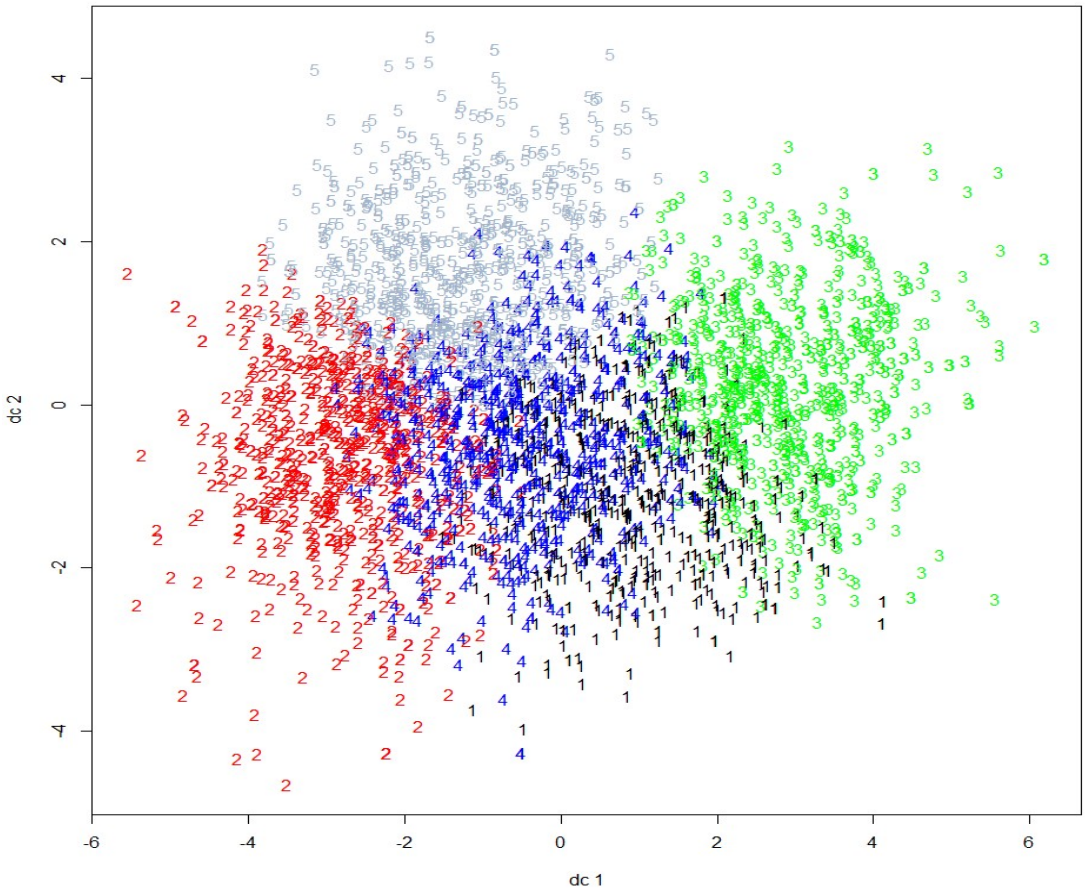
We now fit white_wine data to K-Means with k = 5 and print result:

```
> set.seed(3211)
> fit_km_5 <- kmeans(white_wine_train, 5)
> fit_km_5
K-means clustering with 5 clusters of sizes 699, 776, 956, 796, 788

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         -0.34         0.356745      -0.936          0.37     0.536              -0.167
2         -0.75         0.535618      -0.251         -0.54    -0.940              -0.409
3          0.27        -0.000097       0.467          1.15     0.648               0.897
4         -0.26        -0.612299       0.033         -0.67     0.078              -0.043
5          0.97        -0.225280       0.477         -0.51    -0.414              -0.493
  total.sulfur.dioxide density     pH sulphates alcohol
1                0.036    0.48 -0.066     -0.27  -0.651
2               -0.723   -1.11  0.368     -0.30   1.269
3                1.052    1.23 -0.329      0.14  -0.957
4               -0.034   -0.31  0.830      0.70   0.015
5               -0.562   -0.51 -0.743     -0.35   0.474

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   3    4    5    3    3    5    1    3    4    5    5    5    5    4    4    5    1    2    4    4    3
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   4    4    5    5    3    5    5    4    1    1    5    3    3    1    4    4    4    1    1    1    4
  51   52   53   54   56   57   58   59   60   62   64   65   67   69   70   71   72   74   75   76   77
   4    4    4    4    4    3    1    1    4    1    1    5    2    4    1    3    3    5    4    1    5
```

```
1049 1050 1051 1053 1055 1056 1057 1059 1061 1062 1066 1067 1068 1069 1070 1071 1072 1073 1074 1076 1077
   5    2    3    5    5    5    4    4    5    3    3    3    5    5    4    5    3    5    1    2    3
1078 1079 1080 1081 1082 1083 1084 1085 1087 1088 1089 1090 1092 1093 1094 1095 1097 1098 1099 1101 1102
   4    3    1    4    1    1    3    1    2    5    3    5    3    4    5    3    1    1    1    1    2
1103 1104 1106 1107 1108 1109 1111 1112 1113 1114 1116 1117 1118 1119 1121 1122 1123 1127 1128 1129 1130
   4    5    2    2    2    2    2    5    3    5    5    2    3    2    4    1    2    2    2    4    5
1131 1132 1133 1134 1135 1136 1137 1138 1144 1145 1146 1149 1150 1151 1152 1155 1156 1157 1158 1160 1162
   1    4    3    5    1    3    2    2    4    4    3    5    3    3    3    4    4    3    3    3    3
1163 1165 1166 1167 1168 1169 1171 1173 1174 1175 1176 1177 1178 1180 1182 1183 1184 1185 1187 1188 1189
   4    5    4    2    4    5    4    5    4    3    3    3    4    1    5    4    2    2    5    2    5
1190 1191 1192 1194 1195 1196 1197 1198 1200 1201 1202 1203 1204 1205 1207 1208 1209 1210 1212 1213 1214
   5    5    4    3    1    3    4    3    5    4    5    3    2    2    3    3    4    5    3    5    2
1216 1217 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1230 1231 1232 1233 1234 1235 1236 1237 1238
   3    5    5    5    4    2    4    3    5    2    5    5    5    3    5    5    2    4    5    2    5
1239 1241 1242 1243 1244 1245 1247 1248 1249 1252 1253 1254 1259
   2    4    4    5    2    1    3    5    5    1    1    5    1
 [ reached getOption("max.print") -- omitted 3015 entries ]

Within cluster sum of squares by cluster:
[1] 4864 4921 6743 5360 5515
 (between_SS / total_SS =  37.9 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

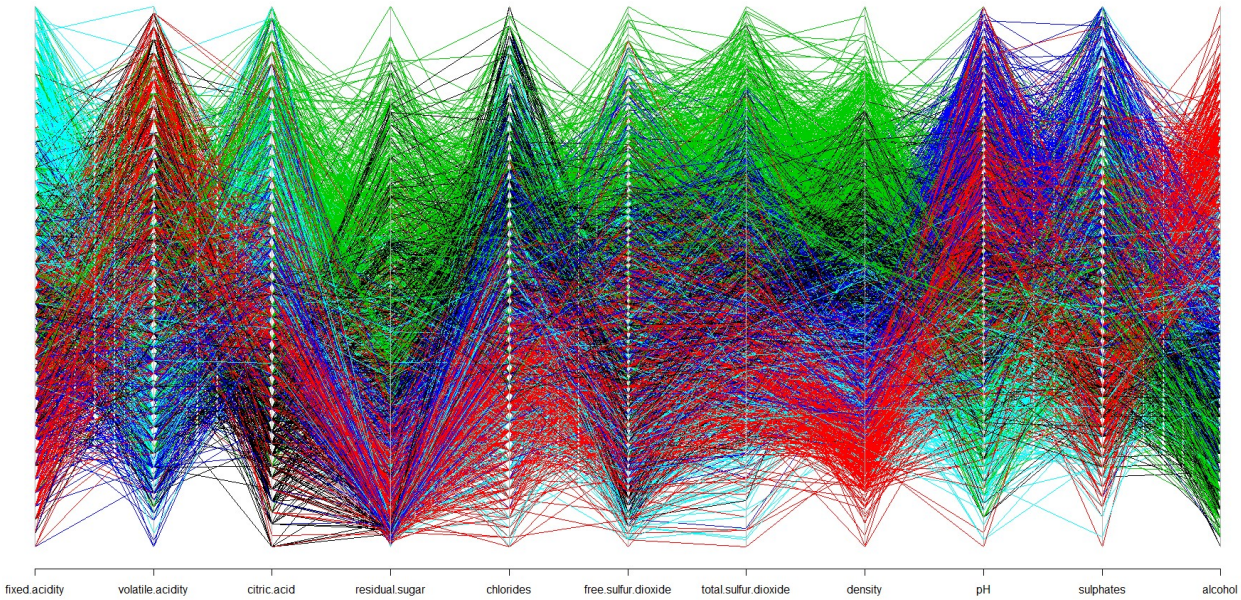
```
> plotcluster(white_wine_train, fit_km_5$cluster)
> |
```



We can see the data is fairly clustered, there are collapse between clusters.

Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km_5$cluster)
```



We can extract some insights from above graph as red cluster contains wine with lower density and higher alcohol than other coloured cluster; green cluster contains wine with higher residual sugar, higher free Sulphur dioxide, higher total Sulphur dioxide and higher density than other coloured cluster; blue cluster contains wine with higher PH and higher sulphate than other coloured cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_5$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           6.6             0.29        0.24            8.2     0.048                  32
2       2           6.3             0.31        0.30            3.7     0.033                  29
3       3           7.0             0.26        0.36           12.0     0.049                  48
4       4           6.6             0.22        0.33            3.1     0.043                  34
5       5           7.5             0.25        0.37            3.8     0.038                  27
  total.sulfur.dioxide density  pH sulphates alcohol
1                  138    1.00 3.2      0.45     9.8
2                  107    0.99 3.2      0.45    12.1
3                  180    1.00 3.1      0.50     9.4
4                  136    0.99 3.3      0.55    10.6
5                  114    0.99 3.1      0.45    11.2
> |
```

To check the consistency of fit_km_5 against column 12

```
> confuseTable_km_5 <- table(white_wine_output, fit_km_5$cluster)
> confuseTable_km_5

white_wine_output   1   2   3   4   5
                3   1   2   2   1   2
                4  25  10  12  13  29
                5 315  53 409 144 185
                6 323 333 407 426 377
                7  23 317 108 184 165
                8  12  57  18  28  30
                9   0   4   0   0   0
> |
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1, 2, 3, 4 and 5 clusters.

```
> randIndex(confuseTable_km_5)
   ARI
0.038
>
```

The agreement between the wine quality and cluster solution is 0.038. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is five. This maybe the possible reason for the low ARI.

Fit the model for k = 6

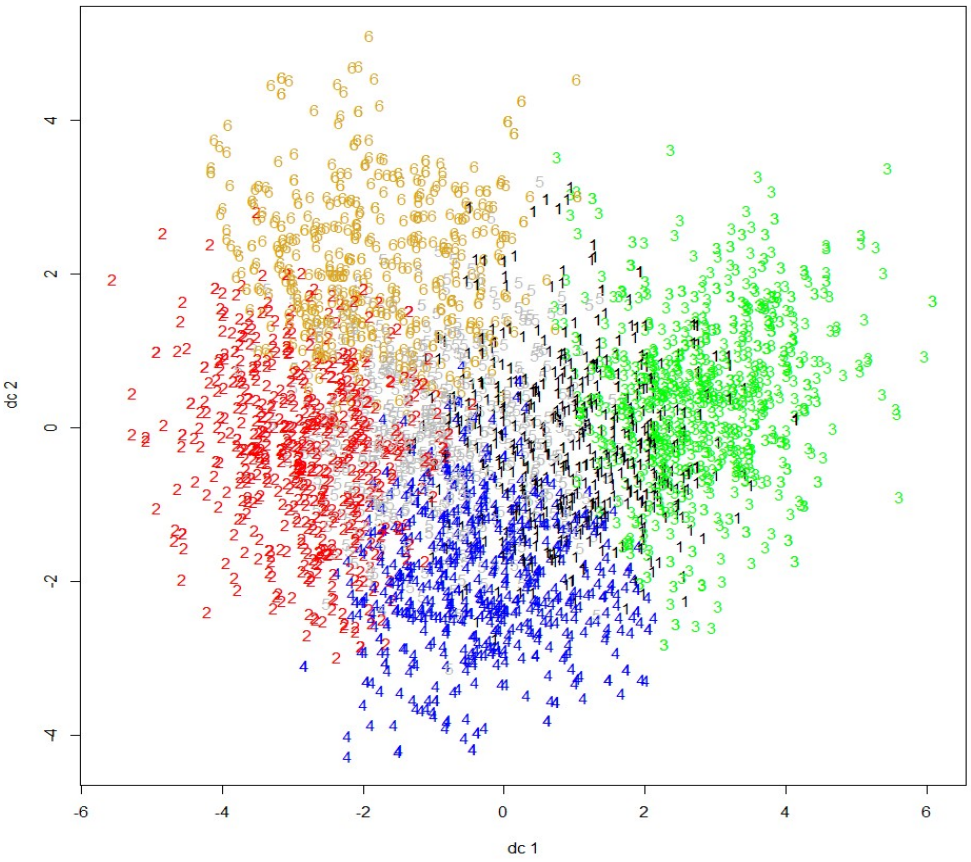
We now fit white_wine data to K-Means with k = 6 and print result:

```
> set.seed(3211)
> fit_km_6 <- kmeans(white_wine_train, 6)
> fit_km_6
K-means clustering with 6 clusters of sizes 655, 643, 944, 653, 619, 501

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         -0.35            0.270      -0.971           0.40    0.5459              -0.180
2         -0.94            0.317      -0.247          -0.63   -0.9389              -0.396
3          0.27            0.031       0.487           1.16    0.6462               0.905
4         -0.35           -0.523      -0.124          -0.61    0.1127               0.093
5          0.72           -0.572       0.653          -0.73    0.0019              -0.618
6          0.73            0.570       0.024          -0.19   -0.8755              -0.319
  total.sulfur.dioxide density     pH sulphates alcohol
1              -0.0056    0.51 -0.076     -0.28  -0.697
2              -0.7464   -1.15  0.603     -0.16   1.222
3               1.0522    1.23 -0.340      0.13  -0.965
4               0.1628   -0.22  1.042      0.61  -0.037
5              -0.5835   -0.47 -0.476      0.37   0.211
6              -0.5085   -0.64 -0.805     -0.93   0.948

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   3    4    5    3    3    5    1    3    4    5    5    5    5    4    5    5    1    6    4    4    3
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   4    4    5    5    5    5    5    4    1    1    6    3    3    1    4    4    4    1    1    1    4
  51   52   53   54   56   57   58   59   60   62   64   65   67   69   70   71   72   74   75   76   77
   4    4    4    4    5    3    1    1    4    1    1    5    2    4    1    3    3    5    4    1    2
  78   79   82   83   84   88   92   93   94   95   96   98  101  102  103  104  105  106  107  108  109
   2    1    3    1    3    3    3    2    2    5    3    6    3    3    1    3    3    1    3    3    3
```

```
 996  997  998  999 1001 1002 1003 1004 1005 1006 1007 1009 1010 1011 1012 1013 1014 1016 1018 1019 1020
   5    5    1    3    5    3    3    4    2    4    2    4    5    5    4    5    3    6    5    5    4
1021 1022 1023 1024 1026 1027 1029 1031 1032 1033 1034 1036 1038 1039 1040 1042 1044 1045 1046 1047 1048
   5    5    6    3    4    4    4    4    4    3    3    5    1    4    5    5    3    5    5    1    5
1049 1050 1051 1053 1055 1056 1057 1059 1061 1062 1066 1067 1068 1069 1070 1071 1072 1073 1074 1076 1077
   6    2    3    6    5    5    4    4    5    3    3    3    5    6    4    6    3    6    1    2    3
1078 1079 1080 1081 1082 1083 1084 1085 1087 1088 1089 1090 1092 1093 1094 1095 1097 1098 1099 1101 1102
   5    3    1    4    1    1    3    1    2    6    3    5    3    4    6    3    1    1    1    1    2
1103 1104 1106 1107 1108 1109 1111 1112 1113 1114 1116 1117 1118 1119 1121 1122 1123 1127 1128 1129 1130
   4    6    2    2    2    2    2    5    3    5    5    2    3    2    4    1    2    6    2    4    6
1131 1132 1133 1134 1135 1136 1137 1138 1144 1145 1146 1149 1150 1151 1152 1155 1156 1157 1158 1160 1162
   4    5    3    5    6    3    2    4    4    4    3    5    3    3    3    4    4    3    3    3    3
1163 1165 1166 1167 1168 1169 1171 1173 1174 1175 1176 1177 1178 1180 1182 1183 1184 1185 1187 1188 1189
   5    5    4    2    5    5    5    5    4    3    3    3    4    1    6    4    2    6    5    2    6
1190 1191 1192 1194 1195 1196 1197 1198 1200 1201 1202 1203 1204 1205 1207 1208 1209 1210 1212 1213 1214
   5    5    4    3    1    3    4    3    6    2    5    3    2    2    3    3    4    5    3    5    2
1216 1217 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1230 1231 1232 1233 1234 1235 1236 1237 1238
   3    6    6    6    4    2    5    3    5    2    5    5    6    3    6    6    6    4    6    2    5
1239 1241 1242 1243 1244 1245 1247 1248 1249 1252 1253 1254 1259
   6    4    4    6    6    1    3    5    5    1    1    6    1
 [ reached getOption("max.print") -- omitted 3015 entries ]

Within cluster sum of squares by cluster:
[1] 4469 3886 6620 4264 3967 3214
 (between_SS / total_SS =  40.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

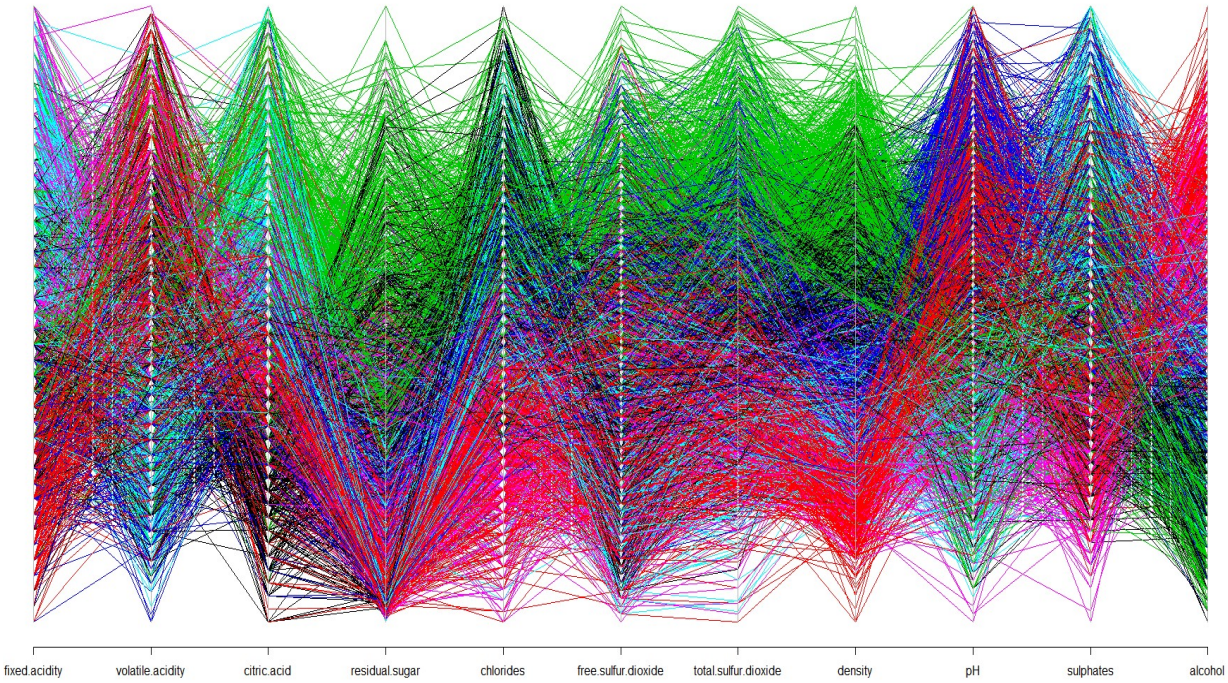
Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

```
> plotcluster(white_wine_train, fit_km_6$cluster)
> |
```



We can see the data is fairly clustered, there are collapse between clusters.
Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km_6$cluster)
>
```



We can extract some insights from above graph as red cluster contains wine with lower fixed acidity, lower chloride, lower density, lower sulphate than other coloured cluster; green cluster contains wine with higher citric acid, higher residual sugar, higher free Sulphur dioxide, higher total Sulphur dioxide, and higher density than other coloured cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_6$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           6.6             0.28        0.24            8.3     0.048                  32
2       2           6.1             0.29        0.30            3.3     0.033                  29
3       3           7.0             0.27        0.37           12.1     0.049                  48
4       4           6.6             0.22        0.31            3.4     0.043                  36
5       5           7.3             0.22        0.38            2.8     0.042                  25
6       6           7.4             0.31        0.33            5.4     0.034                  30
  total.sulfur.dioxide density  pH sulphates alcohol
1                  137    1.00 3.2      0.45     9.7
2                  106    0.99 3.3      0.47    12.1
3                  180    1.00 3.1      0.49     9.4
4                  144    0.99 3.3      0.54    10.5
5                  113    0.99 3.1      0.52    10.8
6                  116    0.99 3.1      0.39    11.7
>
```

To check the consistency of fit_km_6 against column 12

```
> confuseTable_km_6 <- table(white_wine_output, fit_km_6$cluster)
> confuseTable_km_6

white_wine_output   1   2   3   4   5   6
                3   1   1   2   1   0   3
                4  25  10  12  13  17  12
                5 298  46 407 121 162  72
                6 300 279 399 344 323 221
                7  23 259 106 152 103 154
                8   8  44  18  22  14  39
                9   0   4   0   0   0   0
>
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1, 2, 3, 4, 5 and 6 clusters.

```
> randIndex(confuseTable_km_6)
  ARI
0.035
>
```

The agreement between the wine quality and cluster solution is 0.035. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is six. This maybe the possible reason for the low ARI.

Fit the model for k = 7

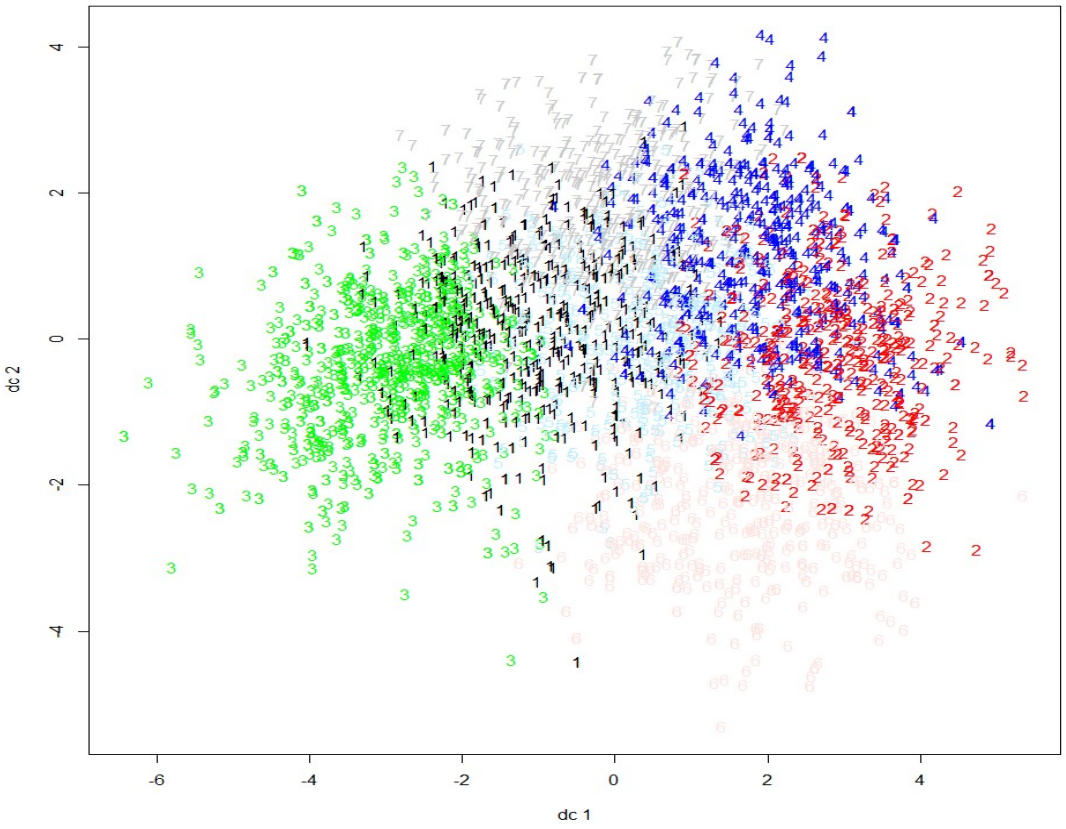
We now fit white_wine data to K-Means with k = 7 and interpret result:

```
> set.seed(3211)
> fit_km_7 <- kmeans(white_wine_train, 7)
> fit_km_7
K-means clustering with 7 clusters of sizes 604, 477, 891, 535, 502, 497, 509

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         -0.35            0.412      -0.988          0.42      0.61               -0.17
2         -0.94            0.822      -0.372         -0.57     -0.88               -0.30
3          0.28            0.004       0.474          1.21      0.63                0.92
4         -0.65           -0.962      -0.079         -0.70     -0.58               -0.42
5          0.98           -0.467       0.679         -0.71      0.21               -0.59
6          0.69            0.477       0.117         -0.20     -0.95               -0.37
7         -0.16           -0.261      -0.008         -0.45      0.34                0.27
  total.sulfur.dioxide density     pH sulphates alcohol
1                0.024    0.55 -0.057    -0.254   -0.73
2               -0.614   -1.16  0.605    -0.064    1.35
3                1.032    1.26 -0.418     0.063   -1.00
4               -0.820   -0.71  0.572    -0.056    0.37
5               -0.482   -0.36 -0.559     0.364    0.12
6               -0.530   -0.66 -0.813    -0.914    0.95
7                0.595   -0.02  0.975     0.844   -0.10

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   3    7    5    3    3    5    1    3    7    5    5    5    5    7    4    5    1    6    7    7    3
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   7    7    5    5    5    4    5    7    1    1    6    3    3    1    7    7    7    1    1    1    7
  51   52   53   54   56   57   58   59   60   62   64   65   67   69   70   71   72   74   75   76   77
   7    7    4    4    5    3    1    1    4    1    1    5    2    7    1    3    7    5    7    1    4
  78   79   82   83   84   88   92   93   94   95   96   98  101  102  103  104  105  106  107  108  109
   2    5    3    1    3    3    3    2    4    5    3    6    3    3    1    3    3    3    3    3    3
```

```
 972  973  974  976  977  978  979  982  983  984  985  986  987  988  989  990  991  992  993  994  995
   2    3    4    1    7    3    7    7    2    2    7    2    1    1    7    6    5    5    3    4    5
 996  997  998  999 1001 1002 1003 1004 1005 1006 1007 1009 1010 1011 1012 1013 1014 1016 1018 1019 1020
   5    4    1    3    5    3    3    5    2    4    2    7    5    5    7    5    3    6    5    5    7
1021 1022 1023 1024 1026 1027 1029 1031 1032 1033 1034 1036 1038 1039 1040 1042 1044 1045 1046 1047 1048
   5    5    6    3    4    7    7    7    7    7    7    5    1    4    5    5    3    5    5    1    6
1049 1050 1051 1053 1055 1056 1057 1059 1061 1062 1066 1067 1068 1069 1070 1071 1072 1073 1074 1076 1077
   5    2    3    6    6    5    7    7    5    7    3    3    5    6    7    5    3    6    1    2    7
1078 1079 1080 1081 1082 1083 1084 1085 1087 1088 1089 1090 1092 1093 1094 1095 1097 1098 1099 1101 1102
   5    3    7    7    1    1    3    4    4    4    3    4    3    7    6    3    1    4    1    1    4
1103 1104 1106 1107 1108 1109 1111 1112 1113 1114 1116 1117 1118 1119 1121 1122 1123 1127 1128 1129 1130
   7    6    2    4    2    2    2    5    3    5    5    2    3    2    7    1    4    4    4    4    6
1131 1132 1133 1134 1135 1136 1137 1138 1144 1145 1146 1149 1150 1151 1152 1155 1156 1157 1158 1160 1162
   7    5    3    5    1    3    2    7    5    5    3    4    3    3    3    7    7    3    3    3    3
1163 1165 1166 1167 1168 1169 1171 1173 1174 1175 1176 1177 1178 1180 1182 1183 1184 1185 1187 1188 1189
   7    5    4    2    5    4    5    5    7    3    3    3    7    1    6    7    4    6    5    4    6
1190 1191 1192 1194 1195 1196 1197 1198 1200 1201 1202 1203 1204 1205 1207 1208 1209 1210 1212 1213 1214
   5    5    7    3    7    3    7    3    6    4    4    3    2    2    3    3    4    6    3    5    2
1216 1217 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1230 1231 1232 1233 1234 1235 1236 1237 1238
   3    6    6    6    7    2    5    3    4    2    4    5    6    3    6    6    6    7    6    2    6
1239 1241 1242 1243 1244 1245 1247 1248 1249 1252 1253 1254 1259
   6    7    7    6    6    1    5    4    4    1    1    6    1
 [ reached getOption("max.print") -- omitted 3015 entries ]

Within cluster sum of squares by cluster:
[1] 4074 2792 6133 2930 3191 3154 3326
 (between_SS / total_SS =  42.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

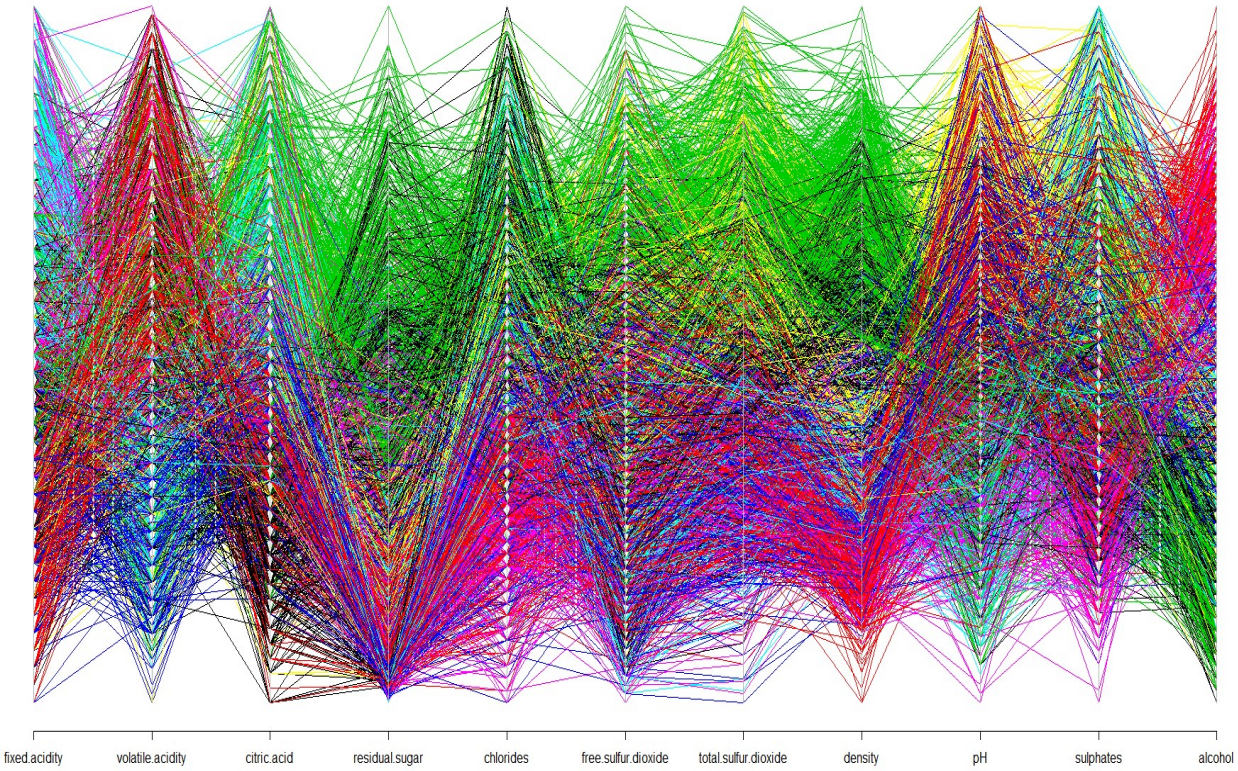
```
> plotcluster(white_wine_train, fit_km_7$cluster)
> |
```



We can see the data is fairly clustered, there are collapse between clusters.

Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km_7$cluster)
>
```



We can extract some insights from above graph as red cluster contains wine with lower density, lower chloride, higher alcohol and higher volatile acidity than other coloured cluster; green cluster contains wine with higher citric acid, higher residual sugar, higher free Sulphur dioxide, higher total Sulphur dioxide and higher density than other coloured cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_7$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           6.6             0.30        0.24            8.4     0.048                  32
2       2           6.1             0.33        0.29            3.5     0.034                  30
3       3           7.0             0.26        0.37           12.3     0.049                  49
4       4           6.3             0.19        0.32            2.9     0.037                  28
5       5           7.5             0.23        0.38            2.9     0.044                  26
6       6           7.3             0.30        0.33            5.4     0.033                  29
7       7           6.7             0.24        0.32            4.2     0.046                  39
  total.sulfur.dioxide density  pH sulphates alcohol
1                  138    1.00 3.2      0.46     9.7
2                  112    0.99 3.3      0.47    12.2
3                  179    1.00 3.1      0.49     9.4
4                  103    0.99 3.3      0.48    11.0
5                  117    0.99 3.1      0.52    10.7
6                  115    0.99 3.1      0.39    11.7
7                  161    0.99 3.3      0.56    10.5
>
```

To check the consistency of fit_km_7 against column 12

```
> confuseTable_km_7 <- table(white_wine_output, fit_km_7$cluster)
> confuseTable_km_7

white_wine_output   1   2   3   4   5   6   7
                3   1   0   2   2   0   3   0
                4  24   7  11   6  16  13  12
                5 297  23 382  81 153  70 100
                6 265 193 379 269 257 228 275
                7  12 217  99 154  71 142 102
                8   5  34  18  23   5  40  20
                9   0   3   0   0   0   1   0
>
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1 to 7 clusters.

```
> randIndex(confuseTable_km_7)
  ARI
0.035
>
```

The agreement between the wine quality and cluster solution is 0.035. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is seven. This maybe the possible reason for the low ARI.

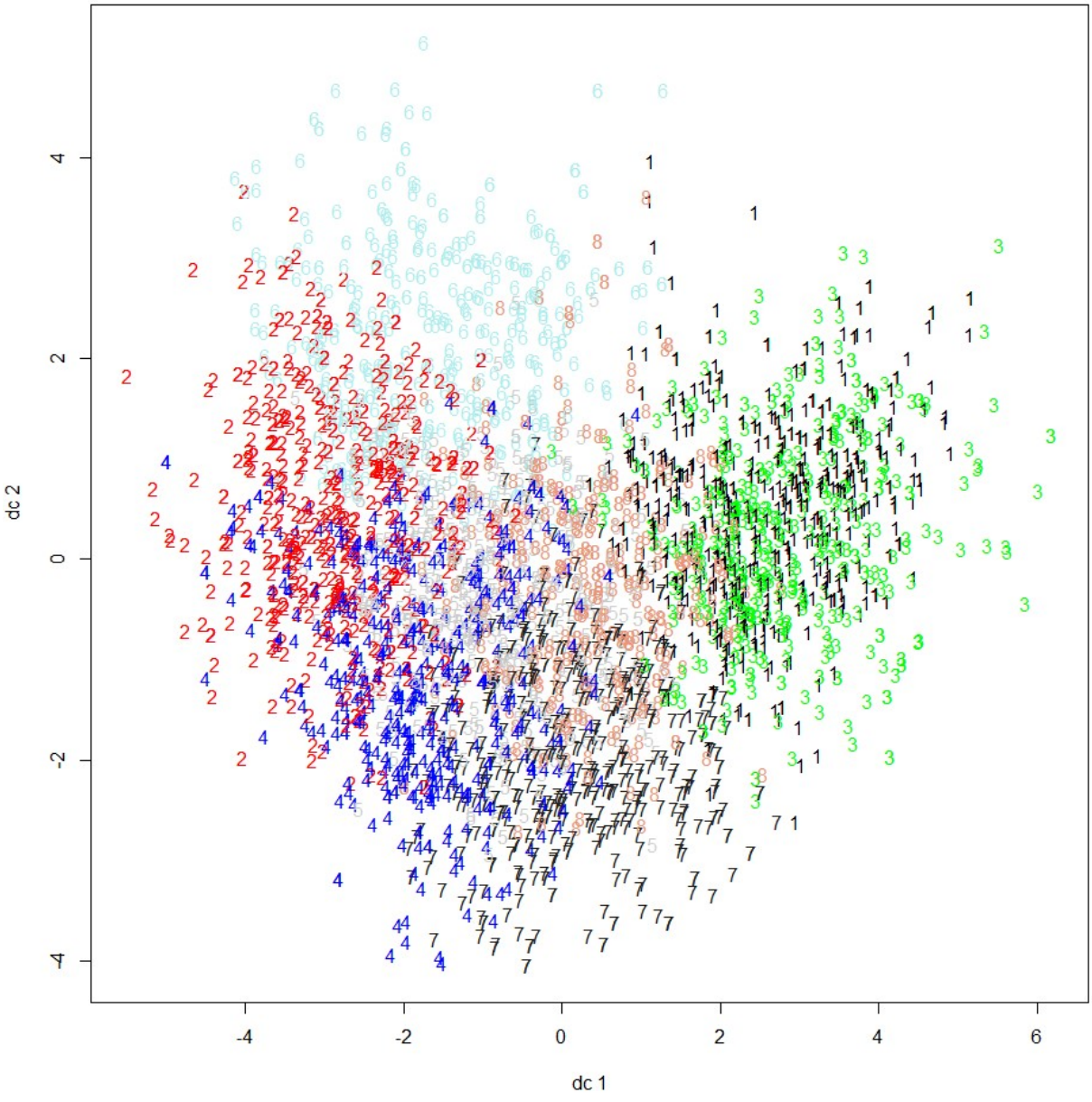
Fit the model for k = 8

We now fit white_wine data to K-Means with k = 8 and interpret result:

```
> set.seed(3211)
> fit_km_8 <- kmeans(white_wine_train, 8)
> fit_km_8
K-means clustering with 8 clusters of sizes 693, 515, 483, 508, 511, 430, 492, 383

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         0.097            -0.26      -0.474          1.361      0.68                0.50
2        -0.768             0.88      -0.260         -0.552     -0.95               -0.31
3         0.158             0.38       1.180          0.864      0.55                1.14
4        -0.733            -0.94      -0.119         -0.684     -0.59               -0.43
5         0.885            -0.48       0.702         -0.735      0.11               -0.59
6         0.873             0.32       0.093         -0.095     -0.89               -0.32
7        -0.158            -0.39      -0.118         -0.403      0.25                0.25
8        -0.328             0.85      -1.012         -0.297      0.67               -0.53
  total.sulfur.dioxide density     pH sulphates alcohol
1                0.547  1.2838 -0.304     -0.23   -0.96
2               -0.637 -1.1814  0.422     -0.11    1.42
3                1.253  1.0383 -0.336      0.38   -0.94
4               -0.829 -0.7355  0.588     -0.17    0.39
5               -0.507 -0.4364 -0.518      0.48    0.22
6               -0.515 -0.5188 -0.928     -1.03    0.82
7                0.539  0.0016  1.051      0.79   -0.10
8               -0.052  0.0945  0.011     -0.19   -0.60

Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   1    8    5    1    1    5    8    1    8    5    5    5    5    7    4    5    8    6    7    7    3
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   7    7    5    5    5    4    5    7    1    8    6    1    1    8    7    7    7    8    8    8    7
  51   52   53   54   56   57   58   59   60   62   64   65   67   69   70   71   72   74   75   76   77
   7    7    4    4    5    1    1    8    4    1    8    5    2    7    1    3    7    5    7    1    4
  78   79   82   83   84   88   92   93   94   95   96   98  101  102  103  104  105  106  107  108  109
   2    1    1    8    1    3    3    2    4    5    3    6    3    1    1    1    3    1    1    3    3
 110  112  113  114  115  117  118  119  120  122  123  124  126  127  128  129  130  131  132  133  134
   7    3    3    8    8    2    4    3    3    1    1    8    4    6    7    7    7    4    7    3    1
 920  921  924  925  926  928  931  932  933  934  935  936  937  939  940  941  942  943  944  945  946
   6    7    1    7    8    3    6    3    3    3    3    3    2    7    1    7    3    8    2    1    7
 948  950  951  952  953  954  955  956  957  958  959  960  961  962  963  964  965  966  967  968  969
   5    5    8    1    4    6    1    8    8    4    4    1    1    6    1    4    1    6    8    1    4
 972  973  974  976  977  978  979  982  983  984  985  986  987  988  989  990  991  992  993  994  995
   2    1    4    8    7    1    7    7    2    2    7    8    8    8    7    6    5    5    1    8    5
 996  997  998  999 1001 1002 1003 1004 1005 1006 1007 1009 1010 1011 1012 1013 1014 1016 1018 1019 1020
   5    4    1    1    5    3    3    5    2    4    2    7    5    5    7    5    3    6    5    6    7
1021 1022 1023 1024 1026 1027 1029 1031 1032 1033 1034 1036 1038 1039 1040 1042 1044 1045 1046 1047 1048
   5    5    6    1    4    7    7    7    7    7    7    5    8    7    5    5    3    5    5    8    5
1049 1050 1051 1053 1055 1056 1057 1059 1061 1062 1066 1067 1068 1069 1070 1071 1072 1073 1074 1076 1077
   6    2    1    6    5    5    7    7    5    1    1    3    5    6    7    5    1    6    8    2    7
1078 1079 1080 1081 1082 1083 1084 1085 1087 1088 1089 1090 1092 1093 1094 1095 1097 1098 1099 1101 1102
   5    1    7    7    8    8    1    4    4    4    1    4    1    7    6    1    8    4    8    8    4
1103 1104 1106 1107 1108 1109 1111 1112 1113 1114 1116 1117 1118 1119 1121 1122 1123 1127 1128 1129 1130
   7    6    2    4    2    2    2    5    1    5    5    8    3    8    1    8    4    4    4    4    6
1131 1132 1133 1134 1135 1136 1137 1138 1144 1145 1146 1149 1150 1151 1152 1155 1156 1157 1158 1160 1162
   7    5    1    5    8    1    2    7    5    5    1    4    1    1    1    7    7    1    1    1    1
1163 1165 1166 1167 1168 1169 1171 1173 1174 1175 1176 1177 1178 1180 1182 1183 1184 1185 1187 1188 1189
   7    5    4    8    5    4    5    5    7    1    3    3    7    8    6    7    4    6    8    4    6
1190 1191 1192 1194 1195 1196 1197 1198 1200 1201 1202 1203 1204 1205 1207 1208 1209 1210 1212 1213 1214
   5    5    7    1    7    3    7    1    6    4    4    3    8    8    3    3    4    5    1    5    8
1216 1217 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1230 1231 1232 1233 1234 1235 1236 1237 1238
   3    6    6    6    7    2    5    3    5    2    5    5    6    1    6    6    8    7    6    2    5
1239 1241 1242 1243 1244 1245 1247 1248 1249 1252 1253 1254 1259
   2    7    7    6    2    8    5    4    4    8    8    6    1
 [ reached getOption("max.print") -- omitted 3015 entries ]

Within cluster sum of squares by cluster:
[1] 4245 2969 3026 2774 3208 2763 3097 2415
 (between_SS / total_SS =  44.5 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
>
```

Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

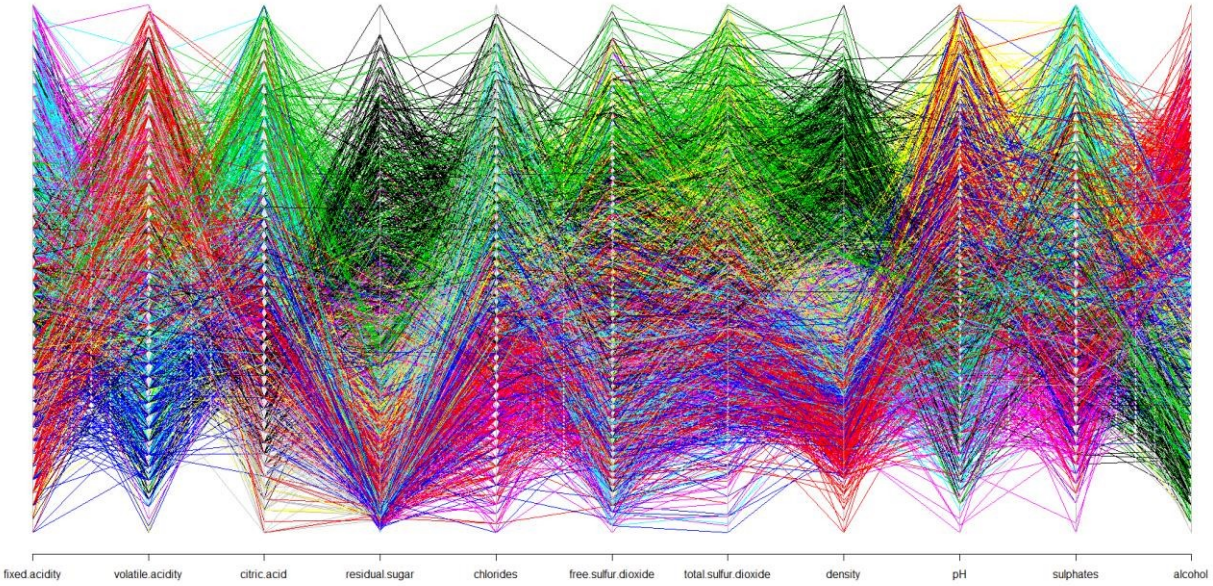
```
> plotcluster(white_wine_train, fit_km_8$cluster)
>
```



We can see the data is fairly clustered, there are few collapse between clusters.

Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km_8$cluster)
>
```



We can extract some insights from above graph as red cluster contains wine with lower chloride and lower density than other coloured cluster; green cluster contains wine with higher citric acid and higher total Sulphur dioxide than other coloured cluster, cyan cluster contains wine with higher fixed acidity than other coloured cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_8$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           6.9             0.24        0.28           13.1     0.049                  42
2       2           6.2             0.33        0.30            3.7     0.033                  30
3       3           6.9             0.29        0.43           10.6     0.048                  52
4       4           6.3             0.19        0.31            3.0     0.036                  28
5       5           7.5             0.23        0.38            2.8     0.043                  26
6       6           7.5             0.29        0.33            5.9     0.033                  30
7       7           6.7             0.23        0.31            4.4     0.045                  39
8       8           6.6             0.33        0.24            4.9     0.049                  27
  total.sulfur.dioxide density  pH sulphates alcohol
1                  159    1.00 3.1      0.46     9.4
2                  111    0.99 3.2      0.47    12.3
3                  189    1.00 3.1      0.52     9.4
4                  103    0.99 3.3      0.46    11.1
5                  116    0.99 3.1      0.53    10.8
6                  116    0.99 3.1      0.38    11.6
7                  159    0.99 3.3      0.56    10.5
8                  135    0.99 3.2      0.46     9.8
>
```

To check the consistency of fit_km_8 against column 12

```
> confuseTable_km_8 <- table(white_wine_output, fit_km_8$cluster)
> confuseTable_km_8

white_wine_output   1   2   3   4   5   6   7   8
                3   2   0   0   2   0   3   0   1
                4   8   7   9   4  16  12  11  22
                5 280  16 234  75 147  65  87 202
                6 307 212 207 258 264 204 267 147
                7  78 241  30 147  75 109 108   9
                8  18  35   3  22   9  37  19   2
                9   0   4   0   0   0   0   0   0
>
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1 to 8 clusters.

```
> randIndex(confuseTable_km_8)
  ARI
0.031
>
```

The agreement between the wine quality and cluster solution is 0.031. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is four. This maybe the possible reason for the low ARI.

Fit the model for k = 9

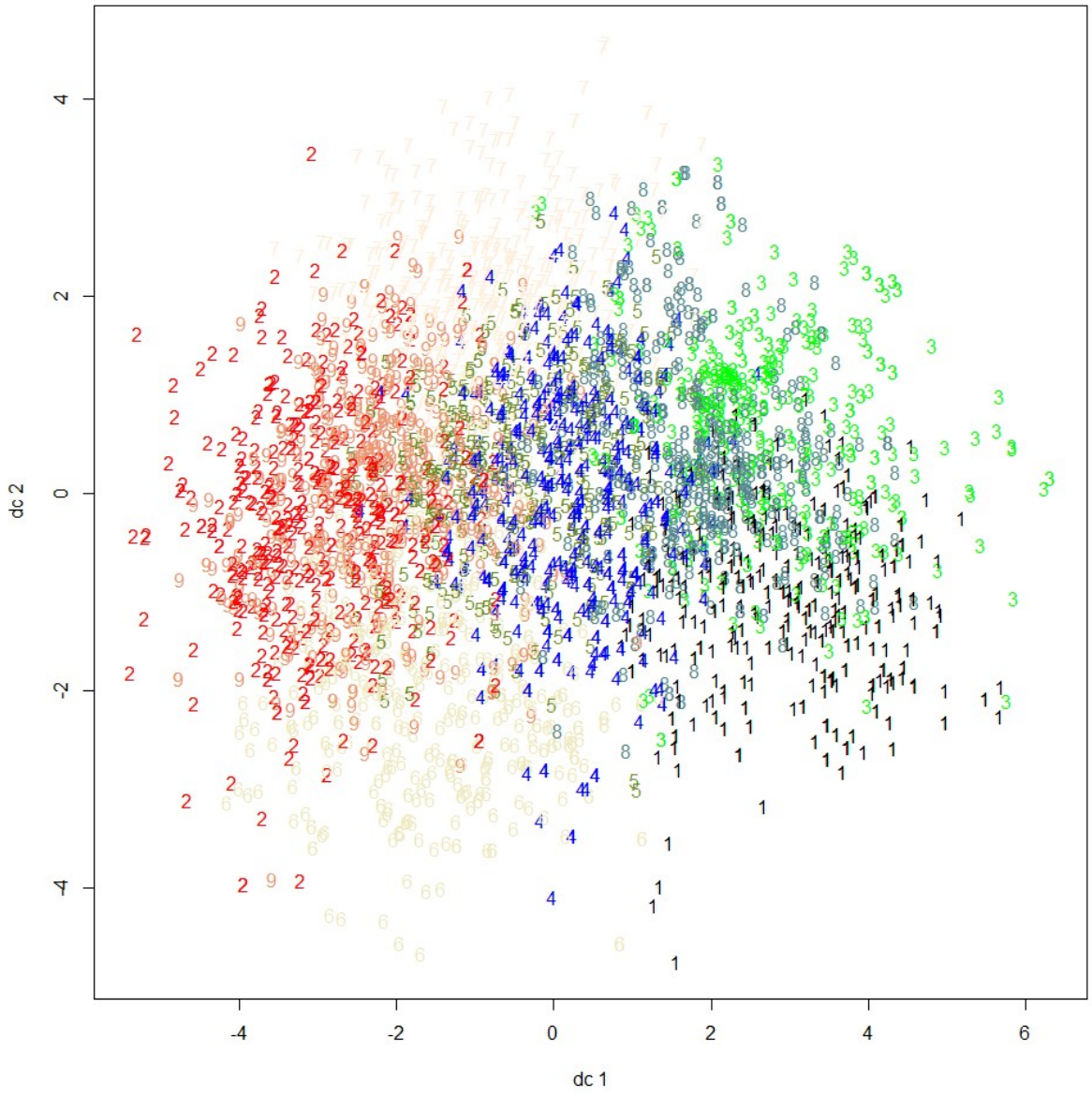
We now fit white_wine data to K-Means with k = 9 and interpret result:

```
> set.seed(3211)
> fit_km_9 <- kmeans(white_wine_train, 9)
> fit_km_9
K-means clustering with 9 clusters of sizes 491, 448, 378, 455, 384, 421, 464, 479, 495

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1         0.201            -0.47      -0.199           1.64      0.74                0.328
2        -0.827             1.07      -0.343          -0.53     -0.86               -0.321
3         0.064             0.30       1.538           0.80      0.54                1.066
4        -0.468             0.12      -0.885          -0.23      0.63               -0.581
5         1.065            -0.56       0.875          -0.77      0.48               -0.599
6         0.960             0.44       0.100          -0.12     -0.92               -0.439
7        -0.168            -0.47       0.011          -0.68     -0.19               -0.082
8         0.060             0.47      -0.668           0.51      0.48                0.908
9        -0.614            -0.79       0.021          -0.70     -0.82               -0.279
  total.sulfur.dioxide density     pH sulphates alcohol
1                 0.38    1.47 -0.462    -0.261  -1.031
2                -0.61   -1.16  0.445    -0.124   1.402
3                 1.18    1.00 -0.277     0.346  -0.941
4                -0.30    0.06  0.196    -0.347  -0.515
5                -0.44   -0.31 -0.373     0.021   0.016
6                -0.57   -0.52 -1.001    -0.744   0.838
7                -0.10   -0.37  0.787     1.353   0.191
8                 1.22    0.74  0.081     0.273  -0.696
9                -0.71   -0.92  0.412    -0.491   0.716
```

```
Clustering vector:
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   16   19   20   22   23   25   26
   1    4    5    8    8    5    4    1    4    5    6    5    5    7    9    6    4    9    7    4    8
  27   28   29   30   31   32   33   34   35   37   38   39   40   43   44   45   46   47   48   49   50
   4    7    5    7    5    5    5    7    4    4    6    1    1    4    4    4    5    8    8    4    4
  51   52   53   54   56   57   58   59   60   62   64   65   67   69   70   71   72   74   75   76   77
   5    7    9    9    7    1    1    4    9    1    4    4    2    7    4    3    8    5    7    4    9
  78   79   82   83   84   88   92   93   94   95   96   98  101  102  103  104  105  106  107  108  109
   7    1    1    8    8    8    3    9    9    9    8    6    1    1    8    1    1    1    1    1    1
 110  112  113  114  115  117  118  119  120  122  123  124  126  127  128  129  130  131  132  133  134
   7    3    3    8    8    2    9    3    3    1    4    4    9    6    8    3    3    9    8    3    1
 135  136  137  139  140  141  143  144  145  146  147  149  150  151  152  153  154  156  157  158  161
   8    8    6    9    6    4    5    5    5    9    9    2    4    6    3    4    5    1    1    2    2
 162  163  164  165  166  167  168  169  171  172  173  174  175  176  177  178  180  181  182  183  184
   3    4    1    3    3    3    2    1    5    5    6    9    1    6    6    1    8    8    8    1    3
 185  186  187  188  190  191  192  193  194  198  199  200  201  202  204  205  206  207  211  212  213
   3    3    3    2    8    8    1    6    4    8    8    8    1    8    7    4    3    1    9    7    1
 214  215  216  217  218  219  220  221  223  224  226  227  229  230  232  233  234  235  236  237  240
   4    9    8    6    8    1    6    6    9    4    1    7    7    1    3    3    1    1    1    1    8
 241  242  243  244  245  247  248  249  250  253  254  255  256  257  258  259  260  261  262  263  264
   8    6    9    8    3    7    9    9    4    3    9    4    7    2    2    9    2    5    1    6    3
 265  266  267  271  273  275  276  277  278  279  280  281  283  284  285  286  287  288  289  290  291
   9    1    8    8    1    3    3    3    5    5    7    9    8    8    8    8    1    1    1    1    1
1049 1050 1051 1053 1055 1056 1057 1059 1061 1062 1066 1067 1068 1069 1070 1071 1072 1073 1074 1076 1077
   6    2    1    6    6    5    5    5    5    8    1    3    5    6    7    6    1    6    8    2    8
1078 1079 1080 1081 1082 1083 1084 1085 1087 1088 1089 1090 1092 1093 1094 1095 1097 1098 1099 1101 1102
   7    1    8    4    8    8    1    4    9    9    1    9    1    5    6    1    4    4    4    8    9
1103 1104 1106 1107 1108 1109 1111 1112 1113 1114 1116 1117 1118 1119 1121 1122 1123 1127 1128 1129 1130
   7    6    2    9    2    2    2    5    8    5    5    2    8    2    1    4    9    9    9    4    6
1131 1132 1133 1134 1135 1136 1137 1138 1144 1145 1146 1149 1150 1151 1152 1155 1156 1157 1158 1160 1162
   4    4    1    5    4    1    9    2    4    5    8    9    8    8    8    8    7    1    1    1    1
1163 1165 1166 1167 1168 1169 1171 1173 1174 1175 1176 1177 1178 1180 1182 1183 1184 1185 1187 1188 1189
   7    5    9    4    5    9    5    5    7    8    3    8    7    4    6    7    9    6    5    9    9
1190 1191 1192 1194 1195 1196 1197 1198 1200 1201 1202 1203 1204 1205 1207 1208 1209 1210 1212 1213 1214
   5    5    7    1    8    1    7    1    6    9    9    3    4    4    3    3    9    6    4    5    4
1216 1217 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1230 1231 1232 1233 1234 1235 1236 1237 1238
   3    6    6    6    7    2    7    3    9    2    9    5    6    8    6    6    6    7    6    2    6
1239 1241 1242 1243 1244 1245 1247 1248 1249 1252 1253 1254 1259
   2    8    5    6    2    4    5    5    9    4    4    9    8
 [ reached getOption("max.print") -- omitted 3015 entries ]

Within cluster sum of squares by cluster:
[1] 2871 2606 2263 2761 2325 2724 2786 2799 2633
 (between_SS / total_SS =  46.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> |
```

Next is to draw discriminant projection plot using the plotcluster function from the fpc library:

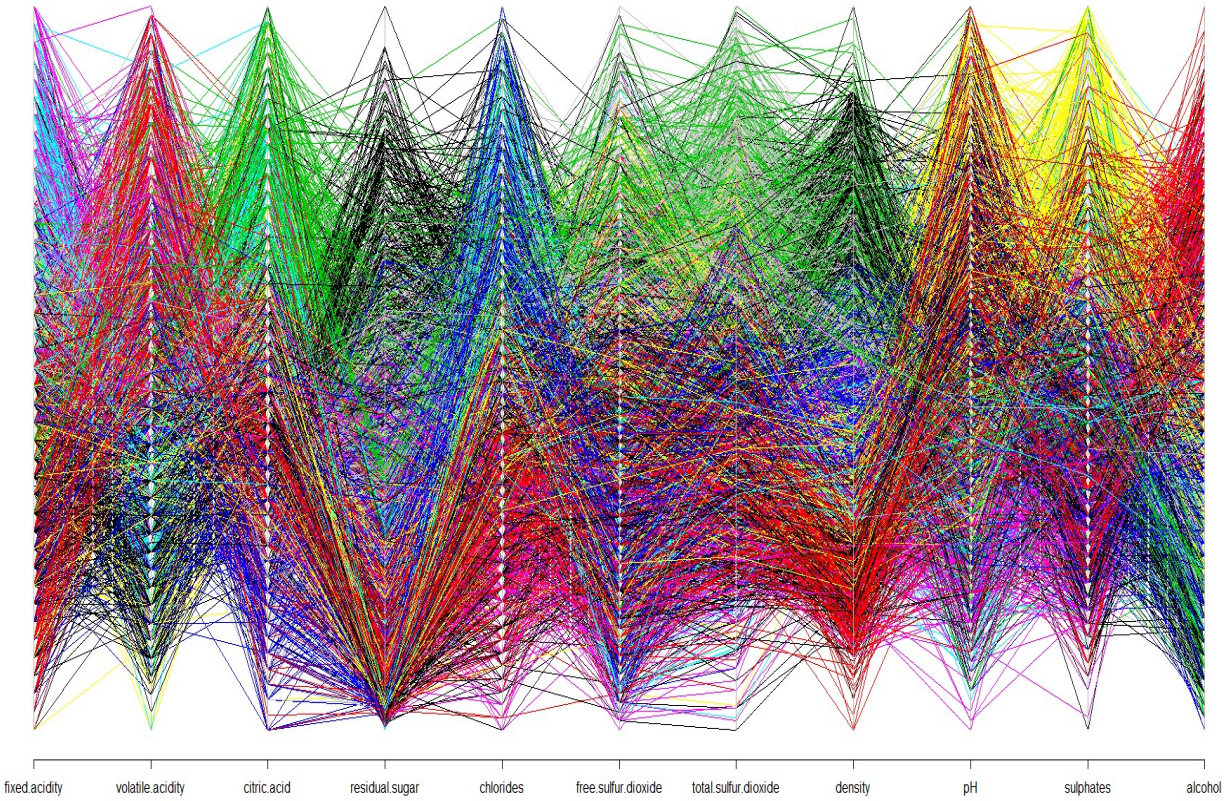
```
> plotcluster(white_wine_train, fit_km_9$cluster)
> |
```



We can see the data is fairly clustered, there are few collapse between clusters.

Next, we draw parallel coordinates plot to see how variables contributed in each cluster:

```
> windows(8,8) #To plot in a new window
> parcoord(white_wine_train, fit_km_9$cluster)
>
```



We can extract some insights from above graph as red cluster contains wine with lower density than other coloured cluster; green cluster contains wine with higher citric acid than other coloured cluster.

To calculate the cluster means, the mean of each attribute:

```
> aggregate(white_wine_input,by=list(fit_km_9$cluster),FUN=mean)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1           7.0              0.23        0.31          14.5     0.050                  40
2       2           6.2              0.35        0.29           3.7     0.034                  30
3       3           6.9              0.29        0.46          10.3     0.048                  51
4       4           6.5              0.27        0.25           5.2     0.049                  26
5       5           7.6              0.22        0.40           2.6     0.047                  26
6       6           7.5              0.30        0.33           5.8     0.033                  28
7       7           6.7              0.23        0.33           3.0     0.040                  34
8       8           6.9              0.30        0.27           8.9     0.047                  48
9       9           6.4              0.20        0.33           2.9     0.034                  31
  total.sulfur.dioxide density  pH sulphates alcohol
1                  153    1.00 3.1      0.46     9.3
2                  112    0.99 3.2      0.47    12.3
3                  186    1.00 3.1      0.52     9.4
4                  125    0.99 3.2      0.45    10.0
5                  119    0.99 3.1      0.48    10.6
6                  113    0.99 3.0      0.41    11.6
7                  133    0.99 3.3      0.62    10.8
8                  187    1.00 3.2      0.51     9.7
9                  108    0.99 3.2      0.43    11.5
>
```

To check the consistency of fit_km_8 against column 12

```
> confuseTable_km_9 <- table(white_wine_output, fit_km_9$cluster)
> confuseTable_km_9

white_wine_output   1   2   3   4   5   6   7   8   9
                3   1   0   0   1   0   3   0   1   2
                4   8   6   8  25  15  12   5   5   5
                5 163  24 187 170 123  68  68 241  62
                6 224 172 156 217 195 204 238 211 249
                7  78 209  24  37  49 103 133  18 146
                8  17  35   3   5   2  31  20   3  29
                9   0   2   0   0   0   0   0   0   2
>
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1 to 9 clusters.

```
> randIndex(confuseTable_km_9)
   ARI
0.026
>
```

The agreement between the wine quality and cluster solution is 0.026. This show a weak agreement though but we should bear in mind that the quality has seven levels while cluster is nine. This maybe the possible reason for the low ARI.

We can see the data is not clustered nicely, there are some collapse between clusters especially between clusters 1 to 4.

The fact that we cannot test the accuracy of our prediction properly due to differences in class labels and moreover, the ARI we've obtained so far suggests that perhaps kmeans is not likely the best algorithm that classifies the wine data properly. Meanwhile we can improve this better by leveraging other algorithms and or training more dataset. Wine quality prediction is kind of difficult task based on the dataset provided.
Finally, it'll be wise to go with the hclust and wss suggestions which stated that k = 2 is the best cluster and k = 4 is the second best. The fact that we cannot easily evaluate our kmeans model pose a big threat.

2nd Objective (hierarchical clustering)
Load required libraries:

```
> library(tidyverse)
> library(dendextend)
> library(gplots)
> library(cluster)
> library(NbClust)
>
```

Data exploration, preparation and processing.
Load the Whitewine dataset and check the structure of each variable:

```
> white_wine <- readxl::read_xlsx("whitewine.xlsx", sheet = 1)
> str(white_wine)
Classes 'tbl_df', 'tbl' and '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             : num  6 6 6 6 6 6 6 6 6 6 ...
>
```

All 12 variables appear to be numeric, although we need quality to be categorical. This will be done later.

Let's summarize the whitewine dataset to see some basic stats:

```
> summary(white_wine)
 fixed acidity    volatile acidity  citric acid     residual sugar     chlorides
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200   Median :0.04300
 Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391   Mean   :0.04577
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900   3rd Qu.:0.05000
 Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800   Max.   :0.34600
 free sulfur dioxide total sulfur dioxide    density             pH          sulphates
 Min.   :  2.00      Min.   :  9.0        Min.   :0.9871   Min.   :2.720   Min.   :0.2200
 1st Qu.: 23.00      1st Qu.:108.0        1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100
 Median : 34.00      Median :134.0        Median :0.9937   Median :3.180   Median :0.4700
 Mean   : 35.31      Mean   :138.4        Mean   :0.9940   Mean   :3.188   Mean   :0.4898
 3rd Qu.: 46.00      3rd Qu.:167.0        3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500
 Max.   :289.00      Max.   :440.0        Max.   :1.0390   Max.   :3.820   Max.   :1.0800
    alcohol         quality
 Min.   : 8.00   Min.   :3.000
 1st Qu.: 9.50   1st Qu.:5.000
 Median :10.40   Median :6.000
 Mean   :10.51   Mean   :5.878
 3rd Qu.:11.40   3rd Qu.:6.000
 Max.   :14.20   Max.   :9.000
>
```

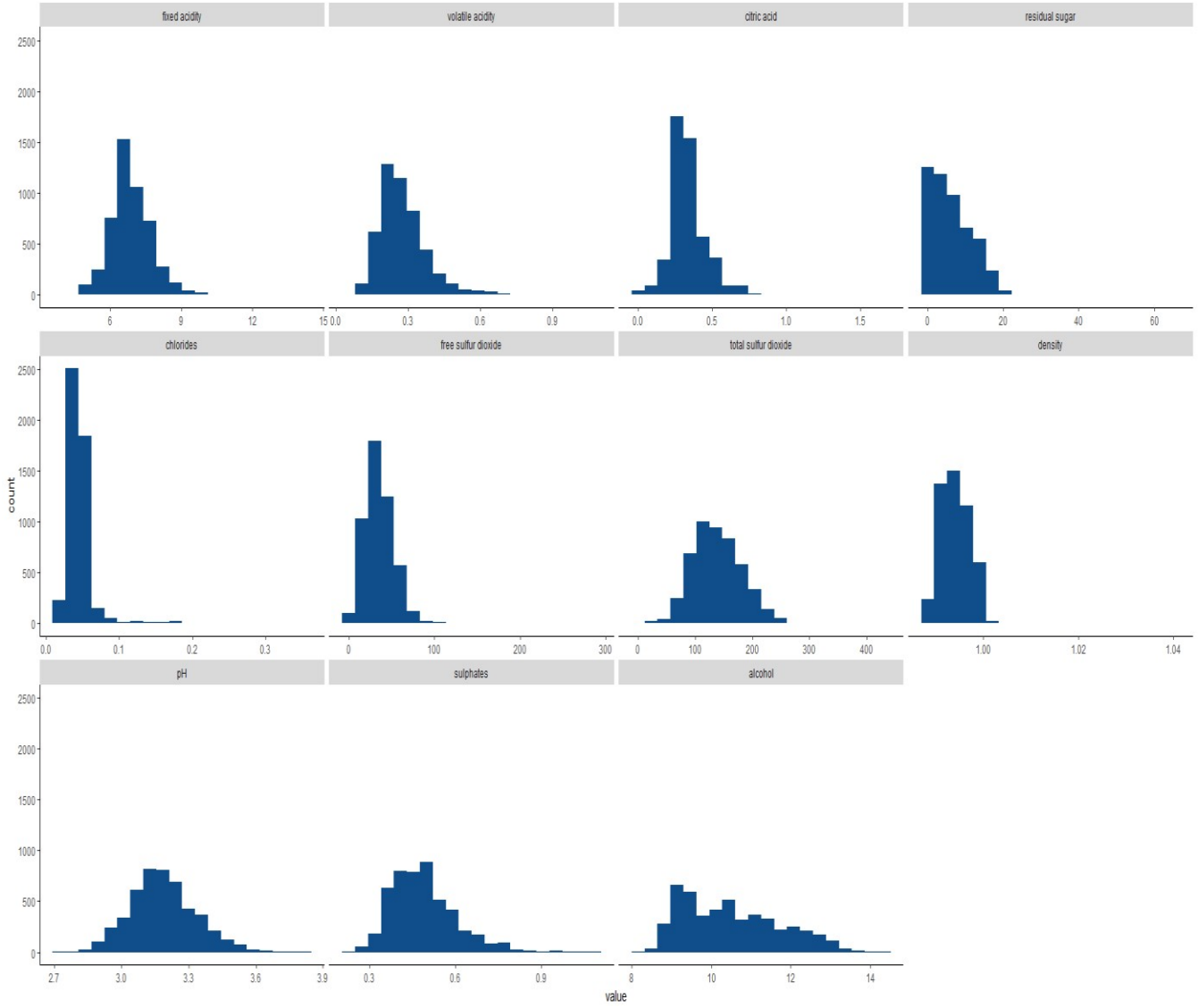
There happen to be no missing value.

```
> anyNA(white_wine)
[1] FALSE
>
```

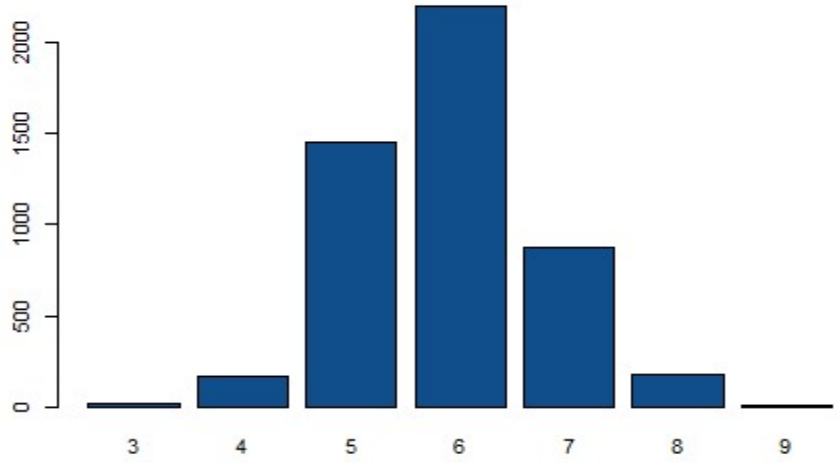
There happen to be no missing value.

Next is to plot the histogram of the first 11 variables to see how the data is being distributed:

```
> ggplot(data = reshape2::melt(white_wine[, -12]), mapping = aes(x = value)) +
+   geom_histogram(bins = 20, fill = "dodgerblue4") + facet_wrap(~variable, scales = 'free_x')+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
```



Bar plot of the quality

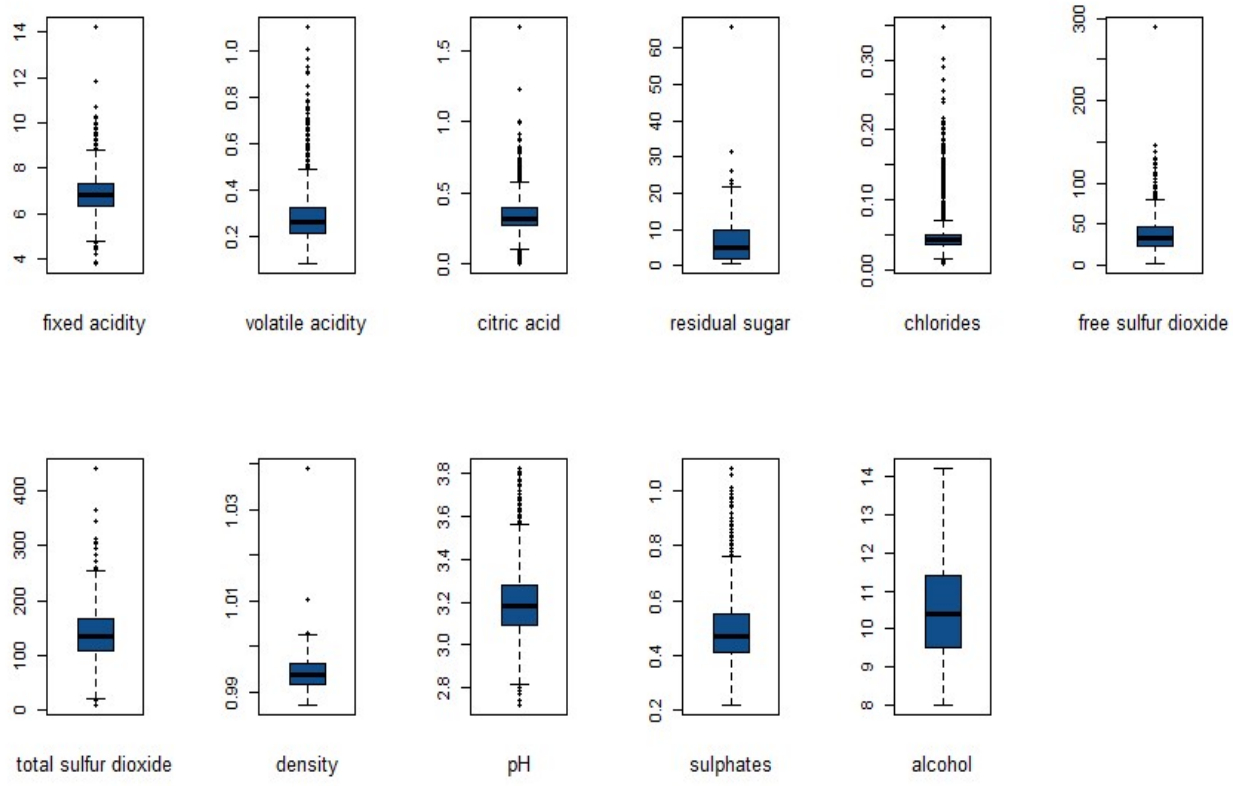



```
> unique(white_wine$quality)
[1] 6 5 7 8 4 3 9
> length(unique(white_wine$quality))
[1] 7
> 
> table(white_wine$quality)

   3    4    5    6    7    8    9 
  20  163 1457 2198  880  175    5 
> 
> round(100*prop.table(table(white_wine$quality)), 2)

    3     4     5     6     7     8     9 
 0.41  3.33 29.75 44.88 17.97  3.57  0.10 
> paste0(round(100*prop.table(table(white_wine$quality)), 2), "%")
[1] "0.41%"  "3.33%"  "29.75%" "44.88%" "17.97%" "3.57%"  "0.1%"
> 
```

There are 7 classes in the quality variable (ranging from 3 to 9) with most of the wine tested between quality 5 and 7. Although with some outliers, some variables such as Fixed acidity, total Sulphur dioxide, PH appear to be normally distributed. The fact that most are in favour of normal distribution but none of the properties appear to be uniformly distributed will form a basis for scaling decision. Hence, standardizing the 11 variables is appropriate for this analysis. Moreover, each variable in our dataset are of different standard unit. Meanwhile, we still need to deal with the outliers present in the dataset.

Dealing with outliers:
Let’s visualize using box to see how the outliers plays out:

```
> par( mfrow = c( 2, 6 ) )
> boxplot(white_wine$`fixed acidity`, col= "dodgerblue4", pch=19)
> mtext("fixed acidity", cex=0.8, side=1, line=2)
> boxplot(white_wine$`volatile acidity`, col= "dodgerblue4", pch=19)
> mtext("volatile acidity", cex=0.8, side=1, line=2)
> boxplot(white_wine$`citric acid`, col= "dodgerblue4", pch=19)
> mtext("citric acid", cex=0.8, side=1, line=2)
> boxplot(white_wine$`residual sugar`, col= "dodgerblue4", pch=19)
> mtext("residual sugar", cex=0.8, side=1, line=2)
> boxplot(white_wine$chlorides, col= "dodgerblue4", pch=19)
> mtext("chlorides", cex=0.8, side=1, line=2)
> boxplot(white_wine$`free sulfur dioxide`, col= "dodgerblue4", pch=19)
> mtext("free sulfur dioxide", cex=0.8, side=1, line=2)
> boxplot(white_wine$`total sulfur dioxide`, col= "dodgerblue4", pch=19)
> mtext("total sulfur dioxide", cex=0.8, side=1, line=2)
> boxplot(white_wine$density, col= "dodgerblue4", pch=19)
> mtext("density", cex=0.8, side=1, line=2)
> boxplot(white_wine$pH, col= "dodgerblue4", pch=19)
> mtext("pH", cex=0.8, side=1, line=2)
> boxplot(white_wine$sulphates, col= "dodgerblue4", pch=19)
> mtext("sulphates", cex=0.8, side=1, line=2)
> boxplot(white_wine$alcohol, col= "dodgerblue4", pch=19)
> mtext("alcohol", cex=0.8, side=1, line=2)
```



Fixed acidity, volatile acidity, citric acid, total Sulphur dioxide, PH and sulphates and have outliers. If those outliers are eliminated distribution of the variables may be taken to be symmetric.
Residual sugar has a positively skewed distribution; even after eliminating the outliers distribution will remain skewed.
Some of the variables like sulphur dioxide and density, have a few outliers but these are very different from the rest. Alcohol has an irregular shaped distribution but it does not have pronounced outliers.

We can further investigate the presence of outliers leveraging summary function and describe function from psych library:

```
> psych::describe(white_wine[, -12])
                      vars    n   mean    sd median trimmed   mad  min    max  range skew kurtosis   se
fixed acidity            1 4898   6.85  0.84   6.80    6.82  0.74 3.80  14.20  10.40 0.65     2.17 0.01
volatile acidity         2 4898   0.28  0.10   0.26    0.27  0.09 0.08   1.10   1.02 1.58     5.08 0.00
citric acid              3 4898   0.33  0.12   0.32    0.33  0.09 0.00   1.66   1.66 1.28     6.16 0.00
residual sugar           4 4898   6.39  5.07   5.20    5.80  5.34 0.60  65.80  65.20 1.08     3.46 0.07
chlorides                5 4898   0.05  0.02   0.04    0.04  0.01 0.01   0.35   0.34 5.02    37.51 0.00
free sulfur dioxide      6 4898  35.31 17.01  34.00   34.36 16.31 2.00 289.00 287.00 1.41    11.45 0.24
total sulfur dioxide     7 4898 138.36 42.50 134.00  136.96 43.00 9.00 440.00 431.00 0.39     0.57 0.61
density                  8 4898   0.99  0.00   0.99    0.99  0.00 0.99   1.04   0.05 0.98     9.78 0.00
pH                       9 4898   3.19  0.15   3.18    3.18  0.15 2.72   3.82   1.10 0.46     0.53 0.00
sulphates               10 4898   0.49  0.11   0.47    0.48  0.10 0.22   1.08   0.86 0.98     1.59 0.00
alcohol                 11 4898  10.51  1.23  10.40   10.43  1.48 8.00  14.20   6.20 0.49    -0.70 0.02
> summary(white_wine[, -12])
 fixed acidity  volatile acidity  citric acid   residual sugar   chlorides    free sulfur dioxide total sulfur dioxide
 Min.   : 3.8   Min.   :0.08     Min.   :0.00   Min.   : 1     Min.   :0.01   Min.   :  2         Min.   :  9
 1st Qu.: 6.3   1st Qu.:0.21     1st Qu.:0.27   1st Qu.: 2     1st Qu.:0.04   1st Qu.: 23         1st Qu.:108
 Median : 6.8   Median :0.26     Median :0.32   Median : 5     Median :0.04   Median : 34         Median :134
 Mean   : 6.9   Mean   :0.28     Mean   :0.33   Mean   : 6     Mean   :0.05   Mean   : 35         Mean   :138
 3rd Qu.: 7.3   3rd Qu.:0.32     3rd Qu.:0.39   3rd Qu.:10     3rd Qu.:0.05   3rd Qu.: 46         3rd Qu.:167
 Max.   :14.2   Max.   :1.10     Max.   :1.66   Max.   :66     Max.   :0.35   Max.   :289         Max.   :440
    density           pH        sulphates       alcohol
 Min.   :0.99   Min.   :2.7   Min.   :0.22   Min.   : 8.0
 1st Qu.:0.99   1st Qu.:3.1   1st Qu.:0.41   1st Qu.: 9.5
 Median :0.99   Median :3.2   Median :0.47   Median :10.4
 Mean   :0.99   Mean   :3.2   Mean   :0.49   Mean   :10.5
 3rd Qu.:1.00   3rd Qu.:3.3   3rd Qu.:0.55   3rd Qu.:11.4
 Max.   :1.04   Max.   :3.8   Max.   :1.08   Max.   :14.2
> |
```

It can be observed that range is much larger than the IQR, and mean is larger than the median. These observations support our suspicion that there are outliers which must be taken care of in the data set and before analysis.

Next we look at the bivariate analysis:
1. Correlation coefficients and
2. Pairwise scatterplot

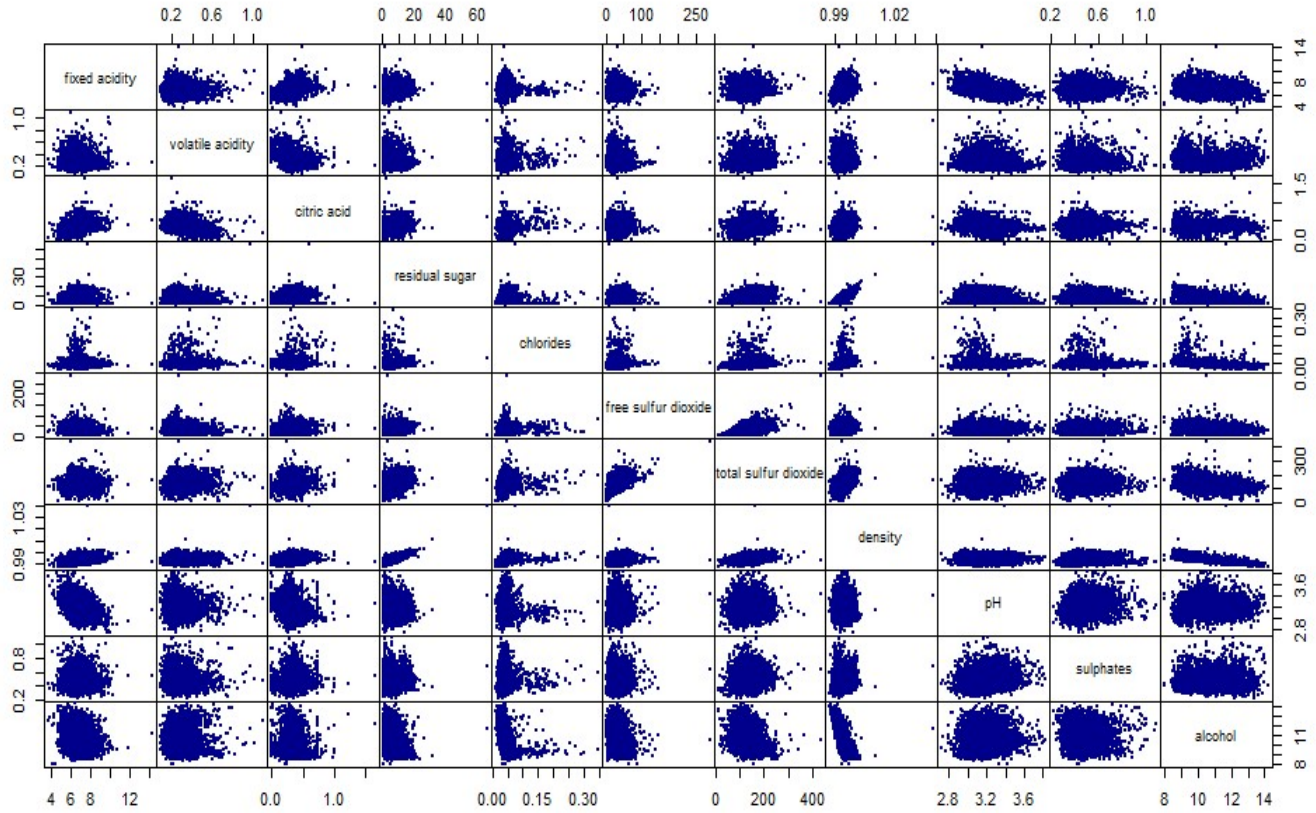
Pearson's correlation coefficients

```
> round(cor(white_wine[, -12], method = "pearson"), 2)
                      fixed acidity volatile acidity citric acid residual sugar chlorides
fixed acidity                  1.00            -0.02        0.29          0.09      0.02
volatile acidity              -0.02             1.00       -0.15          0.06      0.07
citric acid                    0.29            -0.15        1.00          0.09      0.11
residual sugar                 0.09             0.06        0.09          1.00      0.09
chlorides                      0.02             0.07        0.11          0.09      1.00
free sulfur dioxide           -0.05            -0.10        0.09          0.30      0.10
total sulfur dioxide           0.09             0.09        0.12          0.40      0.20
density                        0.27             0.03        0.15          0.84      0.26
pH                            -0.43            -0.03       -0.16         -0.19     -0.09
sulphates                     -0.02            -0.04        0.06         -0.03      0.02
alcohol                       -0.12             0.07       -0.08         -0.45     -0.36
                      free sulfur dioxide total sulfur dioxide density    pH sulphates alcohol
fixed acidity                       -0.05                 0.09    0.27 -0.43     -0.02   -0.12
volatile acidity                    -0.10                 0.09    0.03 -0.03     -0.04    0.07
citric acid                          0.09                 0.12    0.15 -0.16      0.06   -0.08
residual sugar                       0.30                 0.40    0.84 -0.19     -0.03   -0.45
chlorides                            0.10                 0.20    0.26 -0.09      0.02   -0.36
free sulfur dioxide                  1.00                 0.62    0.29  0.00      0.06   -0.25
total sulfur dioxide                 0.62                 1.00    0.53  0.00      0.13   -0.45
density                              0.29                 0.53    1.00 -0.09      0.07   -0.78
pH                                   0.00                 0.00   -0.09  1.00      0.16    0.12
sulphates                            0.06                 0.13    0.07  0.16      1.00   -0.02
alcohol                             -0.25                -0.45   -0.78  0.12     -0.02    1.00
```

The chemical properties with high correlation are highlighted in colors. I have considered correlation of 40% and above as high.

Pairwise scatterplot

```
> pairs(cor(white_wine[, -12]), gap=0, pch=19, cex=0.4, col="darkblue")
```



In lieu of the above, I have decided to apply the boxplot outlier detection method which imply that if a variable has a value more than Q3 + 1.5IQR and if has a value less than Q1 - 1.5IQR, it will be considered outlier.

Removing outliers:
First we write a function that represent outliers as NA and summarize:

```
> remove_outliers <- function(x) {
+   qnt <- quantile(x, probs=c(.25, .75))
+   checker <- 1.5 * IQR(x)
+   x[x < (qnt[[1]] - checker)] <- NA
+   x[x > (qnt[[2]] + checker)] <- NA
+   x
+ }
> white_wine_11 <- as.data.frame(lapply(white_wine[, -12], remove_outliers))
> nrow(white_wine_11)
[1] 4898
> summary(white_wine_11)
 fixed.acidity volatile.acidity  citric.acid  residual.sugar   chlorides   free.sulfur.dioxide
 Min.   :5     Min.   :0        Min.   :0     Min.   : 0.6   Min.   :0     Min.   : 2
 1st Qu.:6     1st Qu.:0        1st Qu.:0     1st Qu.: 1.7   1st Qu.:0     1st Qu.:23
 Median :7     Median :0        Median :0     Median : 5.2   Median :0     Median :33
 Mean   :7     Mean   :0        Mean   :0     Mean   : 6.4   Mean   :0     Mean   :35
 3rd Qu.:7     3rd Qu.:0        3rd Qu.:0     3rd Qu.: 9.8   3rd Qu.:0     3rd Qu.:45
 Max.   :9     Max.   :0        Max.   :1     Max.   :22.0   Max.   :0     Max.   :80
 NA's   :119   NA's   :186      NA's   :270   NA's   :7      NA's   :208   NA's   :50
 total.sulfur.dioxide    density         pH          sulphates      alcohol
 Min.   : 21          Min.   :1   Min.   :3    Min.   :0     Min.   : 8.0
 1st Qu.:108          1st Qu.:1   1st Qu.:3    1st Qu.:0     1st Qu.: 9.5
 Median :134          Median :1   Median :3    Median :0     Median :10.4
 Mean   :138          Mean   :1   Mean   :3    Mean   :0     Mean   :10.5
 3rd Qu.:167          3rd Qu.:1   3rd Qu.:3    3rd Qu.:1     3rd Qu.:11.4
 Max.   :255          Max.   :1   Max.   :4    Max.   :1     Max.   :14.2
 NA's   :19           NA's   :5   NA's   :75   NA's   :124
>
```

Next I add the quality column and check the result:

```
> white_wine_new <- cbind(white_wine_11, white_wine[ , 12])
> nrow(white_wine_new)
[1] 4898
> summary(white_wine_new)
 fixed.acidity volatile.acidity  citric.acid  residual.sugar   chlorides   free.sulfur.dioxide
 Min.   :5     Min.   :0        Min.   :0     Min.   : 0.6   Min.   :0     Min.   : 2
 1st Qu.:6     1st Qu.:0        1st Qu.:0     1st Qu.: 1.7   1st Qu.:0     1st Qu.:23
 Median :7     Median :0        Median :0     Median : 5.2   Median :0     Median :33
 Mean   :7     Mean   :0        Mean   :0     Mean   : 6.4   Mean   :0     Mean   :35
 3rd Qu.:7     3rd Qu.:0        3rd Qu.:0     3rd Qu.: 9.8   3rd Qu.:0     3rd Qu.:45
 Max.   :9     Max.   :0        Max.   :1     Max.   :22.0   Max.   :0     Max.   :80
 NA's   :119   NA's   :186      NA's   :270   NA's   :7      NA's   :208   NA's   :50
 total.sulfur.dioxide    density         pH          sulphates      alcohol        quality
 Min.   : 21          Min.   :1   Min.   :3    Min.   :0     Min.   : 8.0   Min.   :3.0
 1st Qu.:108          1st Qu.:1   1st Qu.:3    1st Qu.:0     1st Qu.: 9.5   1st Qu.:5.0
 Median :134          Median :1   Median :3    Median :0     Median :10.4   Median :6.0
 Mean   :138          Mean   :1   Mean   :3    Mean   :0     Mean   :10.5   Mean   :5.9
 3rd Qu.:167          3rd Qu.:1   3rd Qu.:3    3rd Qu.:1     3rd Qu.:11.4   3rd Qu.:6.0
 Max.   :255          Max.   :1   Max.   :4    Max.   :1     Max.   :14.2   Max.   :9.0
 NA's   :19           NA's   :5   NA's   :75   NA's   :124
>
```

Finally, we drop the NA's and see the result:

```
> white_wine_newer <- white_wine_new %>% drop_na()
> nrow(white_wine_newer)
[1] 4015
> summary(white_wine_newer)
 fixed.acidity volatile.acidity  citric.acid   residual.sugar   chlorides     free.sulfur.dioxide
 Min.   :4.8   Min.   :0.08     Min.   :0.10   Min.   : 0.6   Min.   :0.015   Min.   : 2
 1st Qu.:6.3   1st Qu.:0.21     1st Qu.:0.27   1st Qu.: 1.8   1st Qu.:0.035   1st Qu.:24
 Median :6.8   Median :0.26     Median :0.31   Median : 5.2   Median :0.042   Median :34
 Mean   :6.8   Mean   :0.26     Mean   :0.32   Mean   : 6.4   Mean   :0.042   Mean   :35
 3rd Qu.:7.3   3rd Qu.:0.31     3rd Qu.:0.37   3rd Qu.: 9.7   3rd Qu.:0.049   3rd Qu.:45
 Max.   :8.8   Max.   :0.48     Max.   :0.57   Max.   :22.0   Max.   :0.071   Max.   :80
 total.sulfur.dioxide    density           pH        sulphates       alcohol        quality
 Min.   : 21          Min.   :0.99   Min.   :2.8   Min.   :0.22   Min.   : 8.4   Min.   :3.0
 1st Qu.:107          1st Qu.:0.99   1st Qu.:3.1   1st Qu.:0.41   1st Qu.: 9.5   1st Qu.:5.0
 Median :132          Median :0.99   Median :3.2   Median :0.47   Median :10.5   Median :6.0
 Mean   :137          Mean   :0.99   Mean   :3.2   Mean   :0.48   Mean   :10.6   Mean   :5.9
 3rd Qu.:166          3rd Qu.:1.00   3rd Qu.:3.3   3rd Qu.:0.54   3rd Qu.:11.4   3rd Qu.:6.0
 Max.   :255          Max.   :1.00   Max.   :3.6   Max.   :0.76   Max.   :14.2   Max.   :9.0
>
```

The application of the outlier removal rule has reduced our dataset from the original 4898 to 4015. Now we are left with 4015 dataset.

Assign the data from column 1-11 (features) to variable white_wine_input, and the class to variable white_wine_output.

```
> white_wine_input <- white_wine_newer[, -12]
> names(white_wine_input)
 [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"          "residual.sugar"
 [5] "chlorides"            "free.sulfur.dioxide"  "total.sulfur.dioxide" "density"
 [9] "pH"                   "sulphates"            "alcohol"
> length(white_wine_input)
[1] 11
> white_wine_output <- as.factor(white_wine_newer$quality)
> str(white_wine_output)
 Factor w/ 7 levels "3","4","5","6",..: 4 4 4 4 4 4 4 4 4 4 ...
>
```

Next is to scale the white_wine_input to have mean = 0, standard deviation = 1 using the built in R scale function and assign it value to white_wine_train. . This will enable us to create a meaningful distance matrix.

```
> white_wine_train <- scale(white_wine_input)
> summary(white_wine_train)
 fixed.acidity   volatile.acidity  citric.acid    residual.sugar   chlorides     free.sulfur.dioxide
 Min.   :-2.73   Min.   :-2.42    Min.   :-2.60   Min.   :-1.2   Min.   :-2.75   Min.   :-2.18
 1st Qu.:-0.70   1st Qu.:-0.71    1st Qu.:-0.63   1st Qu.:-0.9   1st Qu.:-0.74   1st Qu.:-0.72
 Median :-0.02   Median :-0.05    Median :-0.17   Median :-0.2   Median :-0.03   Median :-0.05
 Mean   : 0.00   Mean   : 0.00    Mean   : 0.00   Mean   : 0.0   Mean   : 0.00   Mean   : 0.00
 3rd Qu.: 0.66   3rd Qu.: 0.60    3rd Qu.: 0.53   3rd Qu.: 0.7   3rd Qu.: 0.67   3rd Qu.: 0.68
 Max.   : 2.70   Max.   : 2.90    Max.   : 2.85   Max.   : 3.2   Max.   : 2.89   Max.   : 3.02
 total.sulfur.dioxide    density           pH          sulphates        alcohol
 Min.   :-2.82        Min.   :-2.34   Min.   :-2.66   Min.   :-2.64   Min.   :-1.80
 1st Qu.:-0.73        1st Qu.:-0.78   1st Qu.:-0.70   1st Qu.:-0.72   1st Qu.:-0.89
 Median :-0.12        Median :-0.11   Median :-0.05   Median :-0.11   Median :-0.07
 Mean   : 0.00        Mean   : 0.00   Mean   : 0.00   Mean   : 0.00   Mean   : 0.00
 3rd Qu.: 0.71        3rd Qu.: 0.71   3rd Qu.: 0.67   3rd Qu.: 0.59   3rd Qu.: 0.67
 Max.   : 2.87        Max.   : 2.81   Max.   : 2.70   Max.   : 2.82   Max.   : 2.98
>
```

As expected, all the variable has mean of zero.

First we create the distance matrix based on the Euclidean distance.

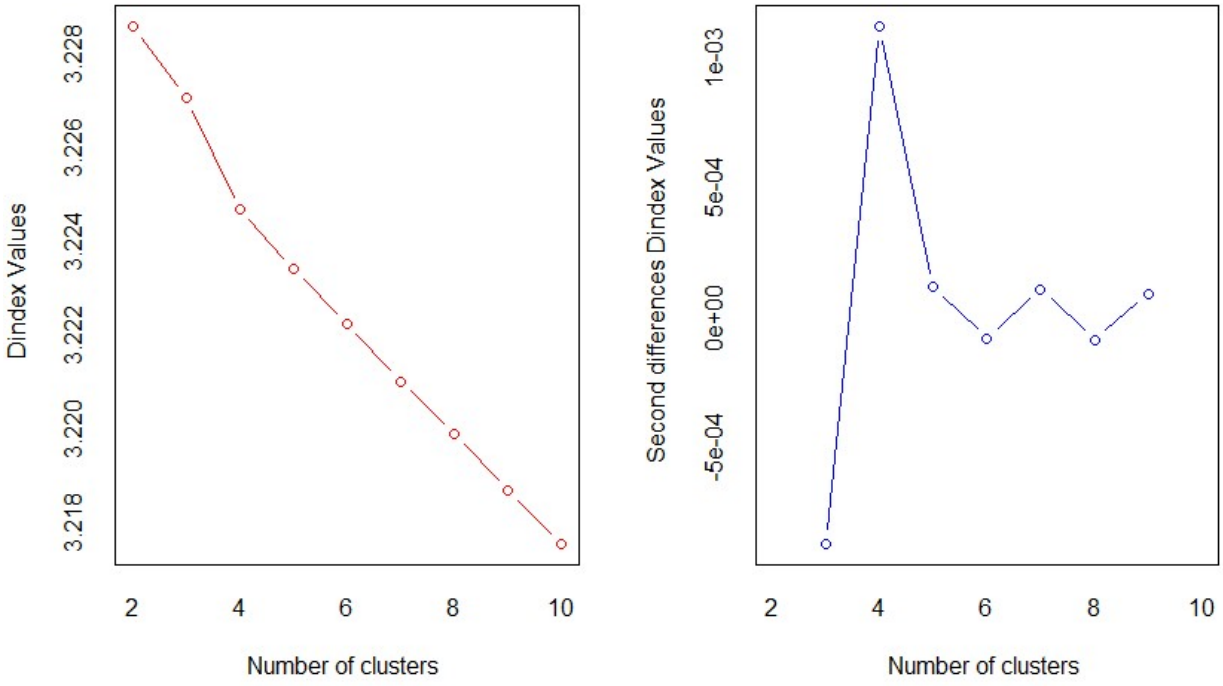
```
> d <- dist(white_wine_train, method = "euclidean")
>
```

Next, we'll conduct Hierarchical clustering for single, complete and average

Single Method:
First, we'll use hierarchical clustering, with the option single. But before that we need to determine what number of clusters to be later considered.

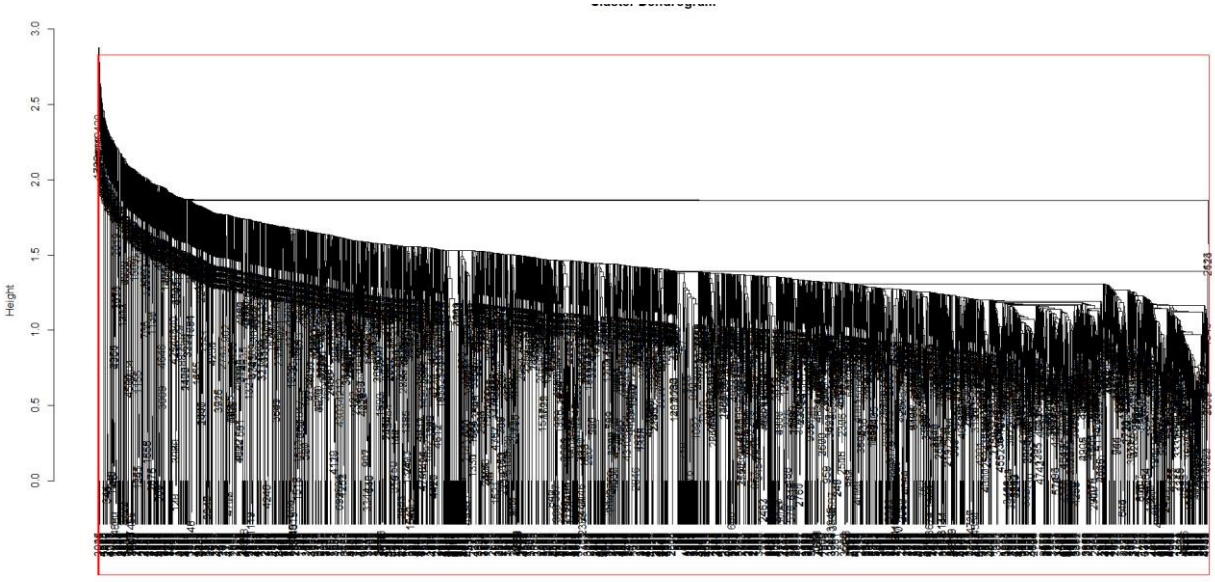
```
> NbClust(white_wine_train, distance = "euclidean", min.nc=2, max.nc=10,
+         method = "single", index = "all", alphaBeale = 0.1)
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
                In the plot of D index, we seek a significant knee (the significant peak in Dindex
                second differences plot) that corresponds to a significant increase of the value of
                the measure.

*******************************************************************
* Among all indices:
* 9 proposed 2 as the best number of clusters
* 3 proposed 3 as the best number of clusters
* 6 proposed 4 as the best number of clusters
* 1 proposed 9 as the best number of clusters
* 4 proposed 10 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2


*******************************************************************
$All.index
       KL      CH Hartigan       CCC    Scott      Marriot   TrCovW   TraceW Friedman  Rubin Cindex     DB Silhouette
2  1.5517 4.8231   3.2879  -67.5335  56.1352 4.850067e+37 26003409 44101.00   0.1134 1.0012 0.4085 0.8869     0.2356
3  1.3358 4.0565   4.0627  -95.0154  79.8821 1.084830e+38 25958440 44064.89   0.2370 1.0020 0.4084 0.7749     0.2226
4  1.2260 4.0603   2.4021 -116.9228 130.6838 1.904338e+38 25922514 44020.32   0.3586 1.0030 0.4083 0.7564     0.1278
5  1.1800 3.6466   1.9346 -136.0707 154.1384 2.958196e+38 25893994 43993.97   0.3999 1.0036 0.4082 0.7341     0.0766
6  1.1539 3.3048   2.2267 -153.3557 181.5540 4.230814e+38 25878949 43972.75   0.4156 1.0041 0.4081 0.7315     0.0354
7  1.1249 3.1259   1.9045 -169.1081 217.8602 5.706769e+38 25864902 43948.35   0.4369 1.0047 0.4081 0.7226     0.0274
8  1.1158 2.9519   2.2199 -183.7987 245.1069 7.403328e+38 25846049 43927.47   0.4536 1.0052 0.4080 0.7228    -0.0033
9  1.0943 2.8612   1.8673 -197.6041 272.6400 9.305803e+38 25827055 43903.15   0.5092 1.0057 0.4080 0.7170    -0.0212
10 1.0923 2.7512   2.0794 -198.6366 294.2303 1.142703e+39 25810864 43882.69   0.5405 1.0062 0.4079 0.7184    -0.0436
     Duda Pseudot2  Beale Ratkowsky      Ball Ptbiserial      Frey McClain   Dunn Hubert SDindex Dindex   SDbw
2  0.9996   1.5860 0.0029    0.0235 22050.498     0.0423 -175.0027  0.0007 0.2594  1e-04  0.6151 3.2286 0.6201
3  0.9990   4.0609 0.0075    0.0252 14688.298     0.0579  211.2704  0.0011 0.2505  1e-04  0.5797 3.2271 0.4132
4  1.0000   0.0662 0.0001    0.0263 11005.079     0.0645   15.8882  0.0019 0.2376  1e-04  0.5212 3.2247 0.3098
5  0.9999   0.2558 0.0005    0.0259  8798.794     0.0701  123.3201  0.0022 0.2376  1e-04  0.4890 3.2234 0.2478
6  0.9999   0.4117 0.0008    0.0253  7328.792     0.0725   21.8051  0.0026 0.2363  1e-04  0.4500 3.2223 0.2064
7  1.0000   0.0191 0.0000    0.0249  6278.335     0.0767   94.0468  0.0030 0.2357  1e-04  0.4340 3.2210 0.1769
8  0.9999   0.5395 0.0010    0.0245  5490.934     0.0787   13.0543  0.0034 0.2302  1e-04  0.4186 3.2199 0.1548
9  1.0000   0.1868 0.0003    0.0246  4878.128     0.0825   74.4267  0.0038 0.2290  1e-04  0.4217 3.2187 0.1375
10 0.9999   0.2627 0.0005    0.0243  4388.269     0.0842   22.5709  0.0042 0.2289  1e-04  0.4109 3.2176 0.1238
$All.CriticalValues
   CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
2          0.9214            342.1826           1
3          0.9214            342.1095           1
4          0.9214            341.9633           1
5          0.9214            341.8902           1
6          0.9214            341.8171           1
7          0.9214            341.7440           1
8          0.9214            341.6708           1
9          0.9214            341.5977           1
10         0.9214            341.5246           1

$Best.nc
                    KL      CH Hartigan      CCC   Scott      Marriot   TrCovW  TraceW Friedman  Rubin  Cindex    DB
Number_clusters 2.0000 2.0000   4.0000   2.0000  4.0000 4.000000e+00     3.00  4.0000   3.0000  4e+00 10.0000 9.000
Value_Index     1.5517 4.8231   1.6605 -67.5335 50.8017 2.343504e+37 44969.09 18.2288   0.1236 -4e-04  0.4079 0.717
                Silhouette   Duda PseudoT2  Beale Ratkowsky   Ball PtBiserial Frey McClain   Dunn Hubert SDindex Dindex
Number_clusters     2.0000 2.0000    2.000 2.0000    4.0000    3.0    10.0000    1   2e+00 2.0000       0 10.0000       0
Value_Index         0.2356 0.9996    1.586 0.0029    0.0263 7362.2     0.0842   NA   7e-04 0.2594       0  0.4109       0
                SDbw
Number_clusters 10.0000
Value_Index      0.1238
```

```
$Best.partition
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  16  19  20  22  23  25  26  27  28  29
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 30  31  32  33  34  35  37  38  39  40  43  44  45  46  47  48  49  50  51  52  53  54  56  57
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 58  59  60  62  64  65  67  69  70  71  72  74  75  76  77  78  79  82  83  84  88  92  93  94
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
 95  96  98 101 102 103 104 105 106 107 108 109 110 112 113 114 115 117 118 119 120 122 123 124
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
126 127 128 129 130 131 132 133 134 135 136 137 139 140 141 143 144 145 146 147 149 150 151 152
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
153 154 156 157 158 161 162 163 164 165 166 167 168 169 171 172 173 174 175 176 177 178 180 181
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
182 183 184 185 186 187 188 190 191 192 193 194 198 199 200 201 202 204 205 206 207 211 212 213
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
214 215 216 217 218 219 220 221 223 224 226 227 229 230 232 233 234 235 236 237 240 241 242 243
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
244 245 247 248 249 250 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 271 273 275
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
276 277 278 279 280 281 283 284 285 286 287 288 289 290 291 292 293 294 296 299 300 302 304 305
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
306 307 308 309 310 311 313 314 315 317 318 319 320 322 323 324 325 327 328 329 330 331 332 333
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
334 335 336 337 338 339 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 360
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
361 362 363 364 365 366 368 369 370 371 372 374 375 376 377 378 379 380 381 382 383 384 385 386
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
```



Fit model, display dendogram and draw dendogram with red borders around the 2 clusters.

```
> fit_single <- hclust(d, method="single")
> windows(8,8) #To plot in a new window
> plot(fit_single)
> rect.hclust(fit_single, k=2, border="red")
>
```



The two clusters are not well separated, and it feels like there are multitude of clusters generated on hclust. It is possible that the number of clusters is higher than proposed by Nbclust.
Let's cut the tree into 2 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_single <- cutree(fit_single, k=2)
> table(groups_single)
groups_single
   1    2
4013    2
>
```

The above classification also validate the cluster with k = 2 above has not clustering the whitewine data well. Almost all the elements in the data is clustered to 1

Let's check the consistency of the results against column 12:

```
> table(white_wine_output,groups_single)
                 groups_single
white_wine_output    1    2
                3    8    0
                4   89    0
                5 1106    0
                6 1864    2
                7  797    0
                8  145    0
                9    4    0
```

Almost all the whitewine quality are found in cluster 1. As seen in the dendogram plot, most of the elements are found in cluster 1.

Next, let's find the mean of each attribute for k = 2.

```
> round(aggregate(white_wine_input,by=list(groups_single),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide total.sulfur.dioxide
1       1          6.81             0.26        0.32          6.37      0.04                34.73               136.96
2       2          8.15             0.39        0.45         14.15      0.05                68.00               205.00
  density   pH sulphates alcohol
1    0.99 3.19      0.48   10.58
2    1.00 3.42      0.36    9.75
>
```

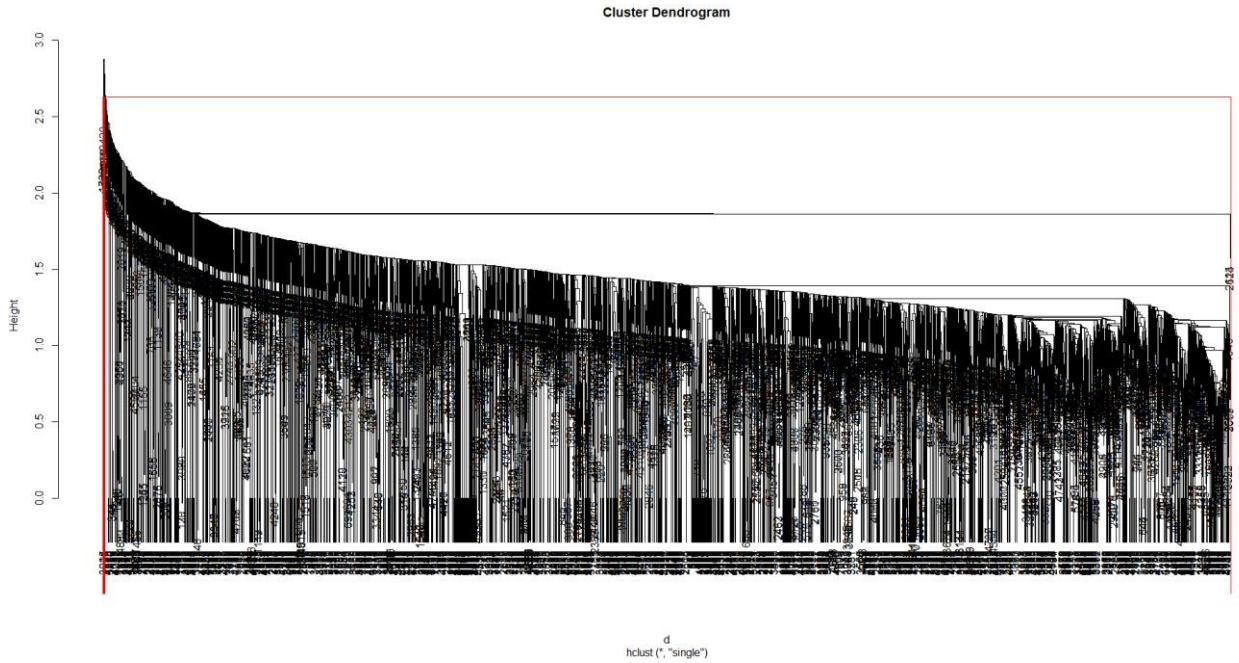
There's are some differences in the quality and chemical properties between the clusters.

Let's look at the 2 clusters:

```
> by(white_wine_input, groups_single, summary)
groups_single: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.600   Min.   :0.01500   Min.   : 2.00
 1st Qu.:6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.800   1st Qu.:0.03500   1st Qu.:24.00
 Median :6.800   Median :0.2600   Median :0.3100   Median : 5.200   Median :0.04200   Median :34.00
 Mean   :6.813   Mean   :0.2641   Mean   :0.3242   Mean   : 6.372   Mean   :0.04231   Mean   :34.73
 3rd Qu.:7.300   3rd Qu.:0.3100   3rd Qu.:0.3700   3rd Qu.: 9.700   3rd Qu.:0.04900   3rd Qu.:45.00
 Max.   :8.800   Max.   :0.4850   Max.   :0.5700   Max.   :22.000   Max.   :0.07100   Max.   :80.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21          Min.   :0.9871   Min.   :2.820   Min.   :0.2200   Min.   : 8.40
 1st Qu.:107          1st Qu.:0.9916   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
 Median :132          Median :0.9935   Median :3.180   Median :0.4700   Median :10.50
 Mean   :137          Mean   :0.9939   Mean   :3.187   Mean   :0.4812   Mean   :10.58
 3rd Qu.:166          3rd Qu.:0.9959   3rd Qu.:3.280   3rd Qu.:0.5400   3rd Qu.:11.40
 Max.   :255          Max.   :1.0020   Max.   :3.560   Max.   :0.7600   Max.   :14.20
------------------------------------------------------------------------------------------
groups_single: 2
 fixed.acidity   volatile.acidity  citric.acid    residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :7.600   Min.   :0.3400   Min.   :0.440   Min.   :13.80   Min.   :0.0450   Min.   :68
 1st Qu.:7.875   1st Qu.:0.3663   1st Qu.:0.445   1st Qu.:13.97   1st Qu.:0.0475   1st Qu.:68
 Median :8.150   Median :0.3925   Median :0.450   Median :14.15   Median :0.0500   Median :68
 Mean   :8.150   Mean   :0.3925   Mean   :0.450   Mean   :14.15   Mean   :0.0500   Mean   :68
 3rd Qu.:8.425   3rd Qu.:0.4188   3rd Qu.:0.455   3rd Qu.:14.32   3rd Qu.:0.0525   3rd Qu.:68
 Max.   :8.700   Max.   :0.4450   Max.   :0.460   Max.   :14.50   Max.   :0.0550   Max.   :68
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   :198.0        Min.   :0.9986   Min.   :3.36   Min.   :0.3600   Min.   : 9.500
 1st Qu.:201.5        1st Qu.:0.9987   1st Qu.:3.39   1st Qu.:0.3625   1st Qu.: 9.625
 Median :205.0        Median :0.9987   Median :3.42   Median :0.3650   Median : 9.750
 Mean   :205.0        Mean   :0.9987   Mean   :3.42   Mean   :0.3650   Mean   : 9.750
 3rd Qu.:208.5        3rd Qu.:0.9988   3rd Qu.:3.45   3rd Qu.:0.3675   3rd Qu.: 9.875
 Max.   :212.0        Max.   :0.9988   Max.   :3.48   Max.   :0.3700   Max.   :10.000
>
```

We will investigate this by using k = 5.

```
> windows(8,8) #To plot in a new window
> plot(fit_single)
> rect.hclust(fit_single, k=5, border="red")
>
```



With k = 5, we can get little separation between clusters. It is becoming clearer that the possible number of k is far bigger.

We will investigate this by using k = 5.

Let's cut the tree into 5 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_single <- cutree(fit_single, k=5)
> table(groups_single)
groups_single
   1    2    3    4    5
4009    2    1    2    1
>
```

The above classification also validate the cluster with k = 5 above has not clustering the whitewine data well as almost all the elements in the data is clustered to 1.

Let's check the consistency of the results against column 12:

```
> table(white_wine_output,groups_single)
                 groups_single
white_wine_output    1    2    3    4    5
                3    8    0    0    0    0
                4   89    0    0    0    0
                5 1105    0    1    0    0
                6 1861    2    0    2    1
                7  797    0    0    0    0
                8  145    0    0    0    0
                9    4    0    0    0    0
>
```

As seen in the dendogram plot, most of the elements are found in cluster 1.

Next, let's find the mean of each attribute for k = 5.

```
> round(aggregate(white_wine_input,by=list(groups_single),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1          6.81             0.26        0.32          6.36      0.04                34.73
2       2          8.15             0.39        0.45         14.15      0.05                68.00
3       3          6.10             0.36        0.41         19.35      0.07                67.00
4       4          7.00             0.45        0.34         19.80      0.04                12.00
5       5          4.90             0.47        0.17          1.90      0.04                60.00
  total.sulfur.dioxide density   pH sulphates alcohol
1               136.98    0.99 3.19      0.48   10.58
2               205.00    1.00 3.42      0.36    9.75
3               207.00    1.00 3.39      0.53    9.10
4                67.00    1.00 3.07      0.38   11.00
5               148.00    0.99 3.27      0.35   11.50
> |
```

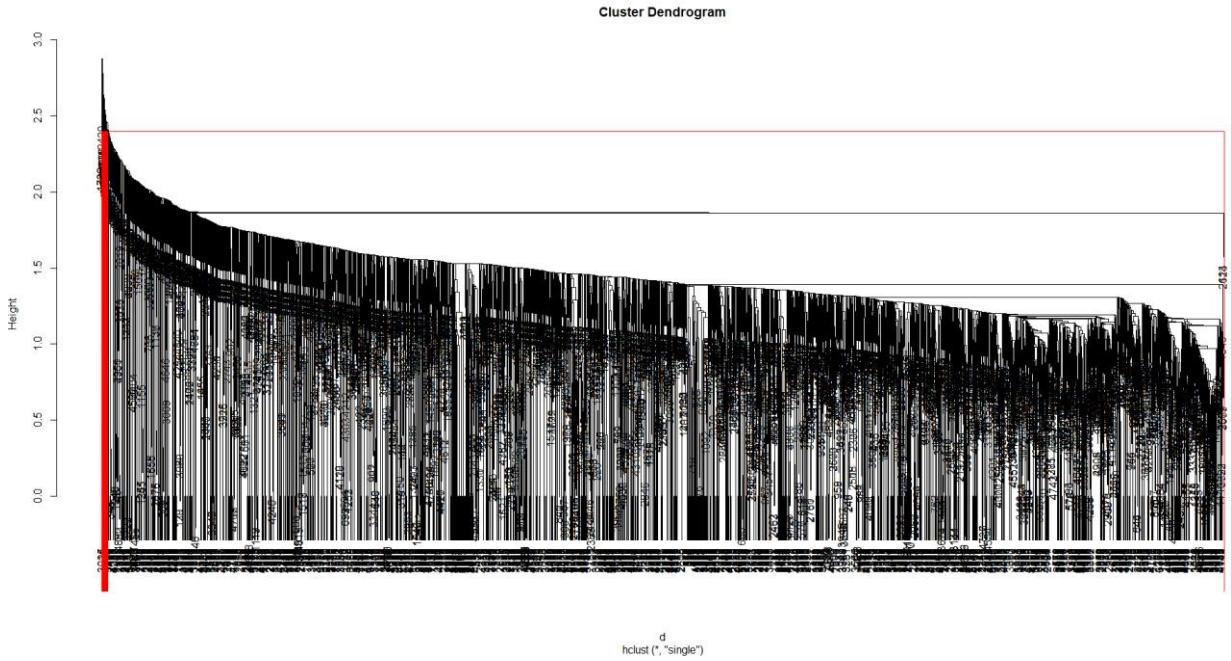
There's are some differences in the quality and chemical properties between the clusters.

```
> by(white_wine_input, groups_single, summary)
groups_single: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.600   Min.   :0.01500   Min.   : 2.00
 1st Qu.:6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.800   1st Qu.:0.03500   1st Qu.:24.00
 Median :6.800   Median :0.2600   Median :0.3100   Median : 5.200   Median :0.04200   Median :34.00
 Mean   :6.813   Mean   :0.2639   Mean   :0.3242   Mean   : 6.363   Mean   :0.04231   Mean   :34.73
 3rd Qu.:7.300   3rd Qu.:0.3100   3rd Qu.:0.3700   3rd Qu.: 9.650   3rd Qu.:0.04900   3rd Qu.:45.00
 Max.   :8.800   Max.   :0.4850   Max.   :0.5700   Max.   :22.000   Max.   :0.07100   Max.   :80.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21          Min.   :0.9871   Min.   :2.820   Min.   :0.2200   Min.   : 8.40
 1st Qu.:107          1st Qu.:0.9916   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
 Median :132          Median :0.9935   Median :3.180   Median :0.4700   Median :10.50
 Mean   :137          Mean   :0.9939   Mean   :3.187   Mean   :0.4813   Mean   :10.58
 3rd Qu.:166          3rd Qu.:0.9959   3rd Qu.:3.280   3rd Qu.:0.5400   3rd Qu.:11.40
 Max.   :255          Max.   :1.0020   Max.   :3.560   Max.   :0.7600   Max.   :14.20
------------------------------------------------------------------------------------
groups_single: 2
 fixed.acidity   volatile.acidity  citric.acid    residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :7.600   Min.   :0.3400   Min.   :0.440   Min.   :13.80   Min.   :0.0450   Min.   :68
 1st Qu.:7.875   1st Qu.:0.3663   1st Qu.:0.445   1st Qu.:13.97   1st Qu.:0.0475   1st Qu.:68
 Median :8.150   Median :0.3925   Median :0.450   Median :14.15   Median :0.0500   Median :68
 Mean   :8.150   Mean   :0.3925   Mean   :0.450   Mean   :14.15   Mean   :0.0500   Mean   :68
 3rd Qu.:8.425   3rd Qu.:0.4188   3rd Qu.:0.455   3rd Qu.:14.32   3rd Qu.:0.0525   3rd Qu.:68
 Max.   :8.700   Max.   :0.4450   Max.   :0.460   Max.   :14.50   Max.   :0.0550   Max.   :68
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   :198.0        Min.   :0.9986   Min.   :3.36   Min.   :0.3600   Min.   : 9.500
 1st Qu.:201.5        1st Qu.:0.9987   1st Qu.:3.39   1st Qu.:0.3625   1st Qu.: 9.625
 Median :205.0        Median :0.9987   Median :3.42   Median :0.3650   Median : 9.750
 Mean   :205.0        Mean   :0.9987   Mean   :3.42   Mean   :0.3650   Mean   : 9.750
 3rd Qu.:208.5        3rd Qu.:0.9988   3rd Qu.:3.45   3rd Qu.:0.3675   3rd Qu.: 9.875
 Max.   :212.0        Max.   :0.9988   Max.   :3.48   Max.   :0.3700   Max.   :10.000
------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
groups_single: 3
 fixed.acidity volatile.acidity  citric.acid   residual.sugar    chlorides    free.sulfur.dioxide
 Min.   :6.1   Min.   :0.36     Min.   :0.41   Min.   :19.35   Min.   :0.07   Min.   :67
 1st Qu.:6.1   1st Qu.:0.36     1st Qu.:0.41   1st Qu.:19.35   1st Qu.:0.07   1st Qu.:67
 Median :6.1   Median :0.36     Median :0.41   Median :19.35   Median :0.07   Median :67
 Mean   :6.1   Mean   :0.36     Mean   :0.41   Mean   :19.35   Mean   :0.07   Mean   :67
 3rd Qu.:6.1   3rd Qu.:0.36     3rd Qu.:0.41   3rd Qu.:19.35   3rd Qu.:0.07   3rd Qu.:67
 Max.   :6.1   Max.   :0.36     Max.   :0.41   Max.   :19.35   Max.   :0.07   Max.   :67
 total.sulfur.dioxide    density             pH          sulphates       alcohol
 Min.   :207           Min.   :1.001   Min.   :3.39   Min.   :0.53   Min.   :9.1
 1st Qu.:207           1st Qu.:1.001   1st Qu.:3.39   1st Qu.:0.53   1st Qu.:9.1
 Median :207           Median :1.001   Median :3.39   Median :0.53   Median :9.1
 Mean   :207           Mean   :1.001   Mean   :3.39   Mean   :0.53   Mean   :9.1
 3rd Qu.:207           3rd Qu.:1.001   3rd Qu.:3.39   3rd Qu.:0.53   3rd Qu.:9.1
 Max.   :207           Max.   :1.001   Max.   :3.39   Max.   :0.53   Max.   :9.1
--------------------------------------------------------------------------------
groups_single: 4
 fixed.acidity volatile.acidity  citric.acid   residual.sugar    chlorides    free.sulfur.dioxide
 Min.   :7     Min.   :0.45     Min.   :0.34   Min.   :19.8   Min.   :0.04   Min.   :12
 1st Qu.:7     1st Qu.:0.45     1st Qu.:0.34   1st Qu.:19.8   1st Qu.:0.04   1st Qu.:12
 Median :7     Median :0.45     Median :0.34   Median :19.8   Median :0.04   Median :12
 Mean   :7     Mean   :0.45     Mean   :0.34   Mean   :19.8   Mean   :0.04   Mean   :12
 3rd Qu.:7     3rd Qu.:0.45     3rd Qu.:0.34   3rd Qu.:19.8   3rd Qu.:0.04   3rd Qu.:12
 Max.   :7     Max.   :0.45     Max.   :0.34   Max.   :19.8   Max.   :0.04   Max.   :12
 total.sulfur.dioxide    density             pH          sulphates       alcohol
 Min.   :67            Min.   :0.9976   Min.   :3.07   Min.   :0.38   Min.   :11
 1st Qu.:67            1st Qu.:0.9976   1st Qu.:3.07   1st Qu.:0.38   1st Qu.:11
 Median :67            Median :0.9976   Median :3.07   Median :0.38   Median :11
 Mean   :67            Mean   :0.9976   Mean   :3.07   Mean   :0.38   Mean   :11
 3rd Qu.:67            3rd Qu.:0.9976   3rd Qu.:3.07   3rd Qu.:0.38   3rd Qu.:11
 Max.   :67            Max.   :0.9976   Max.   :3.07   Max.   :0.38   Max.   :11
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
groups_single: 5
 fixed.acidity volatile.acidity  citric.acid   residual.sugar    chlorides     free.sulfur.dioxide
 Min.   :4.9   Min.   :0.47     Min.   :0.17   Min.   :1.9    Min.   :0.035   Min.   :60
 1st Qu.:4.9   1st Qu.:0.47     1st Qu.:0.17   1st Qu.:1.9    1st Qu.:0.035   1st Qu.:60
 Median :4.9   Median :0.47     Median :0.17   Median :1.9    Median :0.035   Median :60
 Mean   :4.9   Mean   :0.47     Mean   :0.17   Mean   :1.9    Mean   :0.035   Mean   :60
 3rd Qu.:4.9   3rd Qu.:0.47     3rd Qu.:0.17   3rd Qu.:1.9    3rd Qu.:0.035   3rd Qu.:60
 Max.   :4.9   Max.   :0.47     Max.   :0.17   Max.   :1.9    Max.   :0.035   Max.   :60
 total.sulfur.dioxide    density             pH          sulphates       alcohol
 Min.   :148           Min.   :0.9896   Min.   :3.27   Min.   :0.35   Min.   :11.5
 1st Qu.:148           1st Qu.:0.9896   1st Qu.:3.27   1st Qu.:0.35   1st Qu.:11.5
 Median :148           Median :0.9896   Median :3.27   Median :0.35   Median :11.5
 Mean   :148           Mean   :0.9896   Mean   :3.27   Mean   :0.35   Mean   :11.5
 3rd Qu.:148           3rd Qu.:0.9896   3rd Qu.:3.27   3rd Qu.:0.35   3rd Qu.:11.5
 Max.   :148           Max.   :0.9896   Max.   :3.27   Max.   :0.35   Max.   :11.5
> 
```

Let’s investigate for k = 20.
Fit model, display dendogram and draw dendogram with red borders around the 20 clusters.

```
> windows(8,8) #To plot in a new window
> plot(fit_single)
> rect.hclust(fit_single, k=20, border="red")
>
```



Even with k = 20, we still can get proper separation between cluster. We suspect that the single method might not be appropriate for white wine data.

The two clusters are not well separated, and it feels like there are multitude clusters generated by this algorithm. It is possible that the number of clusters is higher than suspected.
Let’s cut the tree into 20 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_single <- cutree(fit_single, k=20)
> table(groups_single)
groups_single
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
3993    1    1    1    1    1    1    1    1    1    1    1    2    1    1    1    3    1    1    1
>
```

The above classification also validate the cluster with k = 20 above has not clustering the whitewine data well. Almost all the elements in the data is clustered to 1

Let’s check the consistency of the results against column 12:

```
> table(white_wine_output,groups_single)
                 groups_single
white_wine_output    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18
                3    8    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                4   89    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                5 1102    1    0    0    0    0    0    1    1    1    0    0    0    0    0    0    0    0
                6 1851    0    0    0    1    1    1    0    0    0    1    1    2    1    1    0    3    1
                7  796    0    1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                8  143    0    0    1    0    0    0    0    0    0    0    0    0    0    0    1    0    0
                9    4    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
                 groups_single
white_wine_output   19   20
                3    0    0
                4    0    0
                5    0    0
                6    1    1
                7    0    0
                8    0    0
                9    0    0
>
```

Almost all the whitewine quality are found in cluster 1. As seen in the dendogram plot, most of the elements are found in cluster 1.

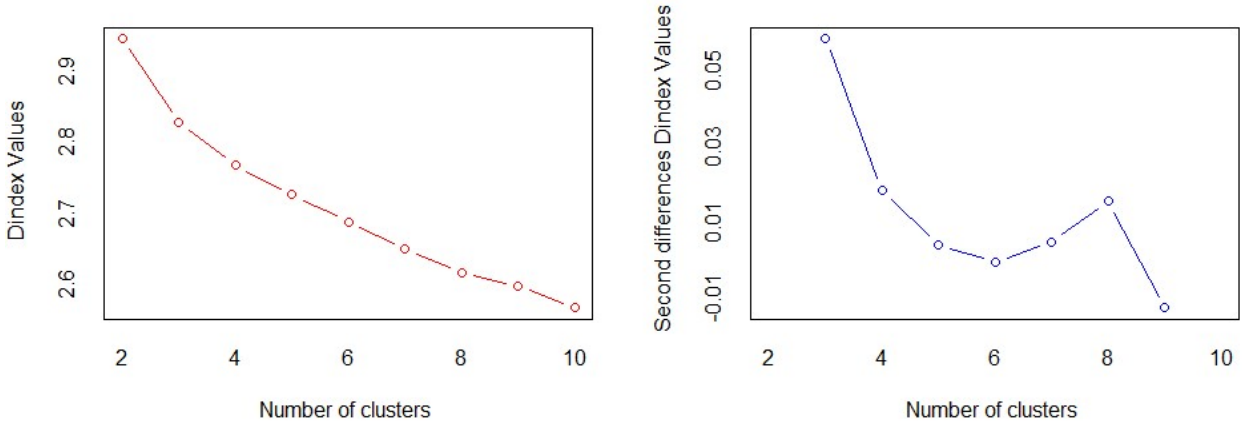
Next, let's find the mean of each attribute for k = 20.

```
> round(aggregate(white_wine_input,by=list(groups_single),FUN=mean), 2)
   Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1        1          6.81             0.26        0.32          6.36      0.04                34.72
2        2          8.30             0.41        0.51          2.00      0.05                11.00
3        3          6.50             0.12        0.29          1.95      0.04                73.00
4        4          7.80             0.21        0.39          1.80      0.03                62.00
5        5          7.60             0.44        0.44         14.50      0.04                68.00
6        6          8.70             0.34        0.46         13.80      0.06                68.00
7        7          8.70             0.45        0.40          1.50      0.07                17.00
8        8          5.70             0.46        0.46          1.40      0.04                31.00
9        9          8.40             0.22        0.28         18.80      0.03                55.00
10      10          6.10             0.36        0.41         19.35      0.07                67.00
11      11          7.70             0.39        0.49          7.70      0.04                11.00
12      12          8.60             0.33        0.34         11.80      0.06                42.00
13      13          7.00             0.45        0.34         19.80      0.04                12.00
14      14          5.80             0.39        0.47          7.50      0.03                12.00
15      15          5.90             0.24        0.12          1.40      0.04                60.00
16      16          5.00             0.29        0.54          5.70      0.04                54.00
17      17          5.80             0.18        0.22         12.70      0.06                30.00
18      18          6.30             0.48        0.48          1.80      0.04                35.00
19      19          6.10             0.41        0.20         12.60      0.03                54.00
20      20          4.90             0.47        0.17          1.90      0.04                60.00
   total.sulfur.dioxide density   pH sulphates alcohol
1                136.88    0.99 3.19      0.48   10.58
2                207.00    0.99 3.02      0.55   11.40
3                166.00    0.99 3.12      0.25   12.90
4                180.00    0.99 3.09      0.75   12.60
5                212.00    1.00 3.48      0.36   10.00
6                198.00    1.00 3.36      0.37    9.50
7                100.00    1.00 3.27      0.57   10.10
8                169.00    0.99 3.13      0.47    8.80
9                130.00    1.00 2.96      0.35   11.60
10               207.00    1.00 3.39      0.53    9.10
11               110.00    1.00 3.33      0.76   10.00
12               240.00    1.00 3.17      0.52   10.00
13                67.00    1.00 3.07      0.38   11.00
14                88.00    0.99 3.38      0.45   14.00
15               247.00    0.99 3.34      0.44    9.60
16               155.00    0.99 3.27      0.34   12.90
17               190.67    1.00 3.32      0.41   11.90
18                96.00    0.99 3.49      0.74   12.20
19               136.00    1.00 2.91      0.43   10.60
20               148.00    0.99 3.27      0.35   11.50
>
```

There's are some differences in the quality and chemical properties between the clusters. It feels like the single method is not clustering the wine data very well. Let's model the complete model to see how it works with the whitewine dataset.

Complete Method:
Next we'll use hierarchical clustering, with the option complete. But before that we need to determine what number of clusters to be later considered.

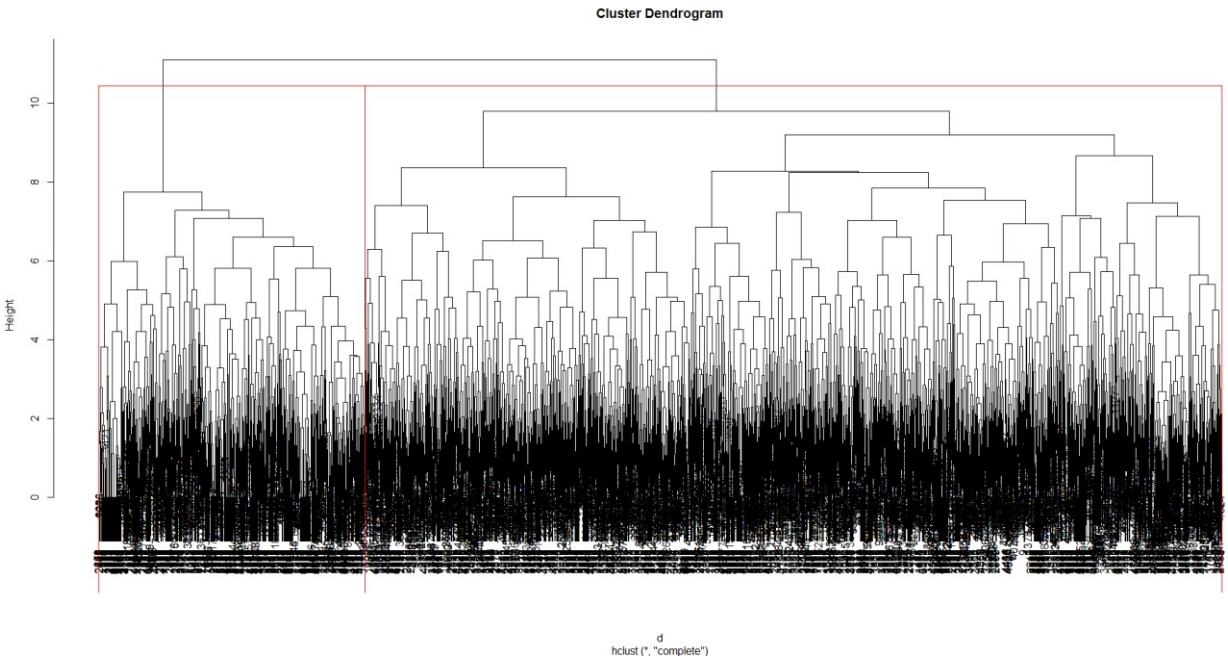
```
> NbClust(white_wine_train, distance = "euclidean", min.nc=2, max.nc=10,
+         method = "complete", index = "all", alphaBeale = 0.1)
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
                In the plot of D index, we seek a significant knee (the significant peak in Dindex
                second differences plot) that corresponds to a significant increase of the value of
                the measure.

*******************************************************************
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 10 proposed 3 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2
```



The complete method also propose two as the best cluster. Meanwhile we'll investigate this further.

Fit model, display dendogram and draw dendogram with red borders around the 2 clusters.

```
> fit_complete <- hclust(d, method="complete")
> windows(8,8) #To plot in a new window
> plot(fit_complete)
> rect.hclust(fit_complete, k=2, border="red")
>
```



The two clusters are now getting clearer but we can also see a big difference in clusters. There is a possibility that the number of clusters is higher than suspected. Let's cut the tree into 2 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_complete <- cutree(fit_complete, k=2)
> table(groups_complete)
groups_complete
   1    2
 951 3064
>
```

More than 75% of the elements are assigned to cluster 2.

Let's check the consistency of the results against column 12:

```
> table(white_wine_output,groups_complete)
                 groups_complete
white_wine_output    1    2
                3    2    6
                4   14   75
                5  410  696
                6  400 1466
                7  108  689
                8   17  128
                9    0    4
>
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1 and 2 clusters.

Next, let's find the mean of each attribute for k = 2.

```
> round(aggregate(white_wine_input,by=list(groups_complete),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1          7.01             0.27        0.32          12.34      0.05                43.84
2       2          6.75             0.26        0.33           4.53      0.04                31.93
  total.sulfur.dioxide density   pH sulphates alcohol
1               173.52    1.00 3.17      0.50    9.41
2               125.66    0.99 3.19      0.48   10.95
>
```

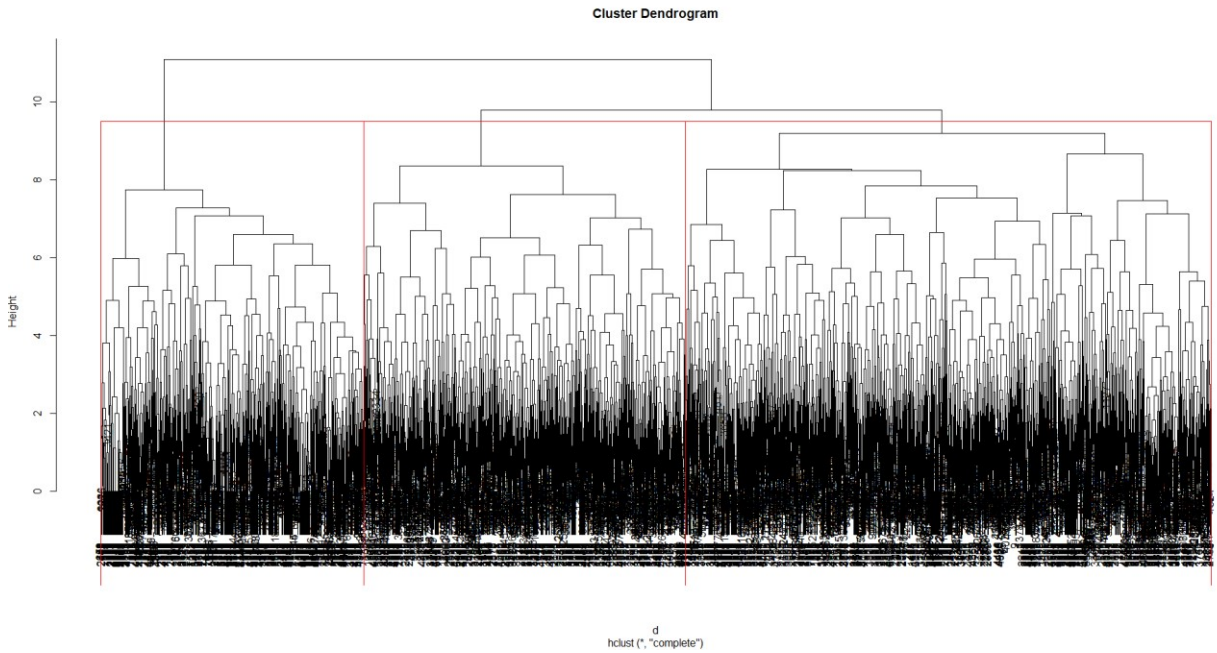
There's are some differences in the quality and chemical properties between the clusters.

Let's look at the 2 clusters:

```
> by(white_wine_input, groups_complete, summary)
groups_complete: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :5.700   Min.   :0.1050   Min.   :0.1200   Min.   : 2.10   Min.   :0.02500   Min.   : 6.00
 1st Qu.:6.600   1st Qu.:0.2200   1st Qu.:0.2600   1st Qu.: 9.20   1st Qu.:0.04400   1st Qu.:34.00
 Median :7.000   Median :0.2600   Median :0.3100   Median :12.70   Median :0.04900   Median :44.50
 Mean   :7.007   Mean   :0.2712   Mean   :0.3217   Mean   :12.34   Mean   :0.04882   Mean   :43.84
 3rd Qu.:7.400   3rd Qu.:0.3100   3rd Qu.:0.3700   3rd Qu.:14.80   3rd Qu.:0.05400   3rd Qu.:53.00
 Max.   :8.800   Max.   :0.4800   Max.   :0.5600   Max.   :22.00   Max.   :0.07000   Max.   :78.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 82.0        Min.   :0.9930   Min.   :2.86   Min.   :0.3100   Min.   : 8.500
 1st Qu.:149.0        1st Qu.:0.9963   1st Qu.:3.08   1st Qu.:0.4400   1st Qu.: 9.000
 Median :173.0        Median :0.9977   Median :3.16   Median :0.5000   Median : 9.300
 Mean   :173.5        Mean   :0.9976   Mean   :3.17   Mean   :0.5001   Mean   : 9.414
 3rd Qu.:197.0        3rd Qu.:0.9986   3rd Qu.:3.26   3rd Qu.:0.5450   3rd Qu.: 9.800
 Max.   :255.0        Max.   :1.0020   Max.   :3.56   Max.   :0.7600   Max.   :11.100
------------------------------------------------------------------------------
groups_complete: 2
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.080    Min.   :0.1000   Min.   : 0.600   Min.   :0.0150   Min.   : 2.00
 1st Qu.:6.200   1st Qu.:0.200    1st Qu.:0.2700   1st Qu.: 1.500   1st Qu.:0.0340   1st Qu.:22.00
 Median :6.700   Median :0.260    Median :0.3200   Median : 3.100   Median :0.0400   Median :31.00
 Mean   :6.753   Mean   :0.262    Mean   :0.3251   Mean   : 4.526   Mean   :0.0403   Mean   :31.93
 3rd Qu.:7.200   3rd Qu.:0.310    3rd Qu.:0.3700   3rd Qu.: 6.800   3rd Qu.:0.0470   3rd Qu.:41.00
 Max.   :8.800   Max.   :0.485    Max.   :0.5700   Max.   :20.800   Max.   :0.0710   Max.   :80.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21.0        Min.   :0.9871   Min.   :2.820   Min.   :0.2200   Min.   : 8.40
 1st Qu.:100.0        1st Qu.:0.9911   1st Qu.:3.100   1st Qu.:0.4000   1st Qu.:10.00
 Median :122.0        Median :0.9926   Median :3.190   Median :0.4600   Median :10.90
 Mean   :125.7        Mean   :0.9927   Mean   :3.193   Mean   :0.4753   Mean   :10.95
 3rd Qu.:148.0        3rd Qu.:0.9942   3rd Qu.:3.290   3rd Qu.:0.5400   3rd Qu.:11.80
 Max.   :247.0        Max.   :1.0000   Max.   :3.560   Max.   :0.7600   Max.   :14.20
>
```

Fit model, display dendogram and draw dendogram with red borders around the 3 clusters.

```
> windows(8,8) #To plot in a new window
> plot(fit_complete)
> rect.hclust(fit_complete, k=3, border="red")
>
```



The two clusters are now getting clearer but we can also see a big difference in clusters. There is a possibility that the number of clusters is higher than suspected.

Let's cut the tree into 3 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_complete <- cutree(fit_complete, k=3)
> table(groups_complete)
groups_complete
   1    2    3
 951 1901 1163
> |
```

The elements are distributed among the 3 clusters with cluster 3 having the highest number of elements.

Let's check the consistency of the results against column 12:

```
> table(white_wine_output,groups_complete)
                 groups_complete
white_wine_output   1   2   3
                3   2   4   2
                4  14  57  18
                5 410 575 121
                6 400 930 536
                7 108 276 413
                8  17  59  69
                9   0   0   4
> |
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1, 2 and 3 clusters, with cluster 3 having the highest number of elements.

Next, let's find the mean of each attribute for k = 3.

```
> round(aggregate(white_wine_input,by=list(groups_complete),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1          7.01             0.27        0.32          12.34      0.05                43.84
2       2          6.91             0.26        0.33           5.18      0.04                33.80
3       3          6.49             0.27        0.32           3.47      0.03                28.87
  total.sulfur.dioxide density   pH sulphates alcohol
1               173.52    1.00 3.17      0.50    9.41
2               135.34    0.99 3.17      0.48   10.45
3               109.84    0.99 3.22      0.46   11.75
>
```

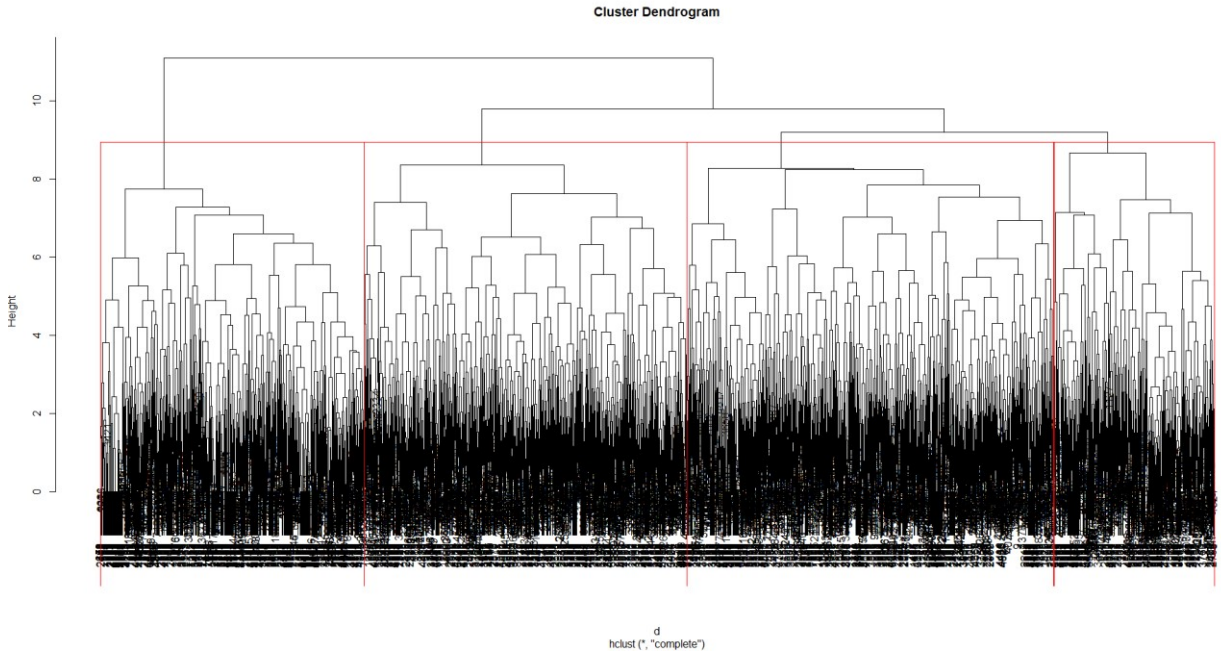
There's are some differences in the quality and chemical properties between the clusters.

Let's look at the 3 clusters:

```
> by(white_wine_input, groups_complete, summary)
groups_complete: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :5.700   Min.   :0.1050   Min.   :0.1200   Min.   : 2.10   Min.   :0.02500   Min.   : 6.00
 1st Qu.:6.600   1st Qu.:0.2200   1st Qu.:0.2600   1st Qu.: 9.20   1st Qu.:0.04400   1st Qu.:34.00
 Median :7.000   Median :0.2600   Median :0.3100   Median :12.70   Median :0.04900   Median :44.50
 Mean   :7.007   Mean   :0.2712   Mean   :0.3217   Mean   :12.34   Mean   :0.04882   Mean   :43.84
 3rd Qu.:7.400   3rd Qu.:0.3100   3rd Qu.:0.3700   3rd Qu.:14.80   3rd Qu.:0.05400   3rd Qu.:53.00
 Max.   :8.800   Max.   :0.4800   Max.   :0.5600   Max.   :22.00   Max.   :0.07000   Max.   :78.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 82.0        Min.   :0.9930   Min.   :2.86   Min.   :0.3100   Min.   : 8.500
 1st Qu.:149.0        1st Qu.:0.9963   1st Qu.:3.08   1st Qu.:0.4400   1st Qu.: 9.000
 Median :173.0        Median :0.9977   Median :3.16   Median :0.5000   Median : 9.300
 Mean   :173.5        Mean   :0.9976   Mean   :3.17   Mean   :0.5001   Mean   : 9.414
 3rd Qu.:197.0        3rd Qu.:0.9986   3rd Qu.:3.26   3rd Qu.:0.5450   3rd Qu.: 9.800
 Max.   :255.0        Max.   :1.0020   Max.   :3.56   Max.   :0.7600   Max.   :11.100
------------------------------------------------------------------------------
groups_complete: 2
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.600   Min.   :0.01500   Min.   : 2.0
 1st Qu.:6.400   1st Qu.:0.2000   1st Qu.:0.2700   1st Qu.: 1.600   1st Qu.:0.03700   1st Qu.:22.0
 Median :6.800   Median :0.2500   Median :0.3200   Median : 4.500   Median :0.04400   Median :32.0
 Mean   :6.915   Mean   :0.2552   Mean   :0.3305   Mean   : 5.175   Mean   :0.04383   Mean   :33.8
 3rd Qu.:7.400   3rd Qu.:0.3000   3rd Qu.:0.3900   3rd Qu.: 7.800   3rd Qu.:0.05000   3rd Qu.:45.0
 Max.   :8.800   Max.   :0.4800   Max.   :0.5700   Max.   :20.800   Max.   :0.07100   Max.   :80.0
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 25.0        Min.   :0.9877   Min.   :2.830   Min.   :0.2500   Min.   : 8.40
 1st Qu.:110.0        1st Qu.:0.9923   1st Qu.:3.080   1st Qu.:0.4000   1st Qu.: 9.70
 Median :131.0        Median :0.9936   Median :3.160   Median :0.4700   Median :10.40
 Mean   :135.3        Mean   :0.9937   Mean   :3.173   Mean   :0.4841   Mean   :10.45
 3rd Qu.:161.0        3rd Qu.:0.9950   3rd Qu.:3.250   3rd Qu.:0.5500   3rd Qu.:11.10
 Max.   :247.0        Max.   :1.0000   Max.   :3.560   Max.   :0.7600   Max.   :13.30
------------------------------------------------------------------------------
groups_complete: 3
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.700   Min.   :0.01500   Min.   : 3.00
 1st Qu.:6.000   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.500   1st Qu.:0.03000   1st Qu.:21.00
 Median :6.500   Median :0.2700   Median :0.3100   Median : 2.350   Median :0.03500   Median :29.00
 Mean   :6.489   Mean   :0.2731   Mean   :0.3163   Mean   : 3.466   Mean   :0.03452   Mean   :28.87
 3rd Qu.:6.900   3rd Qu.:0.3300   3rd Qu.:0.3500   3rd Qu.: 4.800   3rd Qu.:0.03900   3rd Qu.:37.00
 Max.   :8.600   Max.   :0.4850   Max.   :0.5700   Max.   :15.550   Max.   :0.05500   Max.   :67.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21.0        Min.   :0.9871   Min.   :2.820   Min.   :0.2200   Min.   : 8.50
 1st Qu.: 91.0        1st Qu.:0.9900   1st Qu.:3.130   1st Qu.:0.3900   1st Qu.:11.10
 Median :109.0        Median :0.9910   Median :3.230   Median :0.4500   Median :11.80
 Mean   :109.8        Mean   :0.9912   Mean   :3.224   Mean   :0.4609   Mean   :11.75
 3rd Qu.:128.0        3rd Qu.:0.9920   3rd Qu.:3.320   3rd Qu.:0.5200   3rd Qu.:12.50
 Max.   :191.0        Max.   :0.9982   Max.   :3.560   Max.   :0.7600   Max.   :14.20
>
```

Fit model, display dendogram and draw dendogram with red borders around the 4 clusters.

```
> windows(8,8) #To plot in a new window
> plot(fit_complete)
> rect.hclust(fit_complete, k=4, border="red")
>
```



The two clusters are now getting clearer but we can also see a big difference in clusters. There is a possibility that the number of clusters is higher than suspected.

Let’s cut the tree into 4 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> table(groups_complete)
groups_complete
   1    2    3    4
 951 1323  578 1163
>
```

The elements are distributed among the 3 clusters with cluster 2 and 4 having the highest number of elements.

Let’s check the consistency of the results against column 12:

```
> table(white_wine_output,groups_complete)
                 groups_complete
white_wine_output   1   2   3   4
                3   2   3   1   2
                4  14  41  16  18
                5 410 337 238 121
                6 400 657 273 536
                7 108 239  37 413
                8  17  46  13  69
                9   0   0   0   4
> 
```

We can see that quality 5 to 8 has the most proportion of the clusters. It almost evenly distributes the quality with its 1, 2, 3 and 4 clusters.

Next, let’s find the mean of each attribute for k = 4.

```
> round(aggregate(white_wine_input,by=list(groups_complete),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1          7.01             0.27        0.32          12.34      0.05                43.84
2       2          7.14             0.25        0.34           4.68      0.04                29.13
3       3          6.40             0.27        0.31           6.31      0.05                44.48
4       4          6.49             0.27        0.32           3.47      0.03                28.87
  total.sulfur.dioxide density   pH sulphates alcohol
1               173.52    1.00 3.17      0.50    9.41
2               125.96    0.99 3.16      0.50   10.64
3               156.79    0.99 3.19      0.45   10.02
4               109.84    0.99 3.22      0.46   11.75
> 
```

There's are some differences in the quality and chemical properties between the clusters.

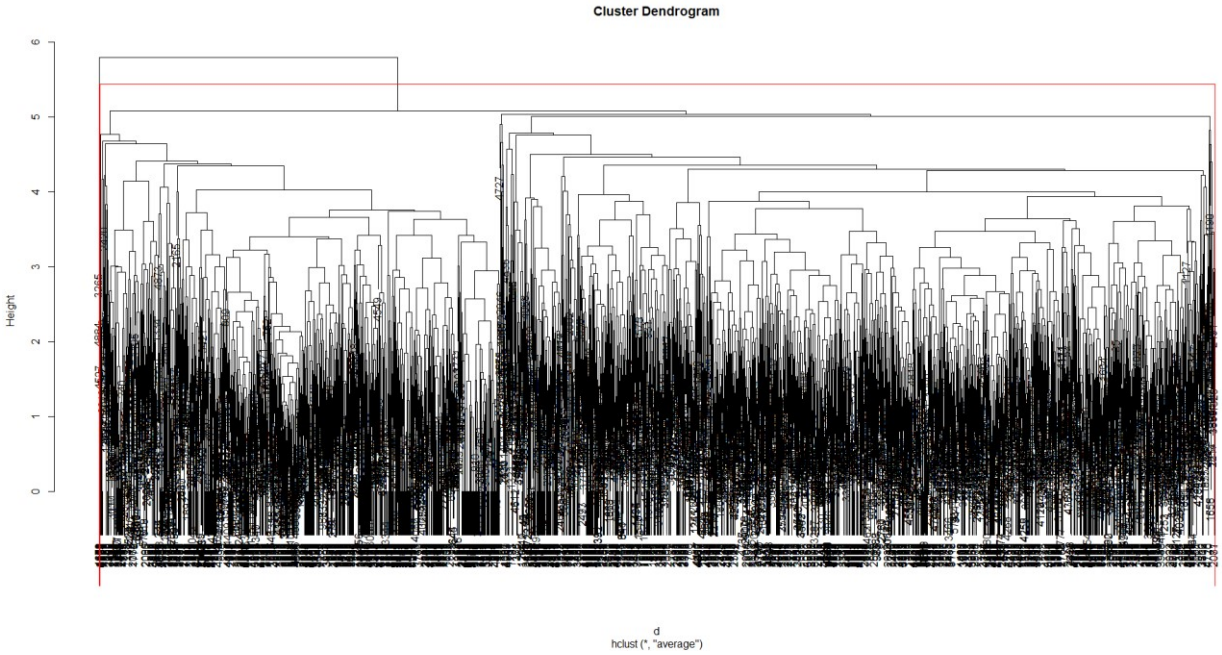
Let's look at the 4 clusters:

```
> by(white_wine_input, groups_complete, summary)
groups_complete: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :5.700   Min.   :0.1050   Min.   :0.1200   Min.   : 2.10   Min.   :0.02500   Min.   : 6.00
 1st Qu.:6.600   1st Qu.:0.2200   1st Qu.:0.2600   1st Qu.: 9.20   1st Qu.:0.04400   1st Qu.:34.00
 Median :7.000   Median :0.2600   Median :0.3100   Median :12.70   Median :0.04900   Median :44.50
 Mean   :7.007   Mean   :0.2712   Mean   :0.3217   Mean   :12.34   Mean   :0.04882   Mean   :43.84
 3rd Qu.:7.400   3rd Qu.:0.3100   3rd Qu.:0.3700   3rd Qu.:14.80   3rd Qu.:0.05400   3rd Qu.:53.00
 Max.   :8.800   Max.   :0.4800   Max.   :0.5600   Max.   :22.00   Max.   :0.07000   Max.   :78.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 82.0        Min.   :0.9930   Min.   :2.86   Min.   :0.3100   Min.   : 8.500
 1st Qu.:149.0        1st Qu.:0.9963   1st Qu.:3.08   1st Qu.:0.4400   1st Qu.: 9.000
 Median :173.0        Median :0.9977   Median :3.16   Median :0.5000   Median : 9.300
 Mean   :173.5        Mean   :0.9976   Mean   :3.17   Mean   :0.5001   Mean   : 9.414
 3rd Qu.:197.0        3rd Qu.:0.9986   3rd Qu.:3.26   3rd Qu.:0.5450   3rd Qu.: 9.800
 Max.   :255.0        Max.   :1.0020   Max.   :3.56   Max.   :0.7600   Max.   :11.100
------------------------------------------------------------------------------------------
groups_complete: 2
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :5.200   Min.   :0.0800   Min.   :0.1000   Min.   : 0.600   Min.   :0.01500   Min.   : 2.00
 1st Qu.:6.600   1st Qu.:0.2000   1st Qu.:0.2800   1st Qu.: 1.500   1st Qu.:0.03600   1st Qu.:20.00
 Median :7.100   Median :0.2400   Median :0.3300   Median : 2.700   Median :0.04200   Median :28.00
 Mean   :7.142   Mean   :0.2501   Mean   :0.3388   Mean   : 4.681   Mean   :0.04286   Mean   :29.13
 3rd Qu.:7.600   3rd Qu.:0.3000   3rd Qu.:0.3800   3rd Qu.: 7.100   3rd Qu.:0.05000   3rd Qu.:37.00
 Max.   :8.800   Max.   :0.4700   Max.   :0.5700   Max.   :20.800   Max.   :0.07000   Max.   :73.50
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 25.0        Min.   :0.9884   Min.   :2.830   Min.   :0.250   Min.   : 8.70
 1st Qu.:103.0        1st Qu.:0.9921   1st Qu.:3.060   1st Qu.:0.410   1st Qu.:10.00
 Median :123.0        Median :0.9933   Median :3.150   Median :0.490   Median :10.60
 Mean   :126.0        Mean   :0.9934   Mean   :3.165   Mean   :0.497   Mean   :10.64
 3rd Qu.:150.0        3rd Qu.:0.9945   3rd Qu.:3.260   3rd Qu.:0.570   3rd Qu.:11.20
 Max.   :218.5        Max.   :0.9989   Max.   :3.540   Max.   :0.760   Max.   :13.30
------------------------------------------------------------------------------------------
groups_complete: 3
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.1050   Min.   :0.1000   Min.   : 0.800   Min.   :0.02500   Min.   : 5.00
 1st Qu.:6.100   1st Qu.:0.2200   1st Qu.:0.2300   1st Qu.: 2.525   1st Qu.:0.04100   1st Qu.:34.00
 Median :6.400   Median :0.2675   Median :0.2900   Median : 6.450   Median :0.04600   Median :47.00
 Mean   :6.396   Mean   :0.2669   Mean   :0.3115   Mean   : 6.306   Mean   :0.04604   Mean   :44.48
 3rd Qu.:6.700   3rd Qu.:0.3200   3rd Qu.:0.4000   3rd Qu.: 8.400   3rd Qu.:0.05100   3rd Qu.:57.00
 Max.   :8.100   Max.   :0.4800   Max.   :0.5700   Max.   :17.050   Max.   :0.07100   Max.   :80.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 63.0        Min.   :0.9877   Min.   :2.860   Min.   :0.2500   Min.   : 8.40
 1st Qu.:128.0        1st Qu.:0.9929   1st Qu.:3.140   1st Qu.:0.3900   1st Qu.: 9.40
 Median :153.0        Median :0.9945   Median :3.180   Median :0.4500   Median : 9.80
 Mean   :156.8        Mean   :0.9942   Mean   :3.193   Mean   :0.4546   Mean   :10.02
 3rd Qu.:186.0        3rd Qu.:0.9956   3rd Qu.:3.240   3rd Qu.:0.5000   3rd Qu.:10.50
 Max.   :247.0        Max.   :1.0000   Max.   :3.560   Max.   :0.7600   Max.   :13.10
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
groups_complete: 4
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.700   Min.   :0.01500   Min.   : 3.00
 1st Qu.:6.000   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.500   1st Qu.:0.03000   1st Qu.:21.00
 Median :6.500   Median :0.2700   Median :0.3100   Median : 2.350   Median :0.03500   Median :29.00
 Mean   :6.489   Mean   :0.2731   Mean   :0.3163   Mean   : 3.466   Mean   :0.03452   Mean   :28.87
 3rd Qu.:6.900   3rd Qu.:0.3300   3rd Qu.:0.3500   3rd Qu.: 4.800   3rd Qu.:0.03900   3rd Qu.:37.00
 Max.   :8.600   Max.   :0.4850   Max.   :0.5700   Max.   :15.550   Max.   :0.05500   Max.   :67.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21.0        Min.   :0.9871   Min.   :2.820   Min.   :0.2200   Min.   : 8.50
 1st Qu.: 91.0        1st Qu.:0.9900   1st Qu.:3.130   1st Qu.:0.3900   1st Qu.:11.10
 Median :109.0        Median :0.9910   Median :3.230   Median :0.4500   Median :11.80
 Mean   :109.8        Mean   :0.9912   Mean   :3.224   Mean   :0.4609   Mean   :11.75
 3rd Qu.:128.0        3rd Qu.:0.9920   3rd Qu.:3.320   3rd Qu.:0.5200   3rd Qu.:12.50
 Max.   :191.0        Max.   :0.9982   Max.   :3.560   Max.   :0.7600   Max.   :14.20
```

Average Method:
Next we'll use hierarchical clustering, with the option average. The dendlist object proposed k = 2 as the method for all cluster method. Therefore, we'll start the analysis with k = 2 and higher for rest of the methods that will be used to fit our model.

Fit model, display dendogram and draw dendogram with red borders around the 2 clusters.

```
> fit_average <- hclust(d, method="average")
> windows(8,8) #To plot in a new window
> windows(8,8) #To plot in a new window
> plot(fit_average)
> rect.hclust(fit_average, k=2, border="red")
>
```



The two clusters are not clearer and we can also see a big difference in clusters. There is a possibility that the number of clusters is higher than suspected.

Let's cut the tree into 2 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_average <- cutree(fit_average, k=2)
> table(groups_average)
groups_average
   1    2
4011    4
>
```

More than 75% of the elements are assigned to cluster 2.

Let's check the consistency of the results against column 12:

```
> table(white_wine_output,groups_average)
                 groups_average
white_wine_output    1    2
                3    8    0
                4   89    0
                5 1104    2
                6 1864    2
                7  797    0
                8  145    0
                9    4    0
> |
```

Almost all the whitewine quality are found in cluster 1. As seen in the dendogram plot, most of the elements are found in cluster 1.

Next, let's find the mean of each attribute for k = 2.

```
> round(aggregate(white_wine_input,by=list(groups_average),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1          6.82             0.26        0.32           6.38      0.04                34.72
2       2          5.15             0.34        0.14           1.48      0.04                64.50
  total.sulfur.dioxide density   pH sulphates alcohol
1               136.95    0.99 3.19      0.48   10.58
2               182.75    0.99 3.39      0.43   10.51
> |
```

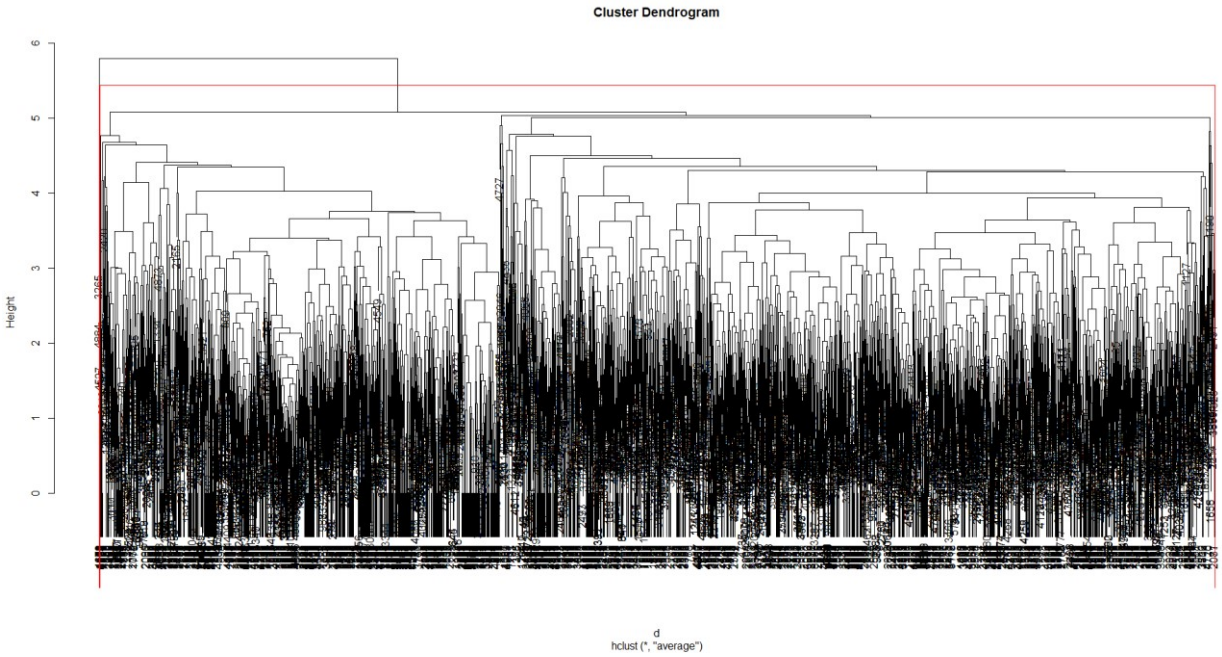
There's are some differences in the quality and chemical properties between the clusters.

Let's look at the 2 clusters:

```
> by(white_wine_input, groups_average, summary)
groups_average: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.600   Min.   :0.01500   Min.   : 2.00
 1st Qu.:6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.800   1st Qu.:0.03500   1st Qu.:24.00
 Median :6.800   Median :0.2600   Median :0.3100   Median : 5.200   Median :0.04200   Median :34.00
 Mean   :6.815   Mean   :0.2641   Mean   :0.3245   Mean   : 6.381   Mean   :0.04232   Mean   :34.72
 3rd Qu.:7.300   3rd Qu.:0.3100   3rd Qu.:0.3700   3rd Qu.: 9.700   3rd Qu.:0.04900   3rd Qu.:45.00
 Max.   :8.800   Max.   :0.4850   Max.   :0.5700   Max.   :22.000   Max.   :0.07100   Max.   :80.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21          Min.   :0.9871   Min.   :2.820   Min.   :0.2200   Min.   : 8.40
 1st Qu.:107          1st Qu.:0.9916   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50
 Median :132          Median :0.9935   Median :3.180   Median :0.4700   Median :10.50
 Mean   :137          Mean   :0.9939   Mean   :3.187   Mean   :0.4812   Mean   :10.58
 3rd Qu.:166          3rd Qu.:0.9959   3rd Qu.:3.280   3rd Qu.:0.5400   3rd Qu.:11.40
 Max.   :255          Max.   :1.0020   Max.   :3.560   Max.   :0.7600   Max.   :14.20
------------------------------------------------------------------------------
groups_average: 2
 fixed.acidity  volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :4.90   Min.   :0.2400   Min.   :0.1200   Min.   :1.300   Min.   :0.0350   Min.   :60.0
 1st Qu.:4.90   1st Qu.:0.3113   1st Qu.:0.1350   1st Qu.:1.300   1st Qu.:0.0350   1st Qu.:60.0
 Median :4.90   Median :0.3350   Median :0.1400   Median :1.350   Median :0.0355   Median :64.5
 Mean   :5.15   Mean   :0.3450   Mean   :0.1425   Mean   :1.475   Mean   :0.0355   Mean   :64.5
 3rd Qu.:5.15   3rd Qu.:0.3688   3rd Qu.:0.1475   3rd Qu.:1.525   3rd Qu.:0.0360   3rd Qu.:69.0
 Max.   :5.90   Max.   :0.4700   Max.   :0.1700   Max.   :1.900   Max.   :0.0360   Max.   :69.0
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   :148.0        Min.   :0.9896   Min.   :3.270   Min.   :0.3500   Min.   : 9.60
 1st Qu.:163.0        1st Qu.:0.9915   1st Qu.:3.322   1st Qu.:0.4175   1st Qu.:10.25
 Median :168.0        Median :0.9921   Median :3.405   Median :0.4500   Median :10.47
 Mean   :182.8        Mean   :0.9919   Mean   :3.388   Mean   :0.4275   Mean   :10.51
 3rd Qu.:187.8        3rd Qu.:0.9925   3rd Qu.:3.470   3rd Qu.:0.4600   3rd Qu.:10.72
 Max.   :247.0        Max.   :0.9936   Max.   :3.470   Max.   :0.4600   Max.   :11.50
> |
```

Fit model, display dendogram and draw dendogram with red borders around the 3 clusters.

```
> windows(8,8) #To plot in a new window
> plot(fit_average)
> rect.hclust(fit_average, k=3, border="red")
>
```



The two clusters are not well separated, and it feels like there are multitude of clusters generated on hclust. There is a possibility that the number of clusters is higher than suspected.

Let's cut the tree into 3 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_average <- cutree(fit_average, k=3)
> table(groups_average)
groups_average
   1    2    3
1439 2572    4
>
```

Clusters 2 has the highest number of elements follow by cluster 2 while cluster 3 has just 3 elemenets.

Let’s check the consistency of the results against column 12:

```
> table(white_wine_output,groups_average)
                  groups_average
white_wine_output    1    2    3
                3    2    6    0
                4   18   71    0
                5  588  516    2
                6  646 1218    2
                7  157  640    0
                8   28  117    0
                9    0    4    0
>
```

As seen in the dendogram plot, most of the elements are found in cluster 1 and 2.

Next, let’s find the mean of each attribute for k = 3.

```
> round(aggregate(white_wine_input,by=list(groups_average),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1          6.90             0.26        0.34          10.77      0.05                45.40
2       2          6.77             0.26        0.32           3.92      0.04                28.74
3       3          5.15             0.34        0.14           1.48      0.04                64.50
  total.sulfur.dioxide density   pH sulphates alcohol
1               171.26    1.00 3.16      0.50    9.59
2               117.76    0.99 3.20      0.47   11.14
3               182.75    0.99 3.39      0.43   10.51
>
```

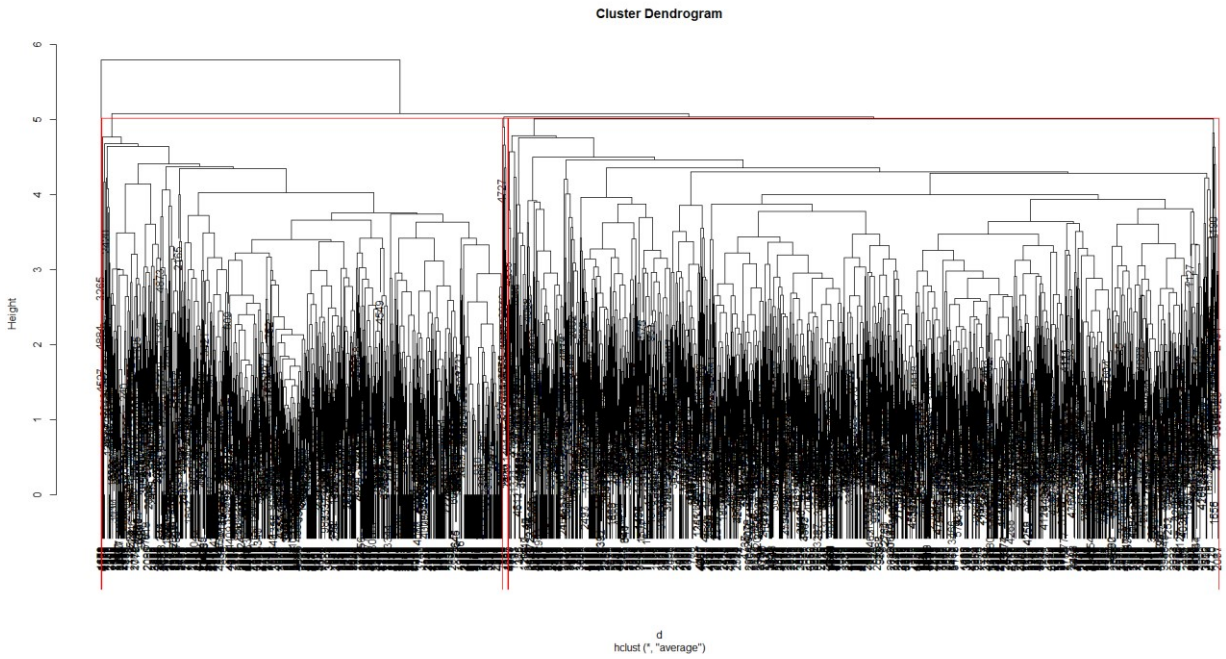
There's are some differences in the quality and chemical properties between the clusters.

Let's look at the 3 clusters:

```
> by(white_wine_input, groups_average, summary)
groups_average: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :5.300   Min.   :0.105    Min.   :0.1200   Min.   : 0.90   Min.   :0.02500   Min.   : 6.0
 1st Qu.:6.425   1st Qu.:0.220    1st Qu.:0.2700   1st Qu.: 7.70   1st Qu.:0.04300   1st Qu.:36.0
 Median :6.800   Median :0.260    Median :0.3200   Median :11.00   Median :0.04800   Median :46.0
 Mean   :6.899   Mean   :0.265    Mean   :0.3371   Mean   :10.77   Mean   :0.04797   Mean   :45.4
 3rd Qu.:7.300   3rd Qu.:0.310    3rd Qu.:0.4000   3rd Qu.:14.20   3rd Qu.:0.05300   3rd Qu.:55.0
 Max.   :8.800   Max.   :0.480    Max.   :0.5700   Max.   :22.00   Max.   :0.07000   Max.   :80.0
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 88.0        Min.   :0.9906   Min.   :2.860   Min.   :0.2900   Min.   : 8.500
 1st Qu.:146.0        1st Qu.:0.9955   1st Qu.:3.080   1st Qu.:0.4400   1st Qu.: 9.100
 Median :171.0        Median :0.9967   Median :3.150   Median :0.5000   Median : 9.500
 Mean   :171.3        Mean   :0.9967   Mean   :3.161   Mean   :0.5037   Mean   : 9.588
 3rd Qu.:195.0        3rd Qu.:0.9982   3rd Qu.:3.230   3rd Qu.:0.5600   3rd Qu.:10.000
 Max.   :255.0        Max.   :1.0020   Max.   :3.560   Max.   :0.7600   Max.   :12.600
------------------------------------------------------------------------------
groups_average: 2
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.600   Min.   :0.01500   Min.   : 2.00
 1st Qu.:6.200   1st Qu.:0.2000   1st Qu.:0.2700   1st Qu.: 1.500   1st Qu.:0.03200   1st Qu.:20.00
 Median :6.700   Median :0.2600   Median :0.3100   Median : 2.500   Median :0.03800   Median :28.00
 Mean   :6.768   Mean   :0.2636   Mean   :0.3174   Mean   : 3.925   Mean   :0.03916   Mean   :28.74
 3rd Qu.:7.300   3rd Qu.:0.3200   3rd Qu.:0.3600   3rd Qu.: 5.700   3rd Qu.:0.04500   3rd Qu.:37.00
 Max.   :8.800   Max.   :0.4850   Max.   :0.5700   Max.   :20.800   Max.   :0.07100   Max.   :75.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21.0        Min.   :0.9871   Min.   :2.820   Min.   :0.2200   Min.   : 8.40
 1st Qu.: 97.0        1st Qu.:0.9909   1st Qu.:3.100   1st Qu.:0.3900   1st Qu.:10.40
 Median :116.0        Median :0.9921   Median :3.200   Median :0.4600   Median :11.10
 Mean   :117.8        Mean   :0.9923   Mean   :3.202   Mean   :0.4686   Mean   :11.14
 3rd Qu.:138.0        3rd Qu.:0.9936   3rd Qu.:3.300   3rd Qu.:0.5300   3rd Qu.:12.00
 Max.   :218.0        Max.   :0.9994   Max.   :3.560   Max.   :0.7600   Max.   :14.20
------------------------------------------------------------------------------
groups_average: 3
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :4.90    Min.   :0.2400   Min.   :0.1200   Min.   :1.300   Min.   :0.0350   Min.   :60.0
 1st Qu.:4.90    1st Qu.:0.3113   1st Qu.:0.1350   1st Qu.:1.300   1st Qu.:0.0350   1st Qu.:60.0
 Median :4.90    Median :0.3350   Median :0.1400   Median :1.350   Median :0.0355   Median :64.5
 Mean   :5.15    Mean   :0.3450   Mean   :0.1425   Mean   :1.475   Mean   :0.0355   Mean   :64.5
 3rd Qu.:5.15    3rd Qu.:0.3688   3rd Qu.:0.1475   3rd Qu.:1.525   3rd Qu.:0.0360   3rd Qu.:69.0
 Max.   :5.90    Max.   :0.4700   Max.   :0.1700   Max.   :1.900   Max.   :0.0360   Max.   :69.0
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   :148.0        Min.   :0.9896   Min.   :3.270   Min.   :0.3500   Min.   : 9.60
 1st Qu.:163.0        1st Qu.:0.9915   1st Qu.:3.322   1st Qu.:0.4175   1st Qu.:10.25
 Median :168.0        Median :0.9921   Median :3.405   Median :0.4500   Median :10.47
 Mean   :182.8        Mean   :0.9919   Mean   :3.388   Mean   :0.4275   Mean   :10.51
 3rd Qu.:187.8        3rd Qu.:0.9925   3rd Qu.:3.470   3rd Qu.:0.4600   3rd Qu.:10.72
 Max.   :247.0        Max.   :0.9936   Max.   :3.470   Max.   :0.4600   Max.   :11.50
>
```

Fit model, display dendogram and draw dendogram with red borders around the 4 clusters.

```
> windows(8,8) #To plot in a new window
> plot(fit_average)
> rect.hclust(fit_average, k=4, border="red")
>
```



The four clusters are now getting clearer but we can also see a big difference in clusters. There is a possibility that the number of clusters is higher than suspected.

Let's cut the tree into 4 clusters and tabulate the number of elements in the resulting clusters as a frequency table.

```
> groups_average <- cutree(fit_average, k=4)
> table(groups_average)
groups_average
   1    2    3    4
1439 2552   20    4
>
```

Most elements are clustered to 2 and 3.

Let's check the consistency of the results against column 12:

```
> table(white_wine_output,groups_average)
                groups_average
white_wine_output    1    2    3    4
                3    2    5    1    0
                4   18   71    0    0
                5  588  516    0    2
                6  646 1210    8    2
                7  157  635    5    0
                8   28  111    6    0
                9    0    4    0    0
>
```

As seen in the dendogram plot, most of the elements are found in cluster 1 and 2.

Next, let's find the mean of each attribute for k = 4.

```
> round(aggregate(white_wine_input,by=list(groups_average),FUN=mean), 2)
  Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1       1          6.90             0.26        0.34          10.77      0.05                45.40
2       2          6.78             0.26        0.32           3.93      0.04                28.73
3       3          5.61             0.27        0.43           3.38      0.03                30.15
4       4          5.15             0.34        0.14           1.48      0.04                64.50
  total.sulfur.dioxide density   pH sulphates alcohol
1               171.26    1.00 3.16      0.50    9.59
2               117.73    0.99 3.20      0.47   11.13
3               121.40    0.99 3.46      0.51   11.95
4               182.75    0.99 3.39      0.43   10.51
>
```

There's are some differences in the quality and chemical properties between the clusters.

Let’s look at the 4 clusters:

```
> by(white_wine_input, groups_average, summary)
groups_average: 1
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :5.300   Min.   :0.105    Min.   :0.1200   Min.   : 0.90   Min.   :0.02500   Min.   : 6.0
 1st Qu.:6.425   1st Qu.:0.220    1st Qu.:0.2700   1st Qu.: 7.70   1st Qu.:0.04300   1st Qu.:36.0
 Median :6.800   Median :0.260    Median :0.3200   Median :11.00   Median :0.04800   Median :46.0
 Mean   :6.899   Mean   :0.265    Mean   :0.3371   Mean   :10.77   Mean   :0.04797   Mean   :45.4
 3rd Qu.:7.300   3rd Qu.:0.310    3rd Qu.:0.4000   3rd Qu.:14.20   3rd Qu.:0.05300   3rd Qu.:55.0
 Max.   :8.800   Max.   :0.480    Max.   :0.5700   Max.   :22.00   Max.   :0.07000   Max.   :80.0
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 88.0        Min.   :0.9906   Min.   :2.860   Min.   :0.2900   Min.   : 8.500
 1st Qu.:146.0        1st Qu.:0.9955   1st Qu.:3.080   1st Qu.:0.4400   1st Qu.: 9.100
 Median :171.0        Median :0.9967   Median :3.150   Median :0.5000   Median : 9.500
 Mean   :171.3        Mean   :0.9967   Mean   :3.161   Mean   :0.5037   Mean   : 9.588
 3rd Qu.:195.0        3rd Qu.:0.9982   3rd Qu.:3.230   3rd Qu.:0.5600   3rd Qu.:10.000
 Max.   :255.0        Max.   :1.0020   Max.   :3.560   Max.   :0.7600   Max.   :12.600
------------------------------------------------------------------------------
groups_average: 2
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar     chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.0800   Min.   :0.1000   Min.   : 0.600   Min.   :0.01500   Min.   : 2.00
 1st Qu.:6.200   1st Qu.:0.2000   1st Qu.:0.2700   1st Qu.: 1.500   1st Qu.:0.03300   1st Qu.:20.00
 Median :6.700   Median :0.2600   Median :0.3100   Median : 2.500   Median :0.03800   Median :28.00
 Mean   :6.777   Mean   :0.2635   Mean   :0.3165   Mean   : 3.929   Mean   :0.03924   Mean   :28.73
 3rd Qu.:7.300   3rd Qu.:0.3200   3rd Qu.:0.3600   3rd Qu.: 5.700   3rd Qu.:0.04500   3rd Qu.:37.00
 Max.   :8.800   Max.   :0.4850   Max.   :0.5700   Max.   :20.800   Max.   :0.07100   Max.   :75.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 21.0        Min.   :0.9871   Min.   :2.82   Min.   :0.2200   Min.   : 8.40
 1st Qu.: 97.0        1st Qu.:0.9909   1st Qu.:3.10   1st Qu.:0.3900   1st Qu.:10.40
 Median :116.0        Median :0.9921   Median :3.20   Median :0.4600   Median :11.10
 Mean   :117.7        Mean   :0.9923   Mean   :3.20   Mean   :0.4683   Mean   :11.13
 3rd Qu.:138.0        3rd Qu.:0.9936   3rd Qu.:3.30   3rd Qu.:0.5300   3rd Qu.:12.00
 Max.   :218.0        Max.   :0.9994   Max.   :3.56   Max.   :0.7600   Max.   :14.20
------------------------------------------------------------------------------
groups_average: 3
 fixed.acidity   volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :4.800   Min.   :0.1200   Min.   :0.2700   Min.   :1.000   Min.   :0.01500   Min.   : 5.00
 1st Qu.:5.150   1st Qu.:0.1975   1st Qu.:0.3800   1st Qu.:1.425   1st Qu.:0.02600   1st Qu.:19.75
 Median :5.600   Median :0.2650   Median :0.4500   Median :2.850   Median :0.02850   Median :24.50
 Mean   :5.605   Mean   :0.2712   Mean   :0.4305   Mean   :3.382   Mean   :0.02855   Mean   :30.15
 3rd Qu.:5.925   3rd Qu.:0.3350   3rd Qu.:0.4825   3rd Qu.:4.825   3rd Qu.:0.03000   3rd Qu.:40.00
 Max.   :6.800   Max.   :0.4800   Max.   :0.5700   Max.   :7.850   Max.   :0.04300   Max.   :64.00
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   : 88.0        Min.   :0.9871   Min.   :3.170   Min.   :0.3400   Min.   :10.30
 1st Qu.:101.0        1st Qu.:0.9899   1st Qu.:3.410   1st Qu.:0.4300   1st Qu.:11.43
 Median :113.5        Median :0.9910   Median :3.485   Median :0.4900   Median :11.75
 Mean   :121.4        Mean   :0.9909   Mean   :3.455   Mean   :0.5070   Mean   :11.95
 3rd Qu.:129.2        3rd Qu.:0.9920   3rd Qu.:3.542   3rd Qu.:0.5925   3rd Qu.:12.30
 Max.   :191.0        Max.   :0.9933   Max.   :3.560   Max.   :0.7400   Max.   :14.00
------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------
groups_average: 4
 fixed.acidity  volatile.acidity  citric.acid     residual.sugar    chlorides       free.sulfur.dioxide
 Min.   :4.90   Min.   :0.2400   Min.   :0.1200   Min.   :1.300   Min.   :0.0350   Min.   :60.0
 1st Qu.:4.90   1st Qu.:0.3113   1st Qu.:0.1350   1st Qu.:1.300   1st Qu.:0.0350   1st Qu.:60.0
 Median :4.90   Median :0.3350   Median :0.1400   Median :1.350   Median :0.0355   Median :64.5
 Mean   :5.15   Mean   :0.3450   Mean   :0.1425   Mean   :1.475   Mean   :0.0355   Mean   :64.5
 3rd Qu.:5.15   3rd Qu.:0.3688   3rd Qu.:0.1475   3rd Qu.:1.525   3rd Qu.:0.0360   3rd Qu.:69.0
 Max.   :5.90   Max.   :0.4700   Max.   :0.1700   Max.   :1.900   Max.   :0.0360   Max.   :69.0
 total.sulfur.dioxide    density             pH          sulphates         alcohol
 Min.   :148.0        Min.   :0.9896   Min.   :3.270   Min.   :0.3500   Min.   : 9.60
 1st Qu.:163.0        1st Qu.:0.9915   1st Qu.:3.322   1st Qu.:0.4175   1st Qu.:10.25
 Median :168.0        Median :0.9921   Median :3.405   Median :0.4500   Median :10.47
 Mean   :182.8        Mean   :0.9919   Mean   :3.388   Mean   :0.4275   Mean   :10.51
 3rd Qu.:187.8        3rd Qu.:0.9925   3rd Qu.:3.470   3rd Qu.:0.4600   3rd Qu.:10.72
 Max.   :247.0        Max.   :0.9936   Max.   :3.470   Max.   :0.4600   Max.   :11.50
>
```

Similarity/difference between various clustering algorithms

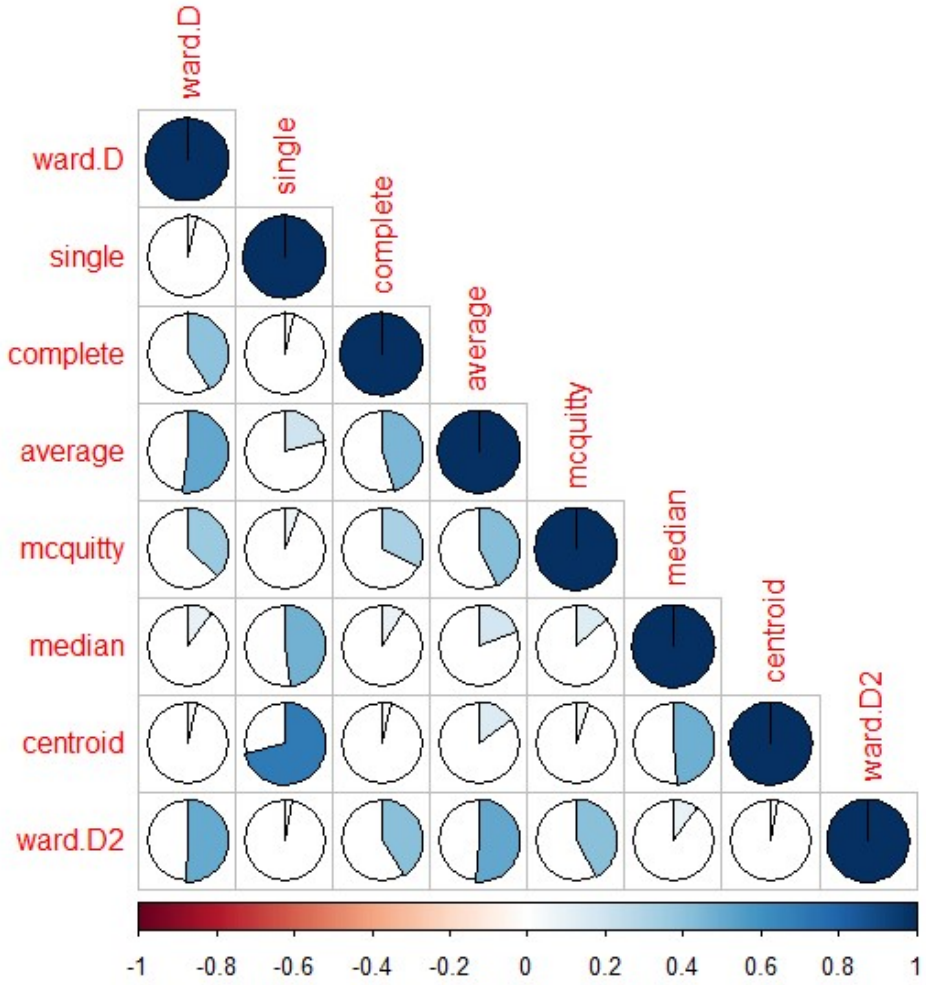
Let's check how many different results we could get if we would use different clustering algorithms (hclust has 8 different algorithms implemented). For the purpose of this analysis, we will create all 8 hclust objects, and chain them together into a single dendlist object (which, as the name implies, can hold a bunch of dendrograms together for the purpose of further analysis).

```
> hclust_methods <- c("ward.D", "single", "complete", "average", "mcquitty",
+                     "median", "centroid", "ward.D2")
> wine_dendlist <- dendlist()
> for(i in seq_along(hclust_methods)) {
+   hc_wine <- hclust(d, method = hclust_methods[i])
+   wine_dendlist <- dendlist(wine_dendlist, as.dendrogram(hc_wine))
+ }
> names(wine_dendlist) <- hclust_methods
> wine_dendlist
$ward.D
'dendrogram' with 2 branches and 4015 members total, at height 1767.831

$single
'dendrogram' with 2 branches and 4015 members total, at height 2.879334

$complete
'dendrogram' with 2 branches and 4015 members total, at height 11.10126

$average
'dendrogram' with 2 branches and 4015 members total, at height 5.80236

$mcquitty
'dendrogram' with 2 branches and 4015 members total, at height 6.703974

$median
'dendrogram' with 2 branches and 4015 members total, at height 6.317246

$centroid
'dendrogram' with 2 branches and 4015 members total, at height 4.434454

$ward.D2
'dendrogram' with 2 branches and 4015 members total, at height 129.346

attr(,"class")
[1] "dendlist"
>
```

Next, we can look at the cophenetic correlation between each clustering result using cor.dendlist:

```
> wine_dendlist_cor <- cor.dendlist(wine_dendlist)
> wine_dendlist_cor
             ward.D     single   complete   average   mcquitty     median   centroid    ward.D2
ward.D   1.00000000 0.03420559 0.40497414 0.5186364 0.36800628 0.09680452 0.03584561 0.50773788
single   0.03420559 1.00000000 0.03550259 0.2081046 0.06230492 0.47739328 0.70609598 0.02783508
complete 0.40497414 0.03550259 1.00000000 0.4507347 0.32419034 0.08983117 0.03410080 0.41083514
average  0.51863643 0.20810462 0.45073474 1.0000000 0.42893233 0.18742261 0.14940488 0.51325742
mcquitty 0.36800628 0.06230492 0.32419034 0.4289323 1.00000000 0.13161257 0.05126098 0.41552922
median   0.09680452 0.47739328 0.08983117 0.1874226 0.13161257 1.00000000 0.48856520 0.09568999
centroid 0.03584561 0.70609598 0.03410080 0.1494049 0.05126098 0.48856520 1.00000000 0.02808357
ward.D2  0.50773788 0.02783508 0.41083514 0.5132574 0.41552922 0.09568999 0.02808357 1.00000000
>
```

```
> library(corrplot)
> corrplot::corrplot(wine_dendlist_cor, "pie", "lower")
```



From the above figure, we can easily see that most clustering methods yield similar results, except for the complete method (the default method in hclust), which yields most of the high correlations.

```
> wine_agnes_s <- agnes(d, method = "single")
> wine_agnes_a <- agnes(d, method = "average")
> wine_agnes_c <- agnes(d, method = "complete")
> cbind(wine_agnes_s$ac, wine_agnes_a$ac, wine_agnes_c$ac)
          [,1]      [,2]      [,3]
[1,] 0.7376747 0.8587468 0.9252567
>
```

We can easily see that the complete method has the highest agnes in relation to the wine dataset. This implies that the complete method performed better among its pairs.

3rd Objective Neural Network (MLP)
Let's load the required library:

```
> library(neuralnet)
> library(grid)
> library(MASS)
> library(NeuralNetTools)
> library(tidyverse)
> 
> set.seed(3211)
> 
```

Next is to load the exchange dataset and check the first 6 rows:

```
> exchange <- readxl::read_xlsx("Exchange.xls.xlsx", sheet = 1)
> head(exchange)
# A tibble: 6 x 3
  `YYYY/MM/DD`   Wdy `USD/EUR`
         <dttm> <chr>     <dbl>
1   2015-03-19   Thu    1.0621
2   2015-03-20   Fri    1.0791
3   2015-03-23   Mon    1.0927
4   2015-03-24   Tue    1.0906
5   2015-03-25   Wed    1.0986
6   2015-03-26   Thu    1.0918
> 
```

Let's check the structure of the dataset:

```
> str(exchange)
Classes 'tbl_df', 'tbl' and 'data.frame':       390 obs. of  3 variables:
 $ YYYY/MM/DD: POSIXct, format: "2015-03-19" "2015-03-20" "2015-03-23" "2015-03-24" ...
 $ Wdy       : chr  "Thu" "Fri" "Mon" "Tue" ...
 $ USD/EUR   : num  1.06 1.08 1.09 1.09 1.1 ...
> 
```

The USD/EUR column is numeric, meanwhile the variable names are not in the format that we want it. Let's change the variable names to proper names and let's see the effect:

```
> exchange <- rename(exchange, date = `YYYY/MM/DD`, usd_eur = `USD/EUR`)
> head(exchange)
# A tibble: 6 x 3
        date   Wdy usd_eur
      <dttm> <chr>   <dbl>
1 2015-03-19   Thu  1.0621
2 2015-03-20   Fri  1.0791
3 2015-03-23   Mon  1.0927
4 2015-03-24   Tue  1.0906
5 2015-03-25   Wed  1.0986
6 2015-03-26   Thu  1.0918
> nrow(exchange)
[1] 390
> 
```

We can see that we have 390 rows in our dataset and the names as needed.

Next is to get the usd_eur variable, this will be used to formulate our first set of input and output variables for neural network model:

```
> usd_eur <- exchange$usd_eur
> head(usd_eur)
[1] 1.0621 1.0791 1.0927 1.0906 1.0986 1.0918
>
```

Let's check if we have any missing data:

```
> anyNA(usd_eur)
[1] FALSE
>
```

Data preparation and input selection

Let's write function that will convert the usd_eur into the required dataset matrix for NN First we'll experiment with two inputs and one output. Here we will check the head to see the first six rows of the newly created dataset for the NN:

```
> create_dataset_2in <- function(dataset){
+   look_back <- 1
+   dataA <- c(); dataB <- c(); data_out <- c()
+   for(i in look_back:(length(dataset)-look_back)){
+
+     dataA <- append(dataA, dataset[i])
+     dataB <- append(dataB, dataset[i + look_back])
+     data_out <- append(data_out, dataset[i + look_back + 1])
+   }
+   result <- as.data.frame(cbind(dataA, dataB, data_out))
+   result
+ }
> usd_eur_checker_2 <- create_dataset_2in(usd_eur)
> head(usd_eur_checker_2, 10)
    dataA  dataB data_out
1  1.0621 1.0791   1.0927
2  1.0791 1.0927   1.0906
3  1.0927 1.0906   1.0986
4  1.0906 1.0986   1.0918
5  1.0986 1.0918   1.0891
6  1.0918 1.0891   1.0817
7  1.0891 1.0817   1.0741
8  1.0817 1.0741   1.0767
9  1.0741 1.0767   1.0876
10 1.0767 1.0876   1.1006
>
```

Let's check the tail to see if we have any NA:

```
> tail(usd_eur_checker_2)
     dataA  dataB data_out
384 1.1200 1.1245   1.1238
385 1.1245 1.1238   1.1211
386 1.1238 1.1211   1.1213
387 1.1211 1.1213   1.1194
388 1.1213 1.1194   1.1157
389 1.1194 1.1157       NA
>
```

We have one NA which need to be dealt with.

Let's remove the NA:

```
> usd_eur_checker_2 <- drop_na(usd_eur_checker_2)
> summary(usd_eur_checker_2)
     dataA            dataB          data_out
 Min.   :1.056   Min.   :1.056   Min.   :1.056
 1st Qu.:1.095   1st Qu.:1.095   1st Qu.:1.095
 Median :1.114   Median :1.114   Median :1.114
 Mean   :1.110   Mean   :1.110   Mean   :1.110
 3rd Qu.:1.126   3rd Qu.:1.126   3rd Qu.:1.126
 Max.   :1.158   Max.   :1.158   Max.   :1.158
> |
```

As expected, the missing value is no more.

Next is normalize the dataset using the normalize function, this will enable the sigmoid activation in the NN train very well.

```
> normalize <- function(x) {
+   return((x - min(x)) / (max(x) - min(x)))
+ }
> usd_eur_norm_2 <- as.data.frame(lapply(usd_eur_checker_2, normalize))
> summary(usd_eur_norm_2)
     dataA            dataB          data_out
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.3825   1st Qu.:0.3850   1st Qu.:0.3854
 Median :0.5698   Median :0.5708   Median :0.5723
 Mean   :0.5285   Mean   :0.5300   Mean   :0.5309
 3rd Qu.:0.6849   3rd Qu.:0.6849   3rd Qu.:0.6849
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
> |
```

Let's see the number of row of our new dataset

```
> nrow(usd_eur_norm_2)
[1] 388
> |
```

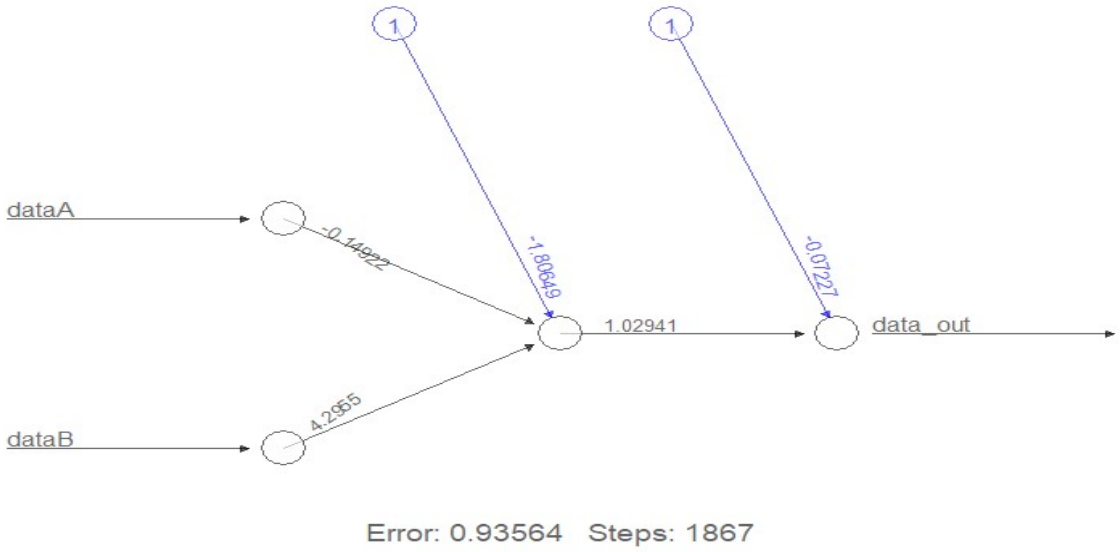
Now that we are left with 388 rows, we will divide the data into 320 training set and 68 test set.

```
> usd_eur_train_2 <- usd_eur_norm_2[1:320, ] #320 training set
> usd_eur_test_2 <- usd_eur_norm_2[321:388, ] #68 test set
> nrow(usd_eur_train_2)
[1] 320
> nrow(usd_eur_test_2)
[1] 68
> head(usd_eur_train_2)
       dataA     dataB  data_out
1 0.05899705 0.2261554 0.3598820
2 0.22615536 0.3598820 0.3392330
3 0.35988201 0.3392330 0.4178958
4 0.33923304 0.4178958 0.3510324
5 0.41789577 0.3510324 0.3244838
6 0.35103245 0.3244838 0.2517207
> tail(usd_eur_test_2)
        dataA     dataB  data_out
383 0.6361849 0.6283186 0.6725664
384 0.6283186 0.6725664 0.6656834
385 0.6725664 0.6656834 0.6391347
386 0.6656834 0.6391347 0.6411013
387 0.6391347 0.6411013 0.6224189
388 0.6411013 0.6224189 0.5860374
>
```

As expected we have 320 training set and 68 test set.

Model Fitting:
Model1 for 2inputs:
Let's train the first neural network, this time around we'll fit with 8 nodes NN.

```
> set.seed(3211)
>
> usd_eur_model2_1 <- neuralnet(data_out ~ dataA + dataB, data = usd_eur_train_2)
>
314 -0.1335721952 3.845014536
315 -0.1381683454 3.977319496
316 -0.1360176119 3.915408397
317 -0.1289309383 3.711411129
318 -0.1479935277 4.260147586
319 -0.1546027111 4.450399803
320 -0.1520413777 4.376669157


$result.matrix
                                    1
error                   0.935639590395
reached.threshold       0.009670456255
steps                1867.000000000000
Intercept.to.1layhid1  -1.806485157261
dataA.to.1layhid1      -0.149221810531
dataB.to.1layhid1       4.295504985145
Intercept.to.data_out  -0.072268461118
1layhid.1.to.data_out   1.029411769706

attr(,"class")
[1] "nn"
>
> plot(usd_eur_model2_1, alpha = 0.6)
>
```



There is one input node for each of the two features followed by a single hidden node and a single output node that predicts the data_out. The weights for each

connection are displayed. Some bias terms are indicated by nodes labelled with the number 1. These bias terms are numeric constants that allow the value at the indicated nodes to be shifted upward or downward, much like the intercept in a linear equation.
The bottom of the plot displays the number of training steps and Sum of Squared Errors (SSE) (sum of the squared predicted minus actual values). In our mode, we have an SSE of 0.93564 and 1867 training steps. A lower SSE implies better predictive performance. This is helpful for estimating the model's performance on the training data, but tells us little about how it will perform on unseen data.

Let's evaluate the correlation:
Since this is a numeric prediction problem rather than a classification problem, we cannot use a confusion matrix to examine model accuracy. Instead, we must measure the correlation between our predicted output and the true value. This provides insight into the output of the linear association between the two variables. Thus, we will use the cor() function to obtain a correlation between the two numeric vectors.

```
> net_results_2_1 <- neuralnet::compute(usd_eur_model2_1, usd_eur_test_2[, 1:2])
> pred_2_1 <- net_results_2_1$net.result
> cor(pred_2_1, usd_eur_test_2$data_out)
             [,1]
[1,] 0.89095166
>
```

The correlation here of about 0.89 indicates a strong relationship. This implies that our model is doing a good job, even with only a single hidden node. Given that we only used one hidden node, it is likely that we can improve the performance of our model. We will improve the performance later.

First let's plot to see how the prediction output vs. desired output:
Now we'll de-normalize the prediction output and desired output:
The denormalize function:

```
> minvec <- sapply(usd_eur_checker_2, min)
> maxvec <- sapply(usd_eur_checker_2, max)
> denormalize <- function(x, minval, maxval) {
+   x*(maxval - minval) + minval
+ }
>
```

Let's now apply it to both the output of the test data and the predicted result from neural network:
De-normalize the training set and subset the expected output:

```
> test_denorm_2 <- as.data.frame(Map(denormalize, usd_eur_test_2, minvec, maxvec))
> test_denorm_out_2 <- test_denorm_2$data_out
> head(test_denorm_out_2)
[1] 1.1034 1.1144 1.1079 1.1076 1.1064 1.1036
> length(test_denorm_out_2)
[1] 68
>
```

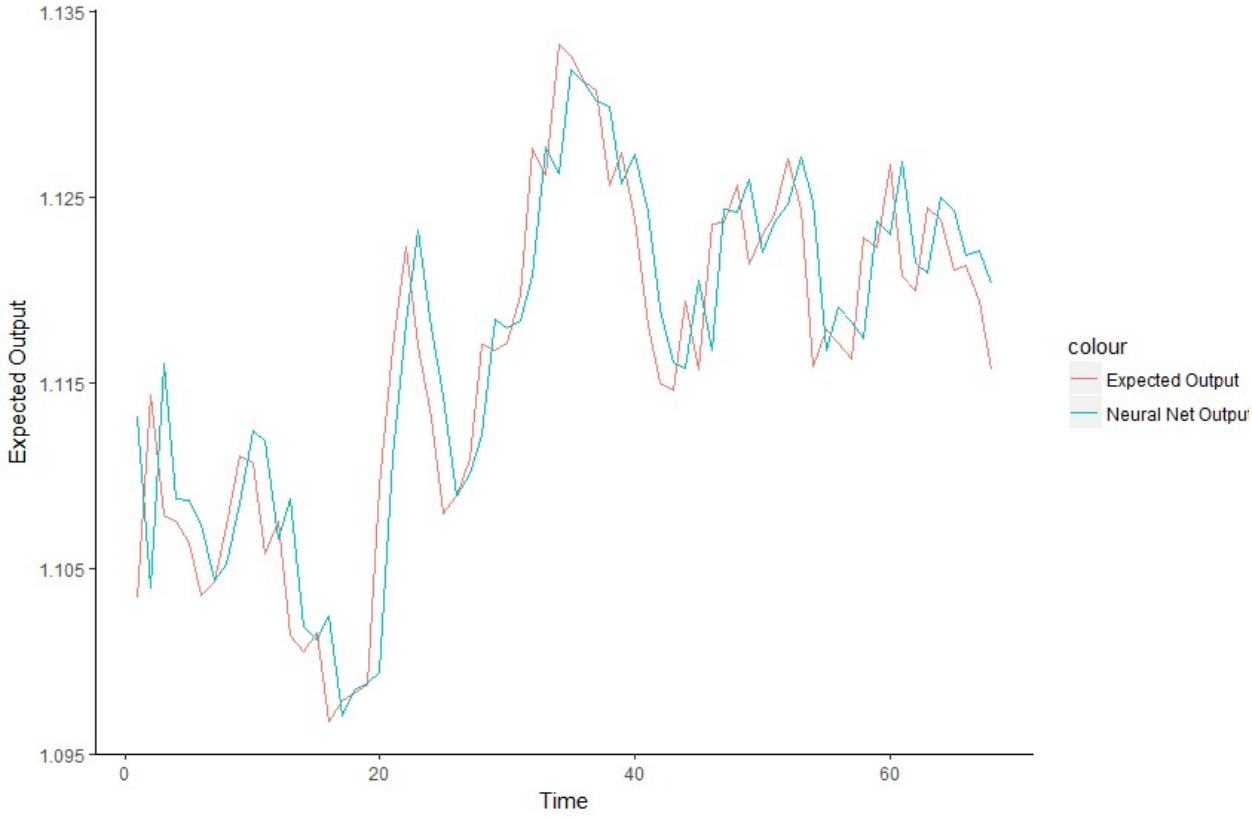
De-normalize the predicted output:

```
> ls(net_results_2_1)
[1] "net.result" "neurons"
> pred_all_2_1 <- as.data.frame(net_results_2_1$net.result)
> head(pred_all_2_1)
              V1
321 0.5611425230
322 0.4706524262
323 0.5891994387
324 0.5180248671
325 0.5172355812
326 0.5045344533
> pred_denorm_2_1 <- as.data.frame(Map(denormalize, pred_all_2_1, minvec, maxvec))
> pred_denorm_2_1 <- pred_denorm_2_1[, 1]
> head(pred_denorm_2_1)
[1] 1.113168195 1.103965352 1.116021583 1.108783129 1.108702859 1.107411154
> length(pred_denorm_2_1)
[1] 68
> series_2_1 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_denorm_2_1)
> head(series_2_1)
      test_denorm_out_2 pred_denorm_2_1
[1,] 1            1.1034     1.113168195
[2,] 2            1.1144     1.103965352
[3,] 3            1.1079     1.116021583
[4,] 4            1.1076     1.108783129
[5,] 5            1.1064     1.108702859
[6,] 6            1.1036     1.107411154
> colnames(series_2_1) <- c("Time","Expected Output","Neural Net Output")
> series_2_1 <- as.data.frame(series_2_1)
> head(series_2_1)
  Time Expected Output Neural Net Output
1    1          1.1034       1.113168195
2    2          1.1144       1.103965352
3    3          1.1079       1.116021583
4    4          1.1076       1.108783129
5    5          1.1064       1.108702859
6    6          1.1036       1.107411154
> |
```

We have finally de-normalized both the expected output and the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_2_1, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



We can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse <- function(actual, predicted){
+   sum((actual - predicted)^2)/length(actual)
+ }
> mse_2_1 <- mse(series_2_1$`Expected Output`, series_2_1$`Neural Net Output`)
> mse_2_1
[1] 0.00001844177611
>
```

Root mean squared Error

```
> rsme <- function(actual, predicted){
+   sqrt(sum((actual - predicted)^2)/length(actual))
+ }
> rmse_2_1 <- rsme(series_2_1$`Expected Output`, series_2_1$`Neural Net Output`)
> rmse_2_1
[1] 0.00429438891
>
```

Mean Percentage Absolute Error

```
> mape <- function(actual, predicted){
+   paste0(round(mean(100*abs((actual - predicted)/ actual)), 6) , "%")
+ }
> mape_2_1 <- mape(series_2_1$`Expected Output`, series_2_1$`Neural Net Output`)
> mape_2_1
[1] "0.298549%"
>
```

Improving Model Performance

Model Fitting:

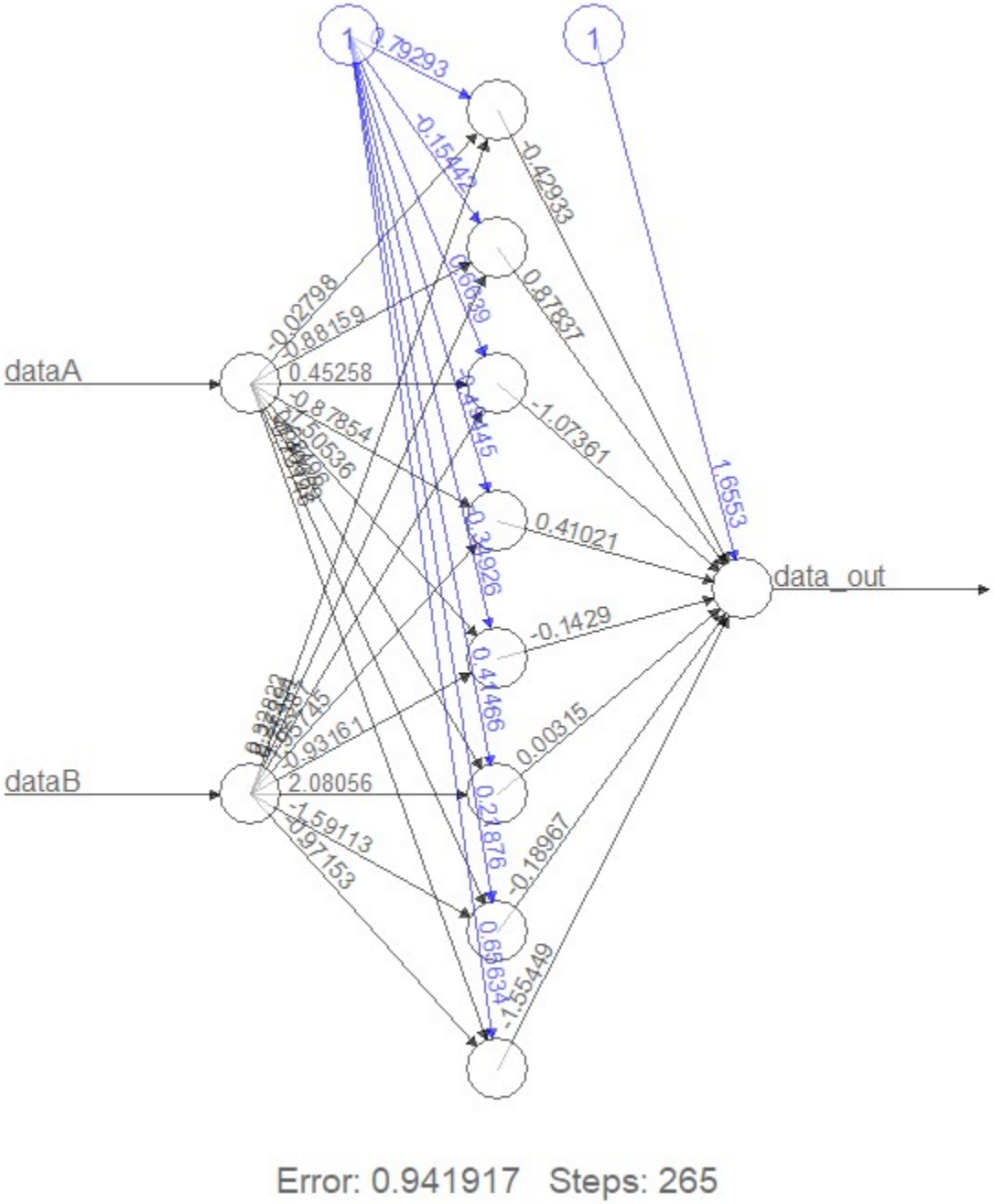
Model2 for 2inputs:

Let's train the first neural network, this time around we'll fit with 8 nodes NN.

```
> usd_eur_model2_2 <- neuralnet(data_out ~ dataA + dataB, data = usd_eur_train_2, hidden = 8)
>
```

```
$result.matrix
                                    1
error                   0.941916621058
reached.threshold       0.005429068729
steps                 265.000000000000
Intercept.to.1layhid1   0.792926093824
dataA.to.1layhid1      -0.027976995012
dataB.to.1layhid1       0.328220038279
Intercept.to.1layhid2  -0.154422502025
dataA.to.1layhid2      -0.881590100834
dataB.to.1layhid2       2.273944098534
Intercept.to.1layhid3   0.663898514759
dataA.to.1layhid3       0.452576699792
dataB.to.1layhid3       0.753867737668
Intercept.to.1layhid4  -0.494454207410
dataA.to.1layhid4      -0.878535981063
dataB.to.1layhid4       1.957451549760
Intercept.to.1layhid5  -0.349263905170
dataA.to.1layhid5      -1.505357967446
dataB.to.1layhid5      -0.931614564258
Intercept.to.1layhid6   0.414657137567
dataA.to.1layhid6       0.684958932411
dataB.to.1layhid6       2.080557391709
Intercept.to.1layhid7   0.218756948107
dataA.to.1layhid7      -1.406889902702
dataB.to.1layhid7      -1.591125184138
Intercept.to.1layhid8   0.656336182896
dataA.to.1layhid8      -0.731461476727
dataB.to.1layhid8      -0.971530090083
Intercept.to.data_out   1.655301899637
1layhid.1.to.data_out  -0.429327552676
1layhid.2.to.data_out   0.878366421488
1layhid.3.to.data_out  -1.073613281878
1layhid.4.to.data_out   0.410208991447
1layhid.5.to.data_out  -0.142902151915
1layhid.6.to.data_out   0.003153077560
1layhid.7.to.data_out  -0.189670602717
1layhid.8.to.data_out  -1.554485504126

attr(,"class")
[1] "nn"
>
```

```
> plot(usd_eur_model2_2, alpha = 0.6)
>
```



We can see from our newly plotted model, we have a rise in SSE of 0.941917 compared to from the previous model 0.93564. We also have a reduced training steps of 265 compared to 1867 from the previous model along with a few bias terms - which are indicated by nodes labelled with the number 1. This should come as no surprise given how complex the model has become. A more complex network takes more iterations to find the optimal weights.

Let's continue with our predictions and correlations.

```
> net_results_2_2 <- neuralnet::compute(usd_eur_model2_2, usd_eur_test_2[, 1:2])
> pred_2_2 <- net_results_2_2$net.result
> cor(pred_2_2, usd_eur_test_2$data_out)
              [,1]
[1,] 0.8923212835
>
```

The correlation here of about 0.89 indicates a strong relationship. This implies that our model is doing a good job with 8 hidden nodes. This is not far different from the previous model. Meanwhile will try to improve the performance adding threashold later.

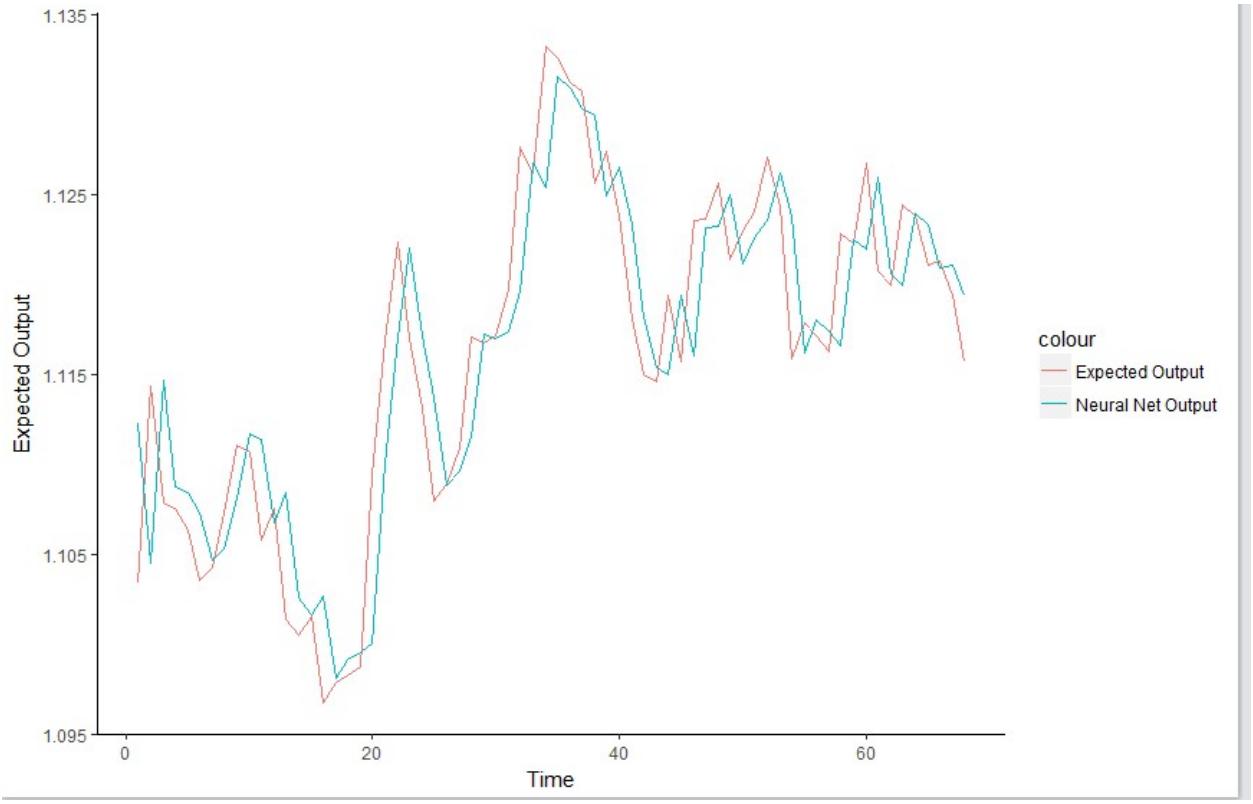
De-normalize the nn predicted output:

```
> pred_all_2_2 <- as.data.frame(net_results_2_2$net.result)
> head(pred_all_2_2)
              V1
321 0.5523058161
322 0.4758484780
323 0.5764306138
324 0.5178667755
325 0.5149016920
326 0.5037529477
> pred_denorm_2_2 <- as.data.frame(Map(denormalize, pred_all_2_2, minvec, maxvec))
> pred_denorm_2_2n <- pred_denorm_2_2[, 1]
> head(pred_denorm_2_2n)
[1] 1.112269501 1.104493790 1.114722993 1.108767051 1.108465502 1.107331675
> length(pred_denorm_2_2n)
[1] 68
> series_2_2 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_denorm_2_2n)
> head(series_2_2)
      test_denorm_out_2 pred_denorm_2_2n
[1,] 1            1.1034      1.112269501
[2,] 2            1.1144      1.104493790
[3,] 3            1.1079      1.114722993
[4,] 4            1.1076      1.108767051
[5,] 5            1.1064      1.108465502
[6,] 6            1.1036      1.107331675
> colnames(series_2_2) <- c("Time","Expected Output","Neural Net Output")
> series_2_2 <- as.data.frame(series_2_2)
> head(series_2_2)
  Time Expected Output Neural Net Output
1    1          1.1034       1.112269501
2    2          1.1144       1.104493790
3    3          1.1079       1.114722993
4    4          1.1076       1.108767051
5    5          1.1064       1.108465502
6    6          1.1036       1.107331675
>
```

We have finally de-normalized both the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_2_2, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



We can see that the plot looks very similar to that of model1, we can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like adding threshold increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse_2_2 <- mse(series_2_2$`Expected Output`, series_2_2$`Neural Net Output`)
> mse_2_2
[1] 0.00001735474227
```

Root mean squared Error

```
> rmse_2_2 <- rsme(series_2_2$`Expected Output`, series_2_2$`Neural Net Output`)
> rmse_2_2
[1] 0.004165902335
```

Mean Percentage Absolute Error

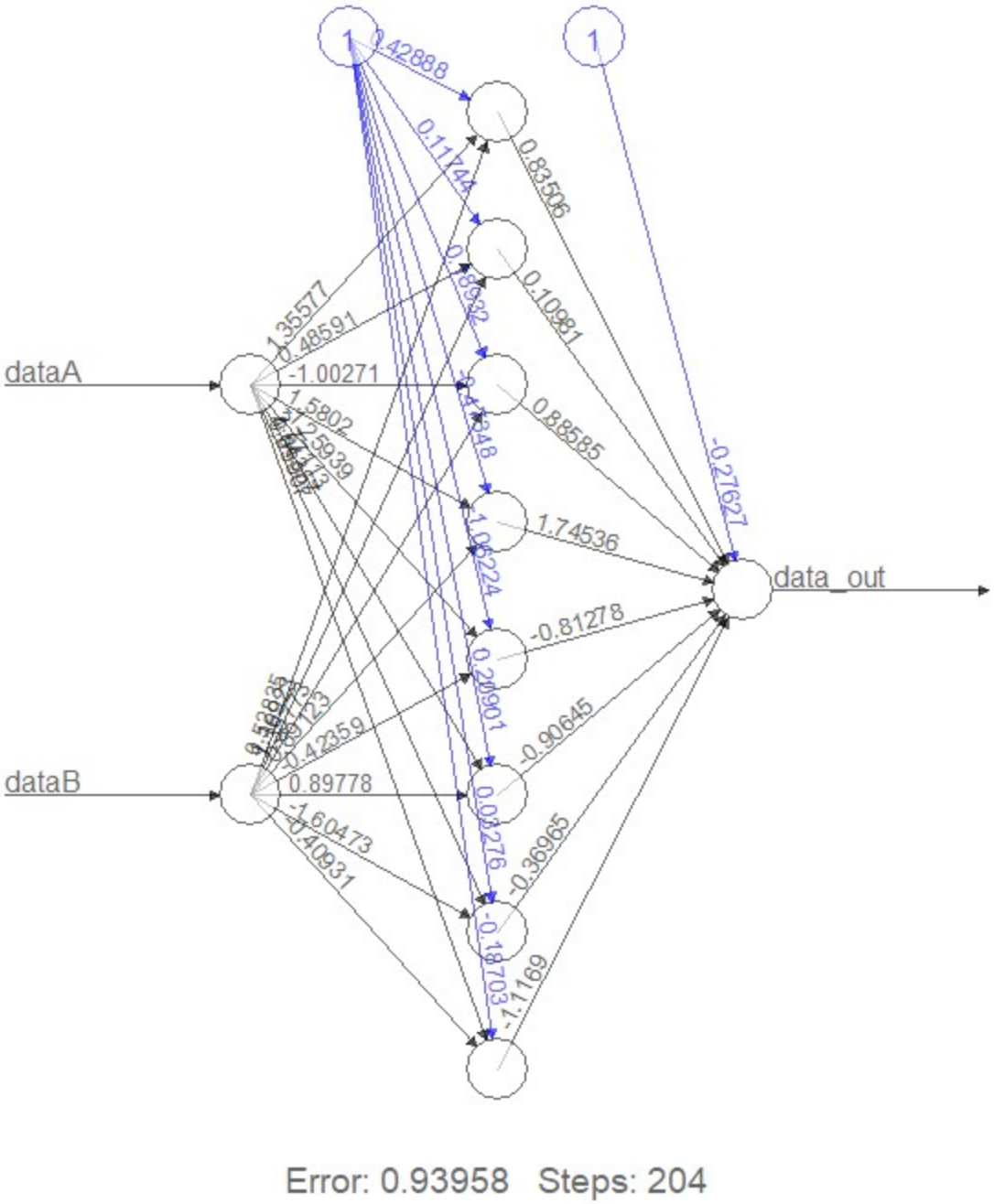
```
> mape_2_2 <- mape(series_2_2$`Expected Output`, series_2_2$`Neural Net Output`)
> mape_2_2
[1] "0.286674%"
>
```

Model3 for 2inputs:

Let’s train the first neural network, this time around we’ll fit with 8 nodes with threshold of 0.01.

```
> usd_eur_model2_3 <- neuralnet(data_out ~ dataA + dataB, data = usd_eur_train_2, hidden = 8, threshold = 0.01)
> 
```

```
$result.matrix
                                   1
error                  0.939579794119
reached.threshold      0.005260968488
steps                204.000000000000
Intercept.to.1layhid1  0.428875158056
dataA.to.1layhid1      1.355765953333
dataB.to.1layhid1      0.528348380440
Intercept.to.1layhid2  0.117442039188
dataA.to.1layhid2      0.485910298023
dataB.to.1layhid2      1.109232700825
Intercept.to.1layhid3  0.189321415310
dataA.to.1layhid3     -1.002711196775
dataB.to.1layhid3      1.357729755604
Intercept.to.1layhid4 -0.473476005654
dataA.to.1layhid4      1.580196520631
dataB.to.1layhid4      0.891230498673
Intercept.to.1layhid5 -1.062235504501
dataA.to.1layhid5     -1.259386131363
dataB.to.1layhid5     -0.423585995477
Intercept.to.1layhid6  0.209007104518
dataA.to.1layhid6      2.771128556829
dataB.to.1layhid6      0.897779707282
Intercept.to.1layhid7  0.032757024399
dataA.to.1layhid7      1.643571477013
dataB.to.1layhid7     -1.604728765252
Intercept.to.1layhid8 -0.187031827111
dataA.to.1layhid8      0.899070905355
dataB.to.1layhid8     -0.409314362020
Intercept.to.data_out -0.276271071708
1layhid.1.to.data_out  0.835059151570
1layhid.2.to.data_out  0.109811697666
1layhid.3.to.data_out  0.885853455624
1layhid.4.to.data_out  1.745360457134
1layhid.5.to.data_out -0.812775555659
1layhid.6.to.data_out -0.906448891894
1layhid.7.to.data_out -0.369650273119
1layhid.8.to.data_out -1.116902506936

attr(,"class")
[1] "nn"
> 
> plot(usd_eur_model2_3, alpha = 0.6)
> 
```



Error: 0.93958 Steps: 204

We can see from our newly plotted model, we have a reduced in SSE of 0.93958 compared to from the previous model2 0.941917 but higher than 0.93564 of model1. We also have a reduced training steps of 204 compared to 265 and 1867 from the previous model along with some bias terms - which are indicated by nodes labelled with the number 1. This should come as no surprise given how complex the model has become. A more complex network takes more iterations to find the optimal weights. A lesser SSE denote a reduction in error. Hence a better performance.

Let's continue with our predictions and correlations.

```
> plot(usd_eur_model2_3, alpha = 0.6)
> net_results_2_3 <- neuralnet::compute(usd_eur_model2_3, usd_eur_test_2[, 1:2])
> pred_2_3 <- net_results_2_3$net.result
> cor(pred_2_3, usd_eur_test_2$data_out)
             [,1]
[1,] 0.8925157516
>
```

The correlation here of about 0.89 indicates a strong relationship. This implies that our model is doing a good job with 8 hidden nodes and threshold of 0.01. This is not far different from the previous model. Meanwhile will try to improve the performance adding adding more hidden layers later.

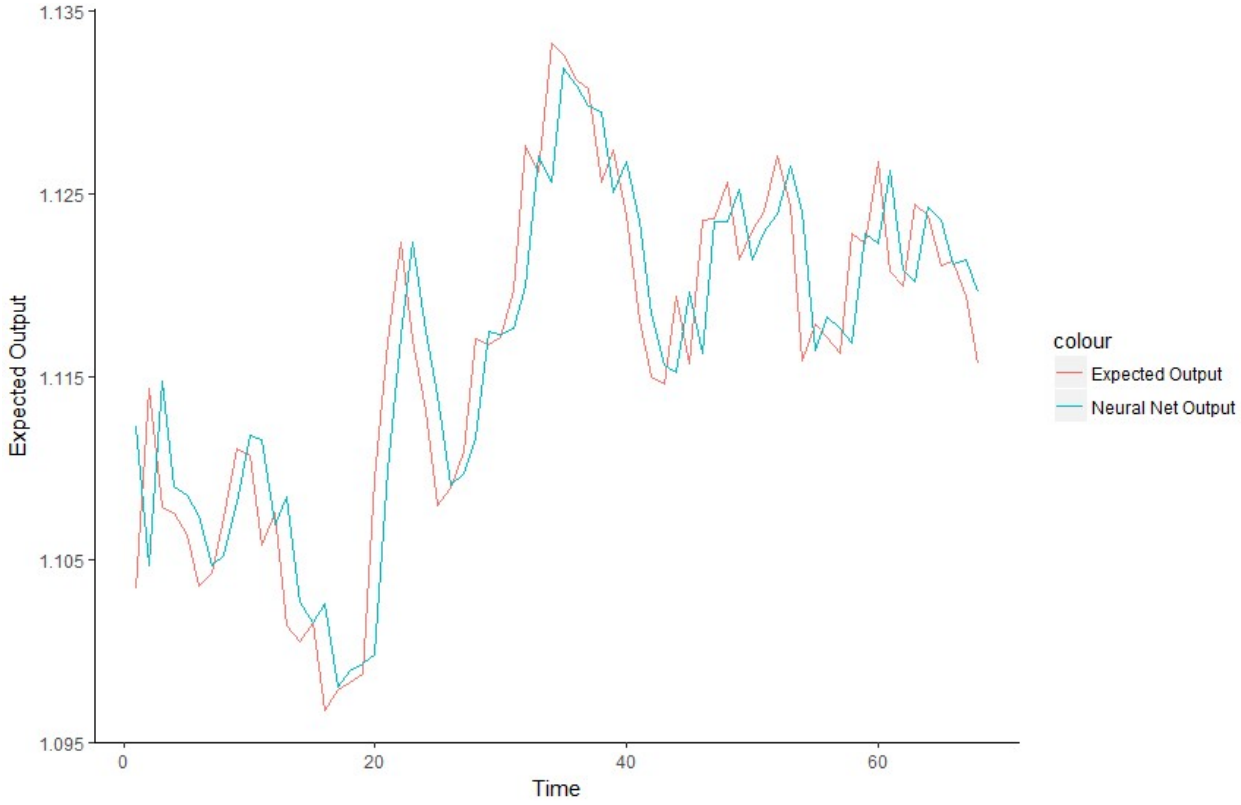
De-normalize the nn predicted output:

```
> pred_all_2_3 <- as.data.frame(net_results_2_3$net.result)
> head(pred_all_2_3)
               V1
321 0.5526499993
322 0.4780087608
323 0.5767939097
324 0.5203587387
325 0.5156703357
326 0.5044506185
> pred_denorm_2_3 <- as.data.frame(Map(denormalize, pred_all_2_3, minvec, maxvec))
> pred_denorm_2_3n <- pred_denorm_2_3[, 1]
> head(pred_denorm_2_3n)
[1] 1.112304505 1.104713491 1.114759941 1.109020484 1.108543673 1.107402628
> length(pred_denorm_2_3n)
[1] 68
> series_2_3 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_denorm_2_3n)
> head(series_2_3)
       test_denorm_out_2 pred_denorm_2_3n
[1,] 1            1.1034      1.112304505
[2,] 2            1.1144      1.104713491
[3,] 3            1.1079      1.114759941
[4,] 4            1.1076      1.109020484
[5,] 5            1.1064      1.108543673
[6,] 6            1.1036      1.107402628
> colnames(series_2_3) <- c("Time","Expected Output","Neural Net Output")
> series_2_3 <- as.data.frame(series_2_3)
> head(series_2_3)
  Time Expected Output Neural Net Output
1    1          1.1034       1.112304505
2    2          1.1144       1.104713491
3    3          1.1079       1.114759941
4    4          1.1076       1.109020484
5    5          1.1064       1.108543673
6    6          1.1036       1.107402628
>
```

We have finally de-normalized both the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_2_3, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



We can see that the plot looks very similar to that of model1 and model2, we can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like adding threshold increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse_2_3 <- mse(series_2_3$`Expected Output`, series_2_3$`Neural Net Output`)
> mse_2_3
[1] 0.00001743605474
```

Root mean squared Error

```
> rmse_2_3 <- rsme(series_2_3$`Expected Output`, series_2_3$`Neural Net Output`)
> rmse_2_3
[1] 0.004175650218
```

Mean Percentage Absolute Error

```
> mape_2_3 <- mape(series_2_3$`Expected Output`, series_2_3$`Neural Net Output`)
> mape_2_3
[1] "0.288677%"
>
```

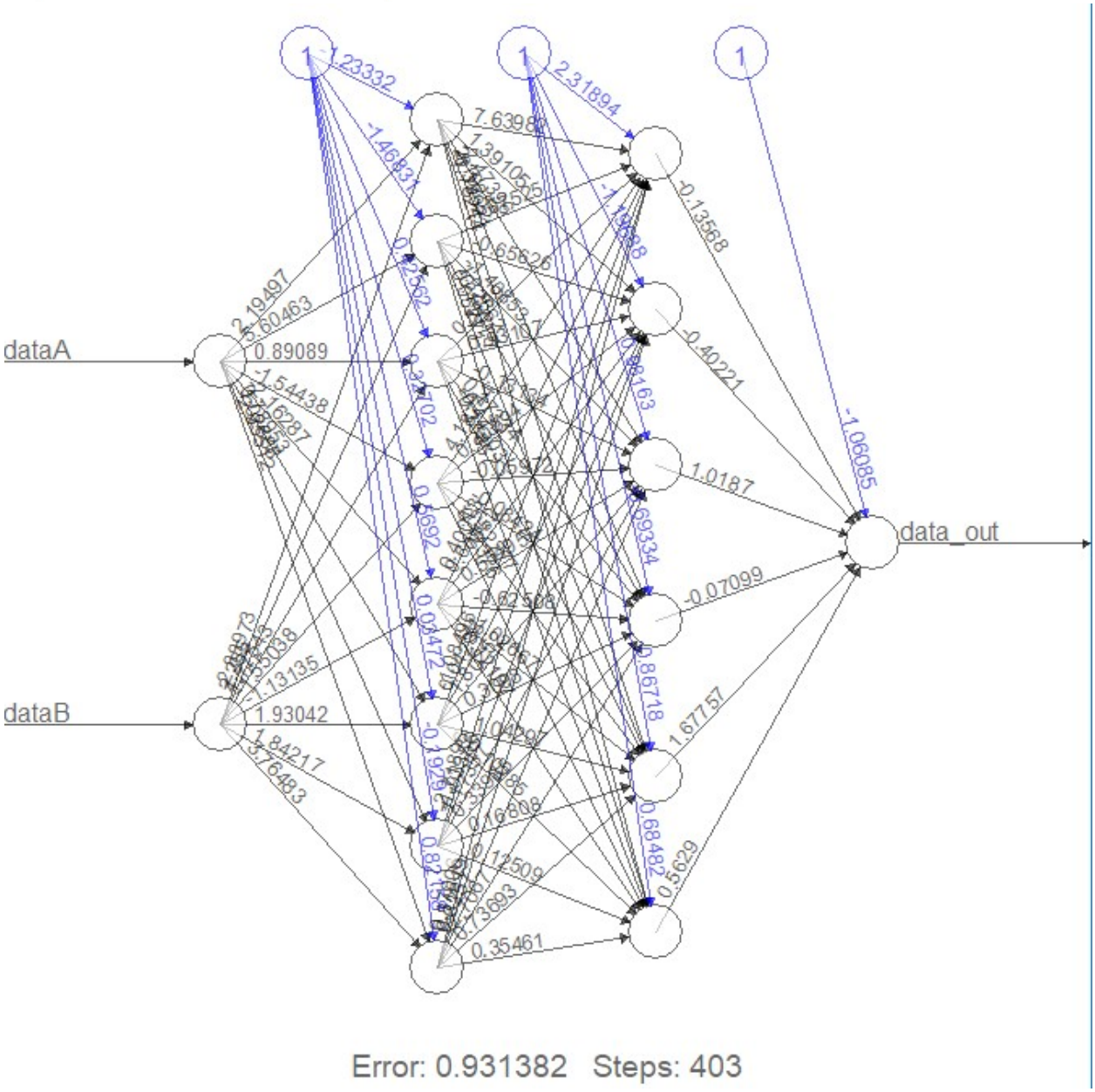
Model4 for 2inputs:

Let's train the first neural network, this time around we'll fit with 8 nodes NN.

```
> usd_eur_model2_4 <- neuralnet(data_out ~ dataA + dataB, data = usd_eur_train_2, hidden = c(8,6))
>
```

```
$result.matrix
                                    1
error                   0.931382398236
reached.threshold       0.005905869702
steps                 403.000000000000
Intercept.to.1layhid1  -1.233320741201
dataA.to.1layhid1       2.194966561793
dataB.to.1layhid1      -2.889726484556
Intercept.to.1layhid2  -1.468305643776
dataA.to.1layhid2       5.604631963359
dataB.to.1layhid2       2.254999034620
Intercept.to.1layhid3   0.425615725158
dataA.to.1layhid3       0.890892854492
dataB.to.1layhid3       1.775431208341
Intercept.to.1layhid4   0.327020268454
dataA.to.1layhid4      -1.544376464878
dataB.to.1layhid4      -1.550376598488
Intercept.to.1layhid5   0.569198217216
dataA.to.1layhid5      -2.162871869894
dataB.to.1layhid5      -1.131349133116
Intercept.to.1layhid6   0.034720939808
dataA.to.1layhid6       0.789534456313
dataB.to.1layhid6       1.930422221055
Intercept.to.1layhid7  -0.192898523865
dataA.to.1layhid7       1.028876000329
dataB.to.1layhid7       1.842174177654
Intercept.to.1layhid8   0.821575855162
dataA.to.1layhid8      -2.955122374781
dataB.to.1layhid8       3.764825275233
Intercept.to.2layhid1   2.318935189102
1layhid.1.to.2layhid1   7.639824794339
1layhid.2.to.2layhid1  -1.075252362940
1layhid.3.to.2layhid1   0.579898775797
1layhid.4.to.2layhid1   4.187441553888
1layhid.5.to.2layhid1   0.400126992784
1layhid.6.to.2layhid1  -1.084963416643
1layhid.7.to.2layhid1  -2.718959298760
1layhid.8.to.2layhid1   0.170056769377
Intercept.to.2layhid2  -1.196881289210
1layhid.1.to.2layhid2   1.391049168625
1layhid.2.to.2layhid2  -0.656264318951
1layhid.3.to.2layhid2   0.931070860160
1layhid.4.to.2layhid2   0.405937262863
1layhid.5.to.2layhid2   0.608515906094
1layhid.6.to.2layhid2   0.042381192517
1layhid.7.to.2layhid2  -1.933562498491
1layhid.8.to.2layhid2   2.815185599018
Intercept.to.2layhid3   0.981628996810
1layhid.1.to.2layhid3  -2.473957930627
1layhid.2.to.2layhid3  -1.408530172498
1layhid.3.to.2layhid3  -0.131341961146
```

```
> plot(usd_eur_model2_4, alpha = 0.6)
```



We can see from our newly plotted model, we have a reduced in SSE of 0.931382 compared to from the previous model 0.941917, 0.93564 and 0.93958. We also have a reduced training steps of 403 compared to 204, 265 and 1867 from the previous model along with some bias terms - which are indicated by nodes labelled with the number 1. This should come as no surprise given how complex the model has become. A more complex network takes more iterations to find the optimal weights. A lesser SSE denote a reduction in error. Hence a better performance.

Let's continue with our predictions and correlations.

```
> net_results_2_4 <- neuralnet::compute(usd_eur_model2_4, usd_eur_test_2[, 1:2])
> pred_2_4 <- net_results_2_4$net.result
> cor(pred_2_4, usd_eur_test_2$data_out)
             [,1]
[1,] 0.891929248
>
```

The correlation here of about 0.89 indicates a strong relationship. This implies that our model is doing a good job with 8 hidden nodes and threshold of 0.01. This is not far different from the previous model. Meanwhile will try to improve the performance adding adding more hidden layers later.

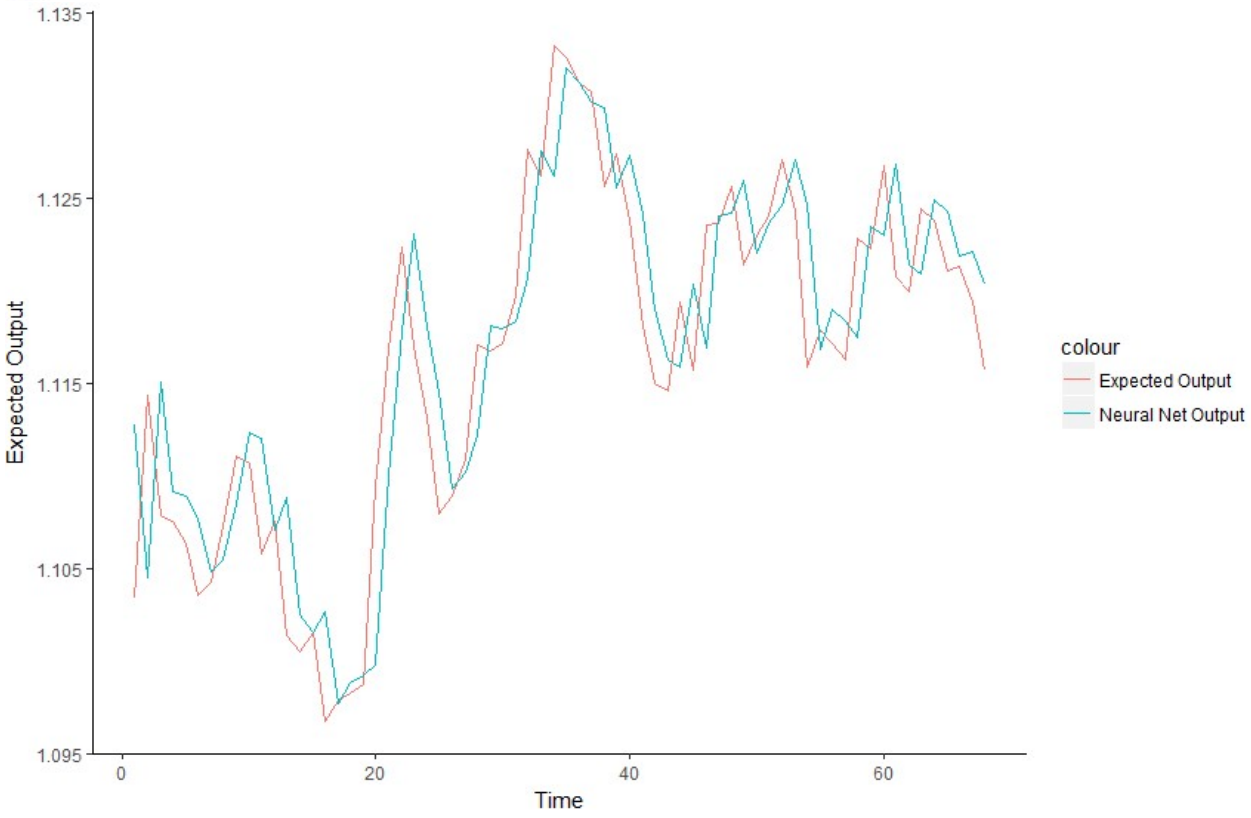
De-normalize the nn predicted output:

```
> pred_all_2_4 <- as.data.frame(net_results_2_4$net.result)
> head(pred_all_2_4)
               V1
321 0.5574787544
322 0.4760027646
323 0.5801469242
324 0.5220310234
325 0.5195432553
326 0.5075898662
> pred_denorm_2_4 <- as.data.frame(Map(denormalize, pred_all_2_4, minvec, maxvec))
> pred_denorm_2_4n <- pred_denorm_2_4[, 1]
> head(pred_denorm_2_4n)
[1] 1.112795589 1.104509481 1.115100942 1.109190555 1.108937549 1.107721889
> length(pred_denorm_2_4n)
[1] 68
> series_2_4 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_denorm_2_4n)
> head(series_2_4)
       test_denorm_out_2 pred_denorm_2_4n
[1,] 1            1.1034      1.112795589
[2,] 2            1.1144      1.104509481
[3,] 3            1.1079      1.115100942
[4,] 4            1.1076      1.109190555
[5,] 5            1.1064      1.108937549
[6,] 6            1.1036      1.107721889
> colnames(series_2_4) <- c("Time","Expected Output","Neural Net Output")
> series_2_4 <- as.data.frame(series_2_4)
> head(series_2_4)
  Time Expected Output Neural Net Output
1    1          1.1034       1.112795589
2    2          1.1144       1.104509481
3    3          1.1079       1.115100942
4    4          1.1076       1.109190555
5    5          1.1064       1.108937549
6    6          1.1036       1.107721889
>
```

We have finally de-normalized both the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_2_4, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



We can see that the plot looks very similar to that of model 1, 2 and 3, we can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like adding threshold increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse_2_4 <- mse(series_2_4$`Expected Output`, series_2_4$`Neural Net Output`)
> mse_2_4
[1] 0.00001814210198
```

Root mean squared Error

```
> rmse_2_4 <- rsme(series_2_4$`Expected Output`, series_2_4$`Neural Net Output`)
> rmse_2_4
[1] 0.004259354643
```

Mean Percentage Absolute Error

```
> mape_2_4 <- mape(series_2_4$`Expected Output`, series_2_4$`Neural Net Output`)
> mape_2_4
[1] "0.29769%"
>
```

Data preparation and input selection

Let's write function that will convert the usd_eur into the required dataset matrix for NN First we'll experiment with three inputs and one output. Here we will check the head to see the first six rows of the newly created dataset for the NN:

```
> create_dataset_3in <- function(dataset){
+   look_back <- 1
+   dataA <- c(); dataB <- c(); dataC <- c(); data_out <- c()
+   for(i in look_back:(length(dataset)-look_back)){
+
+     dataA <- append(dataA, dataset[i])
+     dataB <- append(dataB, dataset[i + look_back])
+     dataC <- append(dataC, dataset[i + look_back + 1])
+     data_out <- append(data_out, dataset[i + look_back + 2])
+   }
+   result <- as.data.frame(cbind(dataA, dataB, dataC, data_out))
+   result
+ }
> usd_eur_checker_3 <- create_dataset_3in(usd_eur)
> head(usd_eur_checker_3, 10)
    dataA  dataB  dataC data_out
1  1.0621 1.0791 1.0927   1.0906
2  1.0791 1.0927 1.0906   1.0986
3  1.0927 1.0906 1.0986   1.0918
4  1.0906 1.0986 1.0918   1.0891
5  1.0986 1.0918 1.0891   1.0817
6  1.0918 1.0891 1.0817   1.0741
7  1.0891 1.0817 1.0741   1.0767
8  1.0817 1.0741 1.0767   1.0876
9  1.0741 1.0767 1.0876   1.1006
10 1.0767 1.0876 1.1006   1.0850
>
```

Let's check the tail to see if we have any NA:

```
> anyNA(usd_eur_checker_2)
[1] TRUE
>
```

```
> tail(usd_eur_checker_3)
     dataA  dataB  dataC data_out
384 1.1200 1.1245 1.1238   1.1211
385 1.1245 1.1238 1.1211   1.1213
386 1.1238 1.1211 1.1213   1.1194
387 1.1211 1.1213 1.1194   1.1157
388 1.1213 1.1194 1.1157       NA
389 1.1194 1.1157     NA       NA
> str(usd_eur_checker_3)
'data.frame':   389 obs. of  4 variables:
 $ dataA   : num  1.06 1.08 1.09 1.09 1.1 ...
 $ dataB   : num  1.08 1.09 1.09 1.1 1.09 ...
 $ dataC   : num  1.09 1.09 1.1 1.09 1.09 ...
 $ data_out: num  1.09 1.1 1.09 1.09 1.08 ...
>
```

We have one NA which need to be dealt with.

Let's remove the NA:

```
> usd_eur_checker_3 <- drop_na(usd_eur_checker_3)
> summary(usd_eur_checker_3)
     dataA             dataB             dataC             data_out
 Min.   :1.056100   Min.   :1.056100   Min.   :1.056100   Min.   :1.05610
 1st Qu.:1.094900   1st Qu.:1.095200   1st Qu.:1.095300   1st Qu.:1.09530
 Median :1.114000   Median :1.114100   Median :1.114200   Median :1.11440
 Mean   :1.109824   Mean   :1.109976   Mean   :1.110081   Mean   :1.11014
 3rd Qu.:1.125800   3rd Qu.:1.125800   3rd Qu.:1.125800   3rd Qu.:1.12580
 Max.   :1.157800   Max.   :1.157800   Max.   :1.157800   Max.   :1.15780
>
```

As expected, the missing value is no more.

Next is normalize the dataset using the normalize function, this will enable the sigmoid activation in the NN train very well.

```
> normalize <- function(x) {
+   return((x - min(x)) / (max(x) - min(x)))
+ }
> usd_eur_norm_3 <- as.data.frame(lapply(usd_eur_checker_3, normalize))
> summary(usd_eur_norm_3)
     dataA             dataB             dataC             data_out
 Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000
 1st Qu.:0.3815143   1st Qu.:0.3844641   1st Qu.:0.3854474   1st Qu.:0.3854474
 Median :0.5693215   Median :0.5703048   Median :0.5712881   Median :0.5732547
 Mean   :0.5282548   Mean   :0.5297590   Mean   :0.5307829   Mean   :0.5313673
 3rd Qu.:0.6853491   3rd Qu.:0.6853491   3rd Qu.:0.6853491   3rd Qu.:0.6853491
 Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000
>
```

Let's see the number of row of our new dataset

```
> nrow(usd_eur_norm_3)
[1] 387
>
```

Now that we are left with 387 rows, we will divide the data into 320 training set and 67 test set.

```
> usd_eur_train_3 <- usd_eur_norm_3[1:320, ]
> usd_eur_test_3 <- usd_eur_norm_3[321:387, ]
> nrow(usd_eur_train_3)
[1] 320
> nrow(usd_eur_test_3)
[1] 67
> head(usd_eur_train_3)
          dataA        dataB        dataC     data_out
1 0.05899705015 0.2261553589 0.3598820059 0.3392330383
2 0.22615535890 0.3598820059 0.3392330383 0.4178957719
3 0.35988200590 0.3392330383 0.4178957719 0.3510324484
4 0.33923303835 0.4178957719 0.3510324484 0.3244837758
5 0.41789577188 0.3510324484 0.3244837758 0.2517207473
6 0.35103244838 0.3244837758 0.2517207473 0.1769911504
> tail(usd_eur_test_3)
           dataA        dataB        dataC     data_out
382 0.6951819076 0.6361848574 0.6283185841 0.6725663717
383 0.6361848574 0.6283185841 0.6725663717 0.6656833825
384 0.6283185841 0.6725663717 0.6656833825 0.6391347099
385 0.6725663717 0.6656833825 0.6391347099 0.6411012783
386 0.6656833825 0.6391347099 0.6411012783 0.6224188791
387 0.6391347099 0.6411012783 0.6224188791 0.5860373648
> |
```

As expected we have 320 training set and 67 test set.

Model Fitting:

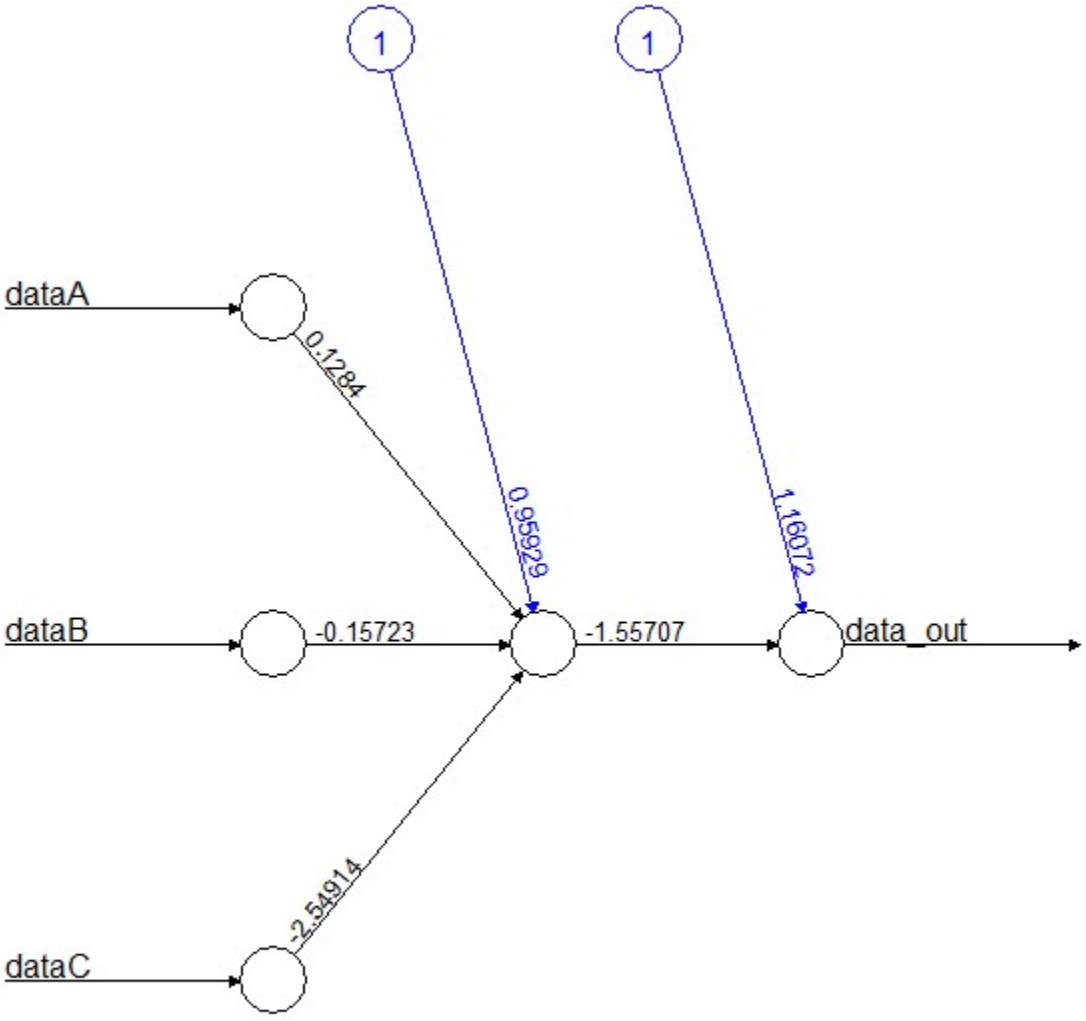
Model1 for 2inputs:

Let's train the first neural network, this time around we'll fit with 8 nodes NN.

```
> usd_eur_model3_1 <- neuralnet(data_out ~ dataA + dataB + dataC, data = usd_eur_train_3)
> |
```

```
308 -0.1980502651 0.2425185019  3.931917765
309 -0.1928943182 0.2362048900  3.829556078
310 -0.1943023265 0.2379290385  3.857509451
311 -0.1919114616 0.2350013525  3.810043297
312 -0.1954865056 0.2393791013  3.881019112
313 -0.2033423091 0.2489987689  4.036981406
314 -0.1967153833 0.2408838990  3.905416182
315 -0.1986104007 0.2432044048  3.943038210
316 -0.2122967053 0.2599636961  4.214754202
317 -0.1921184439 0.2352548085  3.814152543
318 -0.2001165150 0.2450486871  3.972939294
319 -0.1963233782 0.2404038770  3.897633653
320 -0.1923968885 0.2355957724  3.819680542


$result.matrix
                                    1
error                   0.931982123872
reached.threshold       0.009410381809
steps                 773.000000000000
Intercept.to.1layhid1   0.959289744664
dataA.to.1layhid1       0.128399667034
dataB.to.1layhid1      -0.157229251275
dataC.to.1layhid1      -2.549135349816
Intercept.to.data_out   1.160722563297
1layhid.1.to.data_out  -1.557073439972

attr(,"class")
[1] "nn"
> |
```

```
> plot(usd_eur_model3_1)
>
```



There is one input node for each of the three features followed by a single hidden node and a single output node that predicts the data_out. The weights for each connection are displayed. Some bias terms are indicated by nodes labelled with the number 1. These bias terms are numeric constants that allow the value at the indicated nodes to be shifted upward or downward, much like the intercept in a linear equation.

The bottom of the plot displays the number of training steps and Sum of Squared Errors (SSE) (sum of the squared predicted minus actual values). In our mode, we have an SSE of 0.931982 and 773 training steps. A lower SSE implies better

predictive performance. This is helpful for estimating the model's performance on the training data, but tells us little about how it will perform on unseen data.

Let's evaluate the correlation:
Since this is a numeric prediction problem rather than a classification problem, we cannot use a confusion matrix to examine model accuracy. Instead, we must measure the correlation between our predicted output and the true value. This provides insight into the output of the linear association between the two variables. Thus, we will use the cor() function to obtain a correlation between the two numeric vectors.

```
> net_results_3_1 <- neuralnet::compute(usd_eur_model3_1, usd_eur_test_3[, 1:3])
> pred_3_1 <- net_results_3_1$net.result
> cor(pred_3_1, usd_eur_test_3$data_out)
             [,1]
[1,] 0.8993621886
> |
```

The correlation here of about 0.90 indicates a strong relationship. This implies that our model is doing a good job, even with only a single hidden node. Given that we only used one hidden node, it is likely that we can improve the performance of our model. We will improve the performance later.

First let's plot to see how the prediction output vs. desired output:
Now we'll de-normalize the prediction output and desired output:
The denormalize function:

```
> net_results_3_1 <- neuralnet::compute(usd_eur_model3_1, usd_eur_test_3[, 1:3])
> pred_3_1 <- net_results_3_1$net.result
> cor(pred_3_1, usd_eur_test_3$data_out)
             [,1]
[1,] 0.8993621886
> minvec <- sapply(usd_eur_checker_3, min)
> maxvec <- sapply(usd_eur_checker_3, max)
> denormalize <- function(x, minval, maxval) {
+   x*(maxval - minval) + minval
+ }
> |
```

Let's now apply it to both the output of the test data and the predicted result from neural network:
De-normalize the training set and subset the expected output:

```
> test_denorm_3 <- as.data.frame(Map(denormalize, usd_eur_test_3, minvec, maxvec))
> head(test_denorm_3)
   dataA  dataB  dataC data_out
1 1.1053 1.1117 1.1034   1.1144
2 1.1117 1.1034 1.1144   1.1079
3 1.1034 1.1144 1.1079   1.1076
4 1.1144 1.1079 1.1076   1.1064
5 1.1079 1.1076 1.1064   1.1036
6 1.1076 1.1064 1.1036   1.1043
> test_denorm_out_3 <- test_denorm_3$data_out
> head(test_denorm_out_3)
[1] 1.1144 1.1079 1.1076 1.1064 1.1036 1.1043
> length(test_denorm_out_3)
[1] 67
>
```

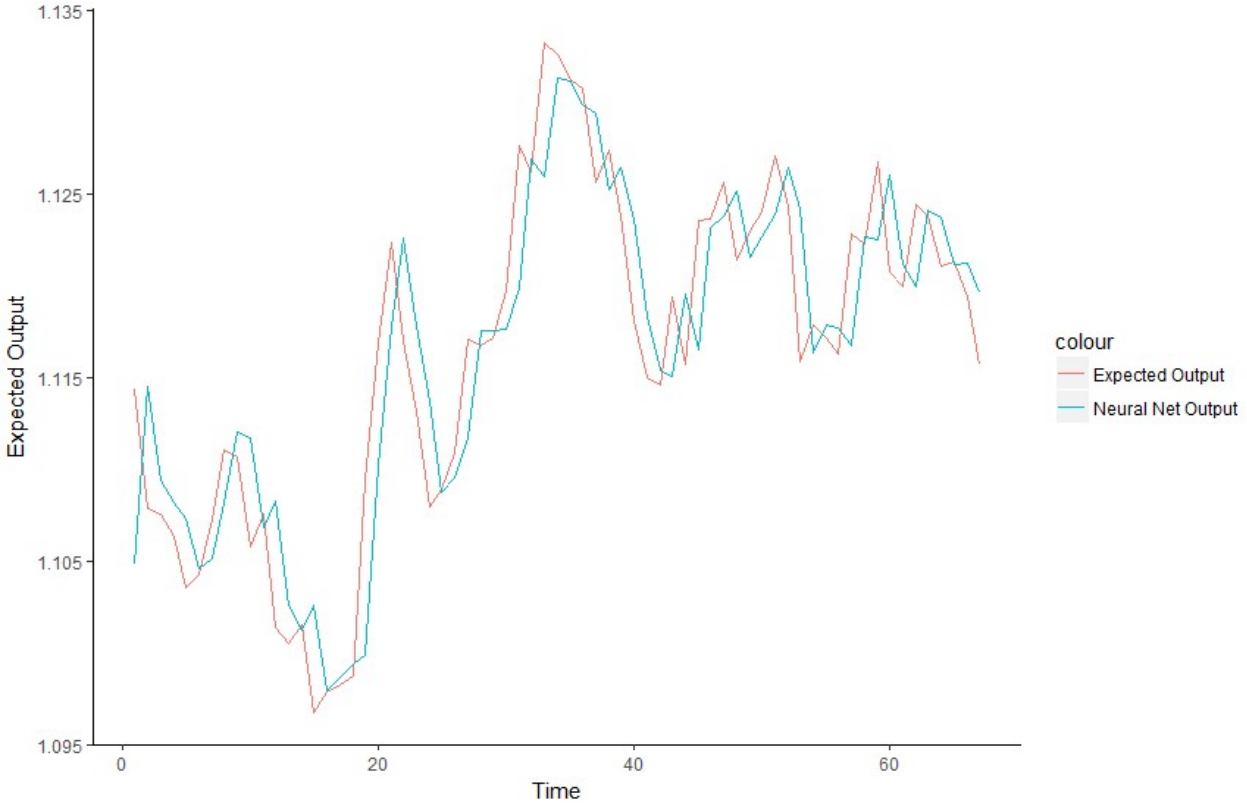
De-normalize the predicted output:

```
> pred_all_3_1 <- as.data.frame(net_results_3_1$net.result)
> head(pred_all_3_1)
               V1
321 0.4790520459
322 0.5746718044
323 0.5244077431
324 0.5125382280
325 0.5040719361
326 0.4767192814
> pred_denorm_3_1 <- as.data.frame(Map(denormalize, pred_all_3_1, minvec, maxvec))
> pred_denorm_3_1n <- pred_denorm_3_1[, 1]
> head(pred_denorm_3_1n)
[1] 1.104819593 1.114544123 1.109432267 1.108225138 1.107364116 1.104582351
> length(pred_denorm_3_1n)
[1] 67
> series_3_1 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_denorm_3_1n)
> head(series_3_1)
      test_denorm_out_3 pred_denorm_3_1n
[1,] 1            1.1144      1.104819593
[2,] 2            1.1079      1.114544123
[3,] 3            1.1076      1.109432267
[4,] 4            1.1064      1.108225138
[5,] 5            1.1036      1.107364116
[6,] 6            1.1043      1.104582351
> colnames(series_3_1) <- c("Time","Expected Output","Neural Net Output")
> series_3_1 <- as.data.frame(series_3_1)
> head(series_3_1)
  Time Expected Output Neural Net Output
1    1          1.1144       1.104819593
2    2          1.1079       1.114544123
3    3          1.1076       1.109432267
4    4          1.1064       1.108225138
5    5          1.1036       1.107364116
6    6          1.1043       1.104582351
>
```

We have finally de-normalized both the expected output and the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_3_1, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



We can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse_3_1 <- mse(series_3_1$`Expected Output`, series_3_1$`Neural Net Output`)
> mse_3_1
[1] 0.0000161682225
```

Root mean squared Error

```
> rmse_3_1 <- rsme(series_3_1$`Expected Output`, series_3_1$`Neural Net Output`)
> rmse_3_1
[1] 0.00402097283
```

Mean Percentage Absolute Error

```
> mape_3_1 <- mape(series_3_1$`Expected Output`, series_3_1$`Neural Net Output`)
> mape_3_1
[1] "0.27865%"
>
```

Improving Model Performance

Model Fitting:

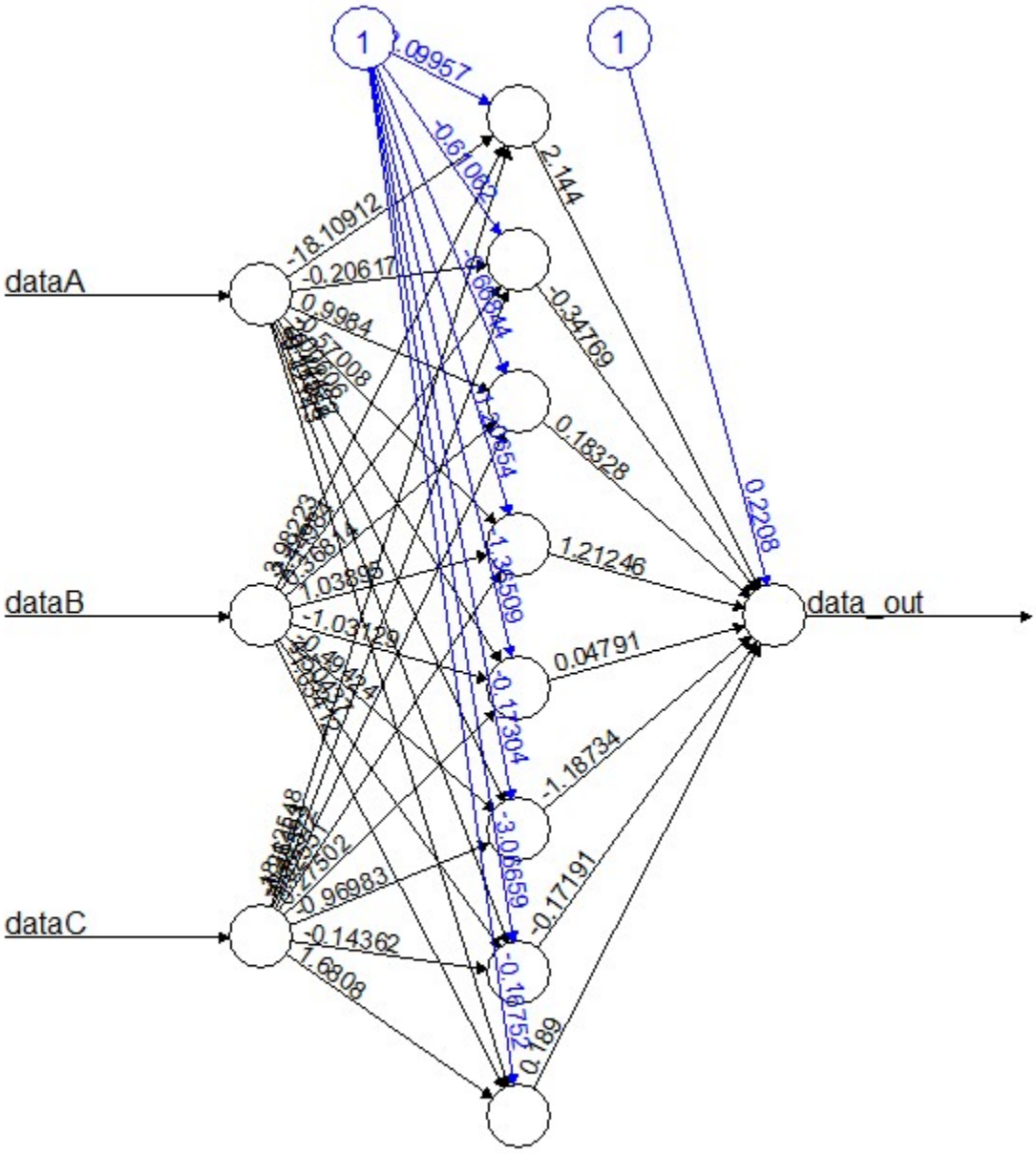
Model2 for 3inputs:

Let's train the first neural network, this time around we'll fit with 1 hidden layer with 8 nodes NN.

```
> usd_eur_model3_2 <- neuralnet(data_out ~ dataA + dataB + dataC, data = usd_eur_train_3, hidden = 8)
> plot(usd_eur_model3_2)
```

```
$result.matrix
                                      1
error                     0.897766622134
reached.threshold         0.009890213183
steps                   784.000000000000
Intercept.to.1layhid1    -2.099569453324
dataA.to.1layhid1       -18.109119234286
dataB.to.1layhid1         3.982230224753
dataC.to.1layhid1       -18.125484476187
Intercept.to.1layhid2    -0.610623255079
dataA.to.1layhid2        -0.206173459175
dataB.to.1layhid2         2.449837794107
dataC.to.1layhid2        -1.261828907164
Intercept.to.1layhid3    -0.668441830653
dataA.to.1layhid3         0.998396068930
dataB.to.1layhid3         0.368138250137
dataC.to.1layhid3        -0.425718634638
Intercept.to.1layhid4    -1.206541884665
dataA.to.1layhid4        -0.570078022502
dataB.to.1layhid4         1.038949013961
dataC.to.1layhid4         1.925514847133
Intercept.to.1layhid5    -1.365089322083
dataA.to.1layhid5         2.006055324571
dataB.to.1layhid5        -1.031293548757
dataC.to.1layhid5         0.275017228754
Intercept.to.1layhid6    -0.173035519651
dataA.to.1layhid6        -0.444222491446
dataB.to.1layhid6        -0.494236925695
dataC.to.1layhid6        -0.969831642333
Intercept.to.1layhid7    -3.066588121326
dataA.to.1layhid7        -0.118676505490
dataB.to.1layhid7         4.504372060406
dataC.to.1layhid7        -0.143621974364
Intercept.to.1layhid8    -0.167515162328
dataA.to.1layhid8        -0.421431876158
dataB.to.1layhid8        -1.034116866398
dataC.to.1layhid8         1.680804775616
Intercept.to.data_out     0.220804021724
1layhid.1.to.data_out     2.143998302638
1layhid.2.to.data_out    -0.347693499885
1layhid.3.to.data_out     0.183280625207
1layhid.4.to.data_out     1.212464686119
1layhid.5.to.data_out     0.047914730824
1layhid.6.to.data_out    -1.187341464227
1layhid.7.to.data_out    -0.171905040436
1layhid.8.to.data_out     0.189001417735

attr(,"class")
[1] "nn"
>
```

```
> plot(usd_eur_model3_2)
>
```



Error: 0.897767 Steps: 784

We can see from our newly plotted model, we have a reduced in SSE of 0.897767 compared to from the previous model 0.931982. We also have a rise in training steps of 784 compared to 773 from the previous model along with a few bias terms - which are indicated by nodes labelled with the number 1. This should come as no surprise given how complex the model has become. A more complex network takes more iterations to find the optimal weights.

Let's continue with our predictions and correlations.

```
> net_results_3_2 <- neuralnet::compute(usd_eur_model3_2, usd_eur_test_3[, 1:3])
> pred_3_2 <- net_results_3_2$net.result
> cor(pred_3_2, usd_eur_test_3$data_out)
             [,1]
[1,] 0.8970701966
>
```

The correlation here of about 0.9 indicates a strong relationship. This implies that our model is doing a good job with 8 hidden nodes. This is not far different from the previous model. Meanwhile will try to improve the performance adding threashold later.

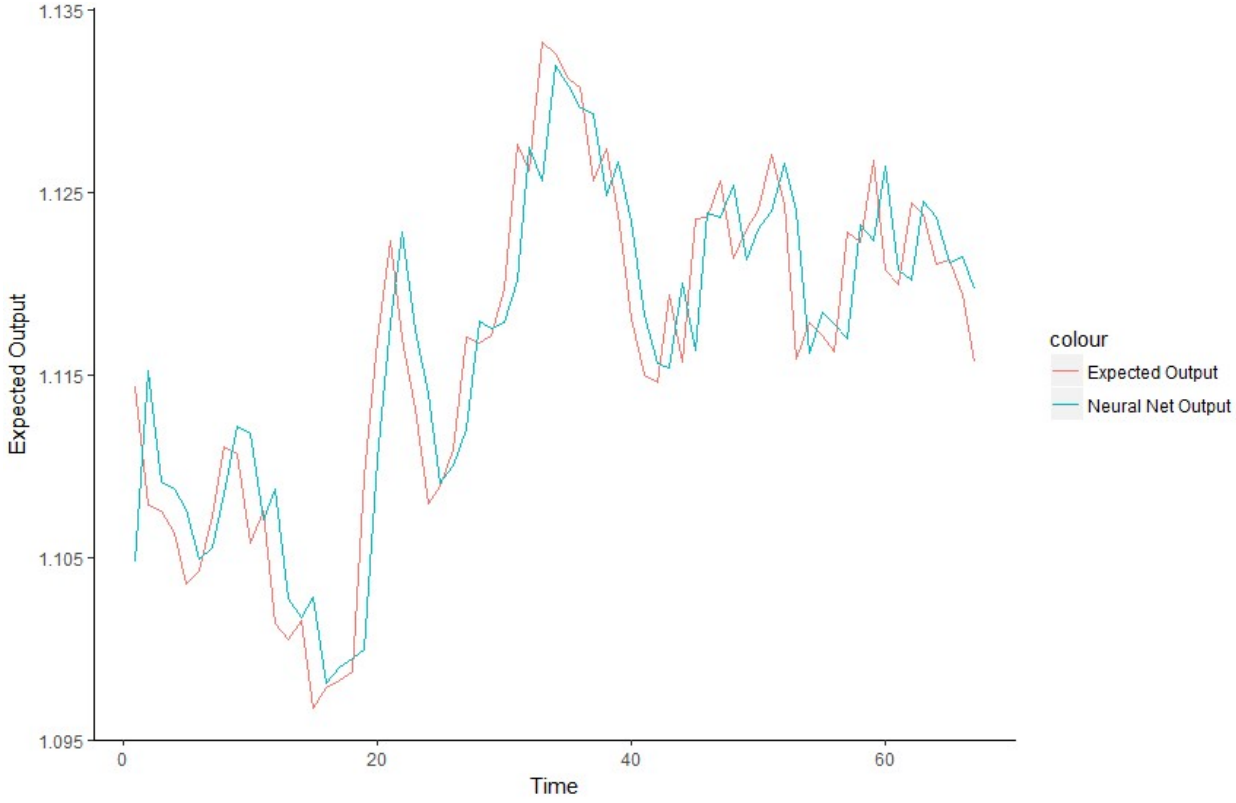
De-normalize the nn predicted output:

```
> pred_all_3_2 <- as.data.frame(net_results_3_2$net.result)
> head(pred_all_3_2)
               V1
321 0.4784327733
322 0.5820334089
323 0.5214769337
324 0.5178727155
325 0.5068520884
326 0.4798836166
> pred_denorm_3_2 <- as.data.frame(Map(denormalize, pred_all_3_2, minvec, maxvec))
> pred_denorm_3_2n <- pred_denorm_3_2[, 1]
> head(pred_denorm_3_2n)
[1] 1.104756613 1.115292798 1.109134204 1.108767655 1.107646857 1.104904164
> length(pred_denorm_3_2n)
[1] 67
> series_3_2 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_denorm_3_2n)
> head(series_3_2)
      test_denorm_out_3 pred_denorm_3_2n
[1,] 1            1.1144      1.104756613
[2,] 2            1.1079      1.115292798
[3,] 3            1.1076      1.109134204
[4,] 4            1.1064      1.108767655
[5,] 5            1.1036      1.107646857
[6,] 6            1.1043      1.104904164
> colnames(series_3_2) <- c("Time","Expected Output","Neural Net Output")
> series_3_2 <- as.data.frame(series_3_2)
> head(series_3_2)
  Time Expected Output Neural Net Output
1    1          1.1144       1.104756613
2    2          1.1079       1.115292798
3    3          1.1076       1.109134204
4    4          1.1064       1.108767655
5    5          1.1036       1.107646857
6    6          1.1043       1.104904164
>
```

We have finally de-normalized both the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_3_2, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



We can see that the plot looks very similar to that of model1, we can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like adding threshold increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse_3_2 <- mse(series_3_2$`Expected Output`, series_3_2$`Neural Net Output`)
> mse_3_2
[1] 0.00001652772332
```

Root mean squared Error

```
> rmse_3_2 <- rsme(series_3_2$`Expected Output`, series_3_2$`Neural Net Output`)
> rmse_3_2
[1] 0.004065430274
```

Mean Percentage Absolute Error

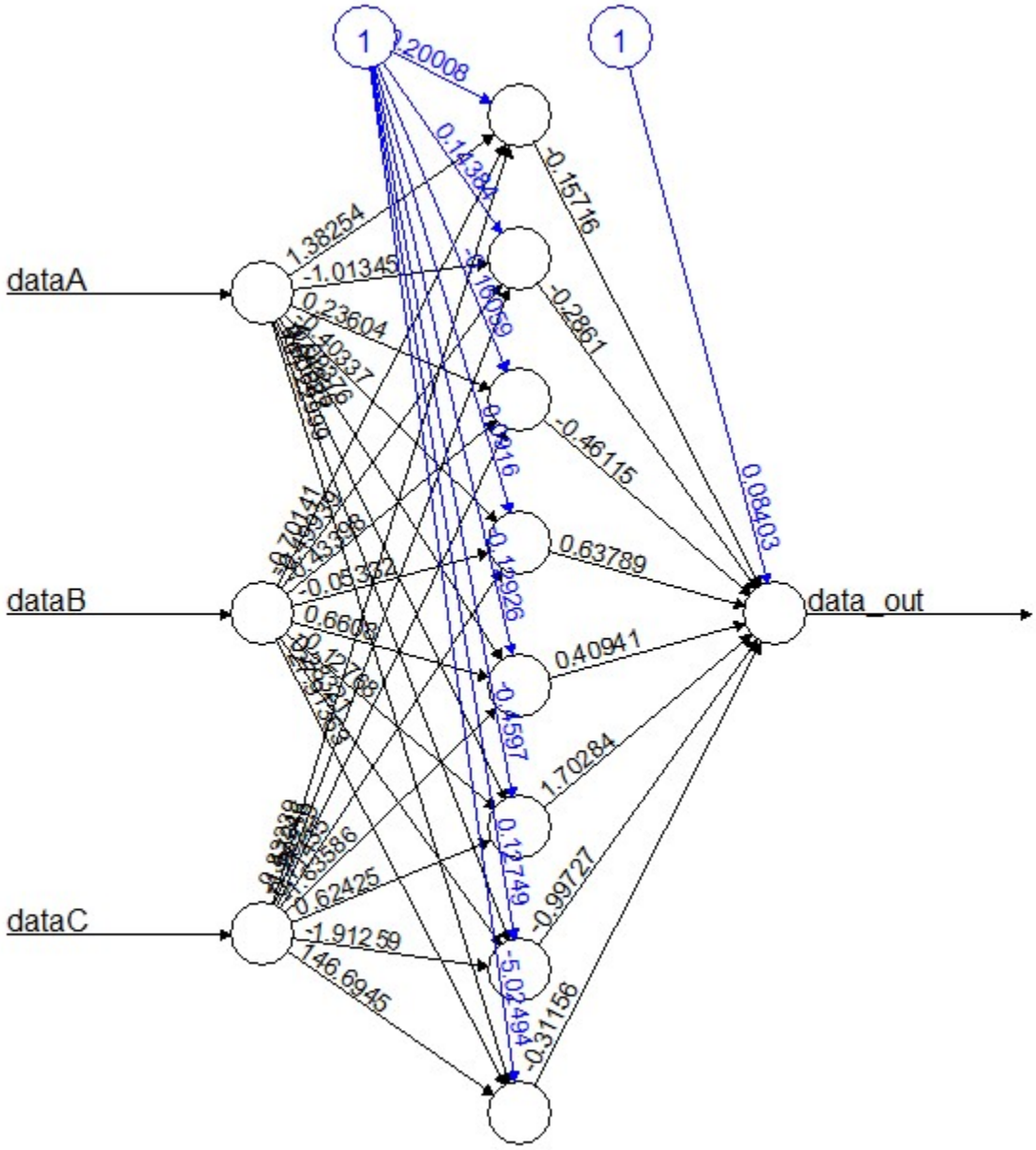
```
> mape_3_2 <- mape(series_3_2$`Expected Output`, series_3_2$`Neural Net Output`)
> mape_3_2
[1] "0.284551%"
>
```

Model3 for 3inputs:

Let's train the third neural network for 3 inputs features, this time around we'll fit with 8 nodes with threshold of 0.01.

```
> usd_eur_model3_3 <- neuralnet(data_out ~ dataA + dataB + dataC, data = usd_eur_train_3, hidden = 8, threshold = 0.01)
$result.matrix
                                    1
error                       0.882187292263
reached.threshold           0.009832258736
steps                    2000.000000000000
Intercept.to.1layhid1      -0.200084528622
dataA.to.1layhid1           1.382543753984
dataB.to.1layhid1          -0.701408287299
dataC.to.1layhid1          -0.832388906685
Intercept.to.1layhid2       0.143837583959
dataA.to.1layhid2          -1.013451475747
dataB.to.1layhid2          -0.499386285025
dataC.to.1layhid2          -1.472458820459
Intercept.to.1layhid3      -0.160587834559
dataA.to.1layhid3           0.236041312728
dataB.to.1layhid3          -0.433977272147
dataC.to.1layhid3          -1.992290272339
Intercept.to.1layhid4       0.091601485253
dataA.to.1layhid4          -0.403371581644
dataB.to.1layhid4          -0.053316910554
dataC.to.1layhid4           0.714352793903
Intercept.to.1layhid5      -0.129261477475
dataA.to.1layhid5          -0.093757402470
dataB.to.1layhid5           0.660797219569
dataC.to.1layhid5          -1.635860421156
Intercept.to.1layhid6      -0.459699500971
dataA.to.1layhid6           0.548350774688
dataB.to.1layhid6          -0.127684678257
dataC.to.1layhid6           0.624254660539
Intercept.to.1layhid7       0.127485288870
dataA.to.1layhid7           0.606873824240
dataB.to.1layhid7           0.283212882532
dataC.to.1layhid7          -1.912585963822
Intercept.to.1layhid8      -5.024938070316
dataA.to.1layhid8         161.229986159059
dataB.to.1layhid8         -27.913629580489
dataC.to.1layhid8         146.694504085351
Intercept.to.data_out       0.084026069647
1layhid.1.to.data_out      -0.157157906675
1layhid.2.to.data_out      -0.286102984856
1layhid.3.to.data_out      -0.461146835557
1layhid.4.to.data_out       0.637891531234
1layhid.5.to.data_out       0.409405041629
1layhid.6.to.data_out       1.702841019600
1layhid.7.to.data_out      -0.997268021562
1layhid.8.to.data_out      -0.311559654513

attr(,"class")
[1] "nn"
>
```

```
> plot(usd_eur_model3_3)
>
```



Error: 0.882187 Steps: 2000

We can see from our newly plotted model, we have a reduced in SSE of 0.882187 compared to from the previous model1 0.931982 and model2 0.897767. We also have a rise in training steps of 2000 compared to 773 and 784 from the previous model along with a few bias terms - which are indicated by nodes labelled with the number 1. This should come as no surprise given how complex the model has

become. A more complex network takes more iterations to find the optimal weights. A lesser SSE denote a reduction in error. Hence a better performance.

Let's continue with our predictions and correlations.

```
> net_results_3_3 <- neuralnet::compute(usd_eur_model3_3, usd_eur_test_3[, 1:3])
> pred_3_3 <- net_results_3_3$net.result
> cor(pred_3_3, usd_eur_test_3$data_out)
              [,1]
[1,] 0.8975258225
>
```

The correlation here of about 0.9 indicates a strong relationship. This implies that our model is doing a good job with 8 hidden nodes and threshold of 0.01. This is not far different from the previous model. Meanwhile will try to improve the performance adding more hidden layers later.

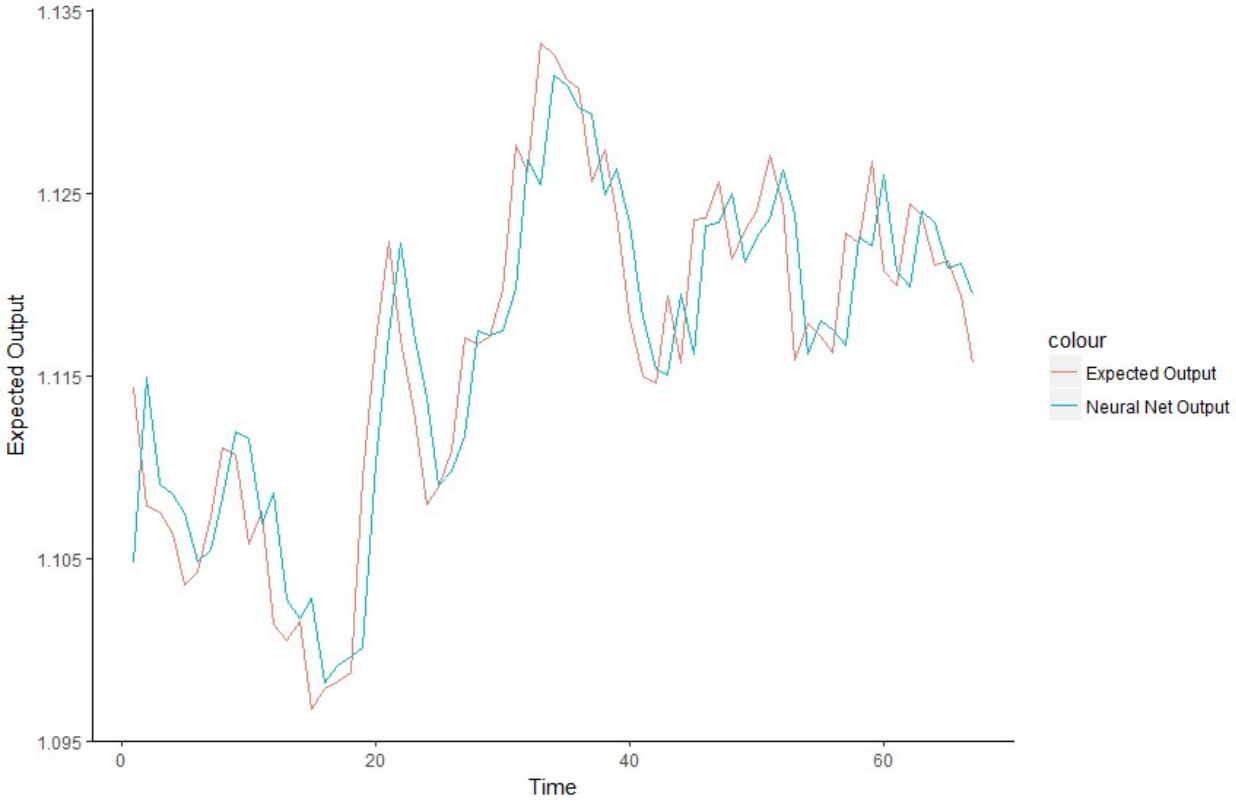
De-normalize the nn predicted output:

```
> pred_all_3_3 <- as.data.frame(net_results_3_3$net.result)
> head(pred_all_3_3)
                 V1
321 0.4783041539
322 0.5783240310
323 0.5207496790
324 0.5159171567
325 0.5054673167
326 0.4790529588
> pred_denorm_3_3 <- as.data.frame(Map(denormalize, pred_all_3_3, minvec, maxvec))
> pred_denorm_3_3n <- pred_denorm_3_3[, 1]
> head(pred_denorm_3_3n)
[1] 1.104743532 1.114915554 1.109060242 1.108568775 1.107506026 1.104819686
> length(pred_denorm_3_3n)
[1] 67
> series_3_3 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_denorm_3_3n)
> head(series_3_3)
      test_denorm_out_3 pred_denorm_3_3n
[1,] 1            1.1144      1.104743532
[2,] 2            1.1079      1.114915554
[3,] 3            1.1076      1.109060242
[4,] 4            1.1064      1.108568775
[5,] 5            1.1036      1.107506026
[6,] 6            1.1043      1.104819686
> colnames(series_3_3) <- c("Time","Expected Output","Neural Net Output")
> series_3_3 <- as.data.frame(series_3_3)
> head(series_3_3)
  Time Expected Output Neural Net Output
1    1          1.1144       1.104743532
2    2          1.1079       1.114915554
3    3          1.1076       1.109060242
4    4          1.1064       1.108568775
5    5          1.1036       1.107506026
6    6          1.1043       1.104819686
>
```

We have finally de-normalized both the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_3_3, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



We can see that the plot looks very similar to that of model1 and model2, we can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like adding threshold increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse_3_3 <- mse(series_3_3$`Expected Output`, series_3_3$`Neural Net Output`)
> mse_3_3
[1] 0.00001633361239
```

Root mean squared Error

```
> rmse_3_3 <- rsme(series_3_3$`Expected Output`, series_3_3$`Neural Net Output`)
> rmse_3_3
[1] 0.004041486409
```

Mean Percentage Absolute Error

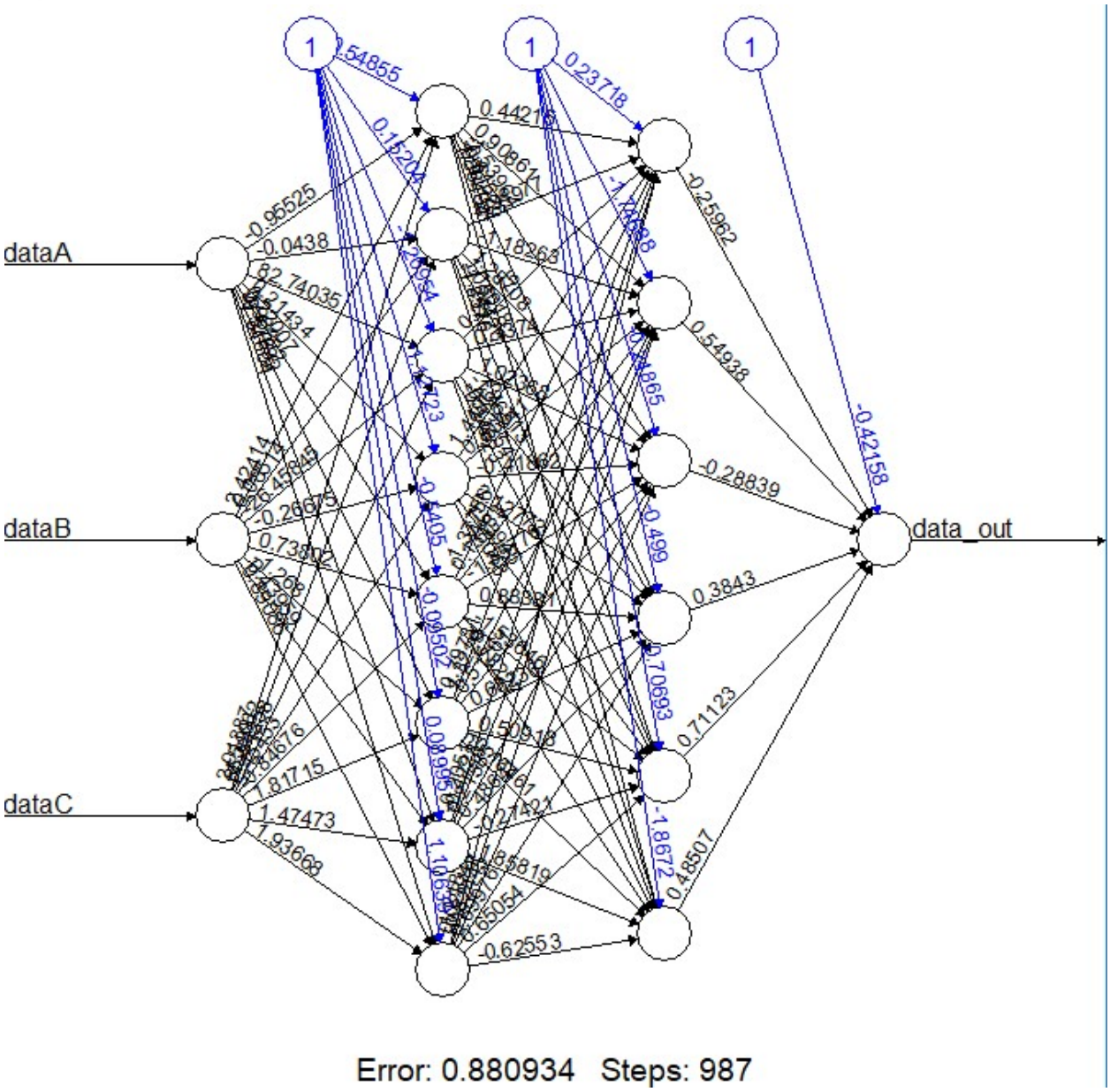
```
> mape_3_3 <- mape(series_3_3$`Expected Output`, series_3_3$`Neural Net Output`)
> mape_3_3
[1] "0.280065%"
>
```

Model4 for 3inputs:

Let's train the first neural network, this time around we'll fit with 8 nodes NN.

```
> usd_eur_model3_4 <- neuralnet(data_out ~ dataA + dataB + dataC, data = usd_eur_train_3, hidden = c(8, 6))
$result.matrix
                                      1
error                      0.880933620358
reached.threshold          0.009264958726
steps                    987.000000000000
Intercept.to.1layhid1     -0.548549821703
dataA.to.1layhid1         -0.955252505150
dataB.to.1layhid1          2.424140257938
dataC.to.1layhid1          2.013867219993
Intercept.to.1layhid2      0.152043471265
dataA.to.1layhid2         -0.043800385825
dataB.to.1layhid2          0.085140444807
dataC.to.1layhid2         -1.540620102998
Intercept.to.1layhid3     -1.269538907425
dataA.to.1layhid3         82.740350406038
dataB.to.1layhid3        -26.458448430346
dataC.to.1layhid3         85.543584938312
Intercept.to.1layhid4     -1.127231070840
dataA.to.1layhid4          0.214335742766
dataB.to.1layhid4         -0.266751404702
dataC.to.1layhid4          2.454525783245
Intercept.to.1layhid5     -0.540498249550
dataA.to.1layhid5          0.220072569614
dataB.to.1layhid5          0.738015848861
dataC.to.1layhid5         -0.846764455236
Intercept.to.1layhid6     -0.095019958000
dataA.to.1layhid6          0.983223765317
dataB.to.1layhid6         -1.268000976285
dataC.to.1layhid6          1.817150867151
Intercept.to.1layhid7      0.089954519840
dataA.to.1layhid7         -0.576917289938
dataB.to.1layhid7          0.439593064328
dataC.to.1layhid7          1.474726937505
Intercept.to.1layhid8      1.106391663925
dataA.to.1layhid8         -1.846037046083
dataB.to.1layhid8          0.890858931118
dataC.to.1layhid8          1.936675484239
Intercept.to.2layhid1      0.237175505635
1layhid.1.to.2layhid1      0.442153350866
1layhid.2.to.2layhid1      1.349770602706
1layhid.3.to.2layhid1      0.277745045671
1layhid.4.to.2layhid1      1.449625868381
1layhid.5.to.2layhid1     -1.377562118797
1layhid.6.to.2layhid1      0.797309460796
1layhid.7.to.2layhid1     -0.205126174593
1layhid.8.to.2layhid1     -1.218240213850
```

```
> plot(usd_eur_model3_4)
>
```



We can see from our newly plotted model, we have a reduced in SSE of 0.880934 compared to from the previous model1 0.931982, model2 0.897767 and model3 0.882187. It is interesting to see this kind of performance even with a reduced in training steps of 987 compared to 773, 784 and 2000 from the previous model along with a few bias terms - which are indicated by nodes labelled with the number 1. This should come as no surprise given how complex the model has become. A more complex network takes more iterations to find the optimal weights. A lesser SSE denote a reduction in error. Hence a better performance.

Let’s continue with our predictions and correlations.

```
> net_results_3_4 <- neuralnet::compute(usd_eur_model3_4, usd_eur_test_3[, 1:3])
> pred_3_4 <- net_results_3_4$net.result
> cor(pred_3_4, usd_eur_test_3$data_out)
              [,1]
[1,] 0.8975355177
>
```

The correlation here of about 0.9 indicates a strong relationship. This implies that our model is doing a good job with 2 hidden layers with 8 nodes in the first layer and 6 nodes in the second layer. This is not far different from the previous model. Meanwhile will try to improve the performance adding more hidden layers later.

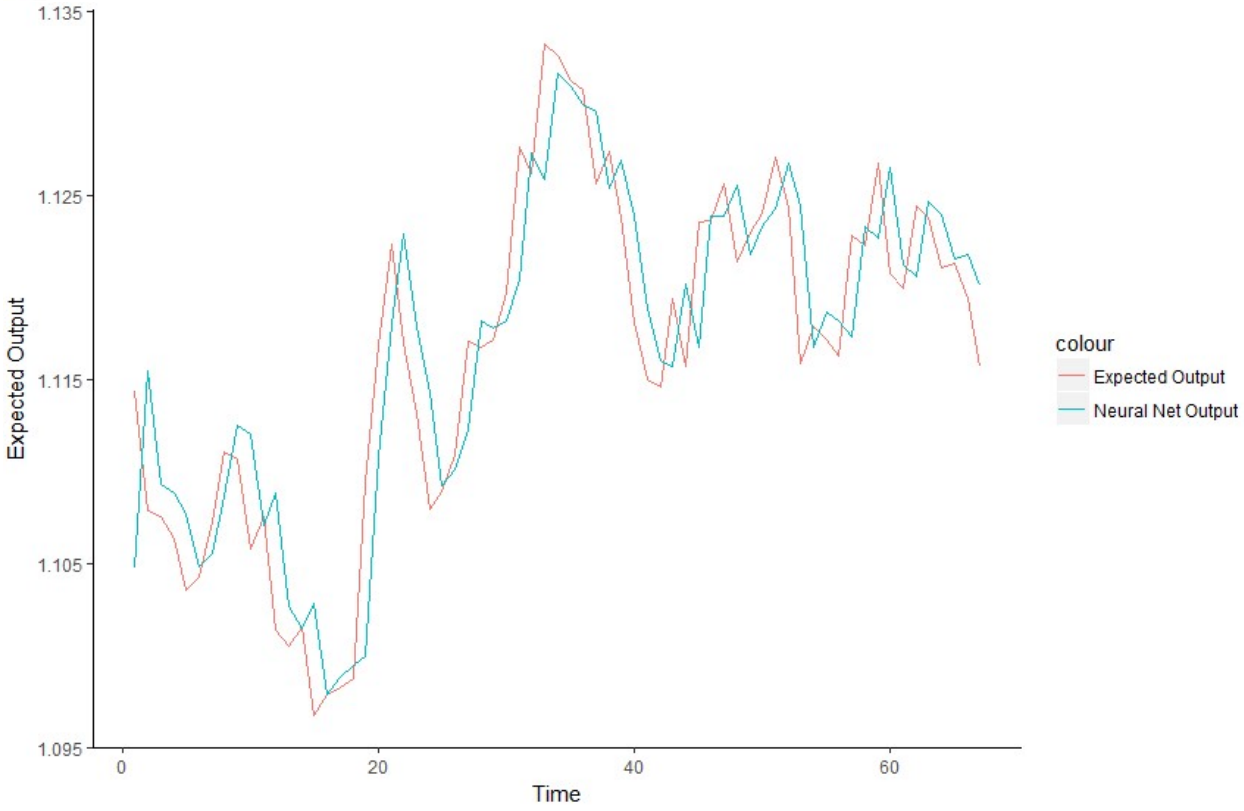
De-normalize the nn predicted output:

```
> pred_all_3_4 <- as.data.frame(net_results_3_4$net.result)
> head(pred_all_3_4)
                V1
321 0.4787896071
322 0.5841630121
323 0.5235401904
324 0.5184075687
325 0.5079463688
326 0.4796749400
> pred_denorm_3_4 <- as.data.frame(Map(denormalize, pred_all_3_4, minvec, maxvec))
> pred_denorm_3_4n <- pred_denorm_3_4[, 1]
> head(pred_denorm_3_4n)
[1] 1.104792903 1.115509378 1.109344037 1.108822050 1.107758146 1.104882941
> length(pred_denorm_3_4n)
[1] 67
> series_3_4 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_denorm_3_4n)
> head(series_3_4)
      test_denorm_out_3 pred_denorm_3_4n
[1,] 1            1.1144      1.104792903
[2,] 2            1.1079      1.115509378
[3,] 3            1.1076      1.109344037
[4,] 4            1.1064      1.108822050
[5,] 5            1.1036      1.107758146
[6,] 6            1.1043      1.104882941
> colnames(series_3_4) <- c("Time","Expected Output","Neural Net Output")
> series_3_4 <- as.data.frame(series_3_4)
> head(series_3_4)
  Time Expected Output Neural Net Output
1    1          1.1144       1.104792903
2    2          1.1079       1.115509378
3    3          1.1076       1.109344037
4    4          1.1064       1.108822050
5    5          1.1036       1.107758146
6    6          1.1043       1.104882941
>
```

We have finally de-normalized both the predicted output.

Next we'll plot the expected output against the predicted output:

```
> ggplot(series_3_4, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



We can see that the plot looks very similar to that of model 1, 2 and 3, we can see that the neural network prediction is moving closely along with the expected output. Meanwhile, we can observe some lagging behind and leading between both the expected output and the neural net output. With this in mind, we'll build another model by trying various parameters like adding threshold increasing the number of hidden layers and so on.

Next we'll calculate the errors and other performance indices to further evaluate the performance:

Mean squared Error

```
> mse_3_4 <- mse(series_3_4$`Expected Output`, series_3_4$`Neural Net Output`)
> mse_3_4
[1] 0.00001667961415
```

Root mean squared Error

```
> rmse_3_4 <- rsme(series_3_4$`Expected Output`, series_3_4$`Neural Net Output`)
> rmse_3_4
[1] 0.004084068333
```

Mean Percentage Absolute Error

```
> mape_3_4 <- mape(series_3_4$`Expected Output`, series_3_4$`Neural Net Output`)
> mape_3_4
[1] "0.286304%"
>
```

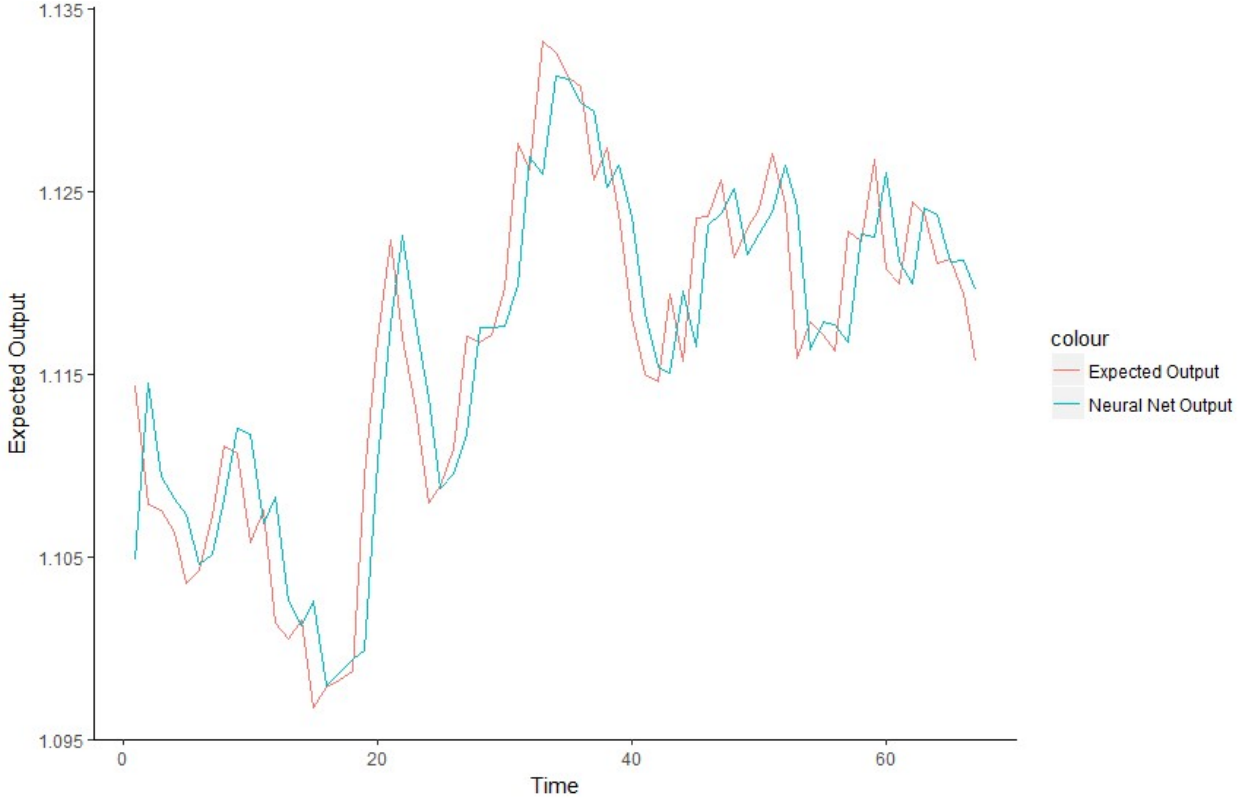
Collating and tabulating the errors to be able to figure out which model so far is the best:

```
> error_2_1 <- c(mse_2_1, rmse_2_1, mape_2_1)
> error_2_2 <- c(mse_2_2, rmse_2_2, mape_2_2)
> error_2_3 <- c(mse_2_3, rmse_2_3, mape_2_3)
> error_2_4 <- c(mse_2_4, rmse_2_4, mape_2_4)
> error_3_1 <- c(mse_3_1, rmse_3_1, mape_3_1)
> error_3_2 <- c(mse_3_2, rmse_3_2, mape_3_2)
> error_3_3 <- c(mse_3_3, rmse_3_3, mape_3_3)
> error_3_4 <- c(mse_3_4, rmse_3_4, mape_3_4)
> error_nn <- cbind(error_2_1, error_2_2, error_2_3, error_2_4, error_3_1, error_3_2, error_3_3, error_3_4)
> colnames(error_nn) <- c("model1_2", "model2_2", "model3_2", "model4_2",
+                         "model1_3", "model2_3", "model3_3", "model4_3")
> error_nn <-  as.data.frame(error_nn)
> error_nn
             model1_2              model2_2              model3_2              model4_2              model1_3
1 0.0000184417761069653 0.0000173547422680275 0.0000174360547404776 0.0000181421019750335 0.0000161682224969046
2   0.00429438890960813   0.00416590233539235   0.00417565021768797   0.00425935464302205   0.00402097282966506
3             0.298549%             0.286674%             0.288677%             0.29769%              0.27865%
             model2_3              model3_3              model4_3
1 0.0000165277233161855 0.0000163336123929337 0.0000166796141491061
2   0.00406543027442182   0.00404148640885179   0.00408406833306032
3             0.284551%             0.280065%             0.286304%
>
```

```
> cor_all <- rbind(cor(pred_2_1, usd_eur_test_2$data_out),
+       cor(pred_2_2, usd_eur_test_2$data_out),
+       cor(pred_2_3, usd_eur_test_2$data_out),
+       cor(pred_2_4, usd_eur_test_2$data_out),
+       cor(pred_3_1, usd_eur_test_3$data_out),
+       cor(pred_3_2, usd_eur_test_3$data_out),
+       cor(pred_3_3, usd_eur_test_3$data_out),
+       cor(pred_3_4, usd_eur_test_3$data_out))
> row.names(cor_all) <- c("cor_model1_2", "cor_model2_2", "cor_model3_2", "cor_model4_2",
+           "cor_model1_3", "cor_model2_3", "cor_model1_3", "cor_model1_3")
> cor_all
                    [,1]
cor_model1_2 0.8909516600
cor_model2_2 0.8923212835
cor_model3_2 0.8925157516
cor_model4_2 0.8919292480
cor_model1_3 0.8993621886
cor_model2_3 0.8970701966
cor_model1_3 0.8975258225
cor_model1_3 0.8975355177
>
```

So far, we can observe that model 1 of three input has the lowest error (MSE , RMSE, and MAPE). It can also be noted that the same model1 has the highest correlation of 0.8993621886 among others. Therefore, we will choose as the best model. It is very amazing that the model with only one hidden and single node outperformed other hyper-parameter optimized model. This maybe because we have few data points because neural network need a lot of data to train very well. Perhaps, if we get more data, we might be able to improve the model better based on hyper parameter tuning. Let visualize our best model:

```
> ggplot(series_3_1, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `Neural Net Output`, colour = "Neural Net Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



4th Objective (SVR)
Let's load the required library:

```
> library(e1071)
> library(tidyverse)
> |
```

Next is to load the exchange dataset and check the first 6 rows:

```
> exchange <- readxl::read_xlsx("Exchange.xls.xlsx", sheet = 1)
> head(exchange)
# A tibble: 6 x 3
  `YYYY/MM/DD`   Wdy `USD/EUR`
         <dttm> <chr>     <dbl>
1   2015-03-19   Thu    1.0621
2   2015-03-20   Fri    1.0791
3   2015-03-23   Mon    1.0927
4   2015-03-24   Tue    1.0906
5   2015-03-25   Wed    1.0986
6   2015-03-26   Thu    1.0918
> |
```

Let's check the structure of the dataset:

```
> str(exchange)
Classes 'tbl_df', 'tbl' and 'data.frame':       390 obs. of  3 variables:
 $ YYYY/MM/DD: POSIXct, format: "2015-03-19" "2015-03-20" "2015-03-23" "2015-03-24" ...
 $ Wdy       : chr  "Thu" "Fri" "Mon" "Tue" ...
 $ USD/EUR   : num  1.06 1.08 1.09 1.09 1.1 ...
> |
```

The USD/EUR column is numeric, meanwhile the variable names are not in the format that we want it. Let's change the variable names to proper names and let's see the effect:

```
> exchange <- rename(exchange, date = `YYYY/MM/DD`, usd_eur = `USD/EUR`)
> head(exchange)
# A tibble: 6 x 3
        date   Wdy usd_eur
      <dttm> <chr>   <dbl>
1 2015-03-19   Thu  1.0621
2 2015-03-20   Fri  1.0791
3 2015-03-23   Mon  1.0927
4 2015-03-24   Tue  1.0906
5 2015-03-25   Wed  1.0986
6 2015-03-26   Thu  1.0918
> nrow(exchange)
[1] 390
> |
```

We can see that we have 390 rows in our dataset and the names as needed.

Next is to get the usd_eur variable, this will be used to formulate our first set of input and output variables for SVR model:

```
> usd_eur <- exchange$usd_eur
> head(usd_eur)
[1] 1.0621 1.0791 1.0927 1.0906 1.0986 1.0918
>
```

Let's check if we have any missing data:

```
> anyNA(usd_eur)
[1] FALSE
>
```

Data preparation and input selection

Let's write function that will convert the usd_eur into the required dataset matrix for SVR First we'll experiment with two inputs and one output. Here we will check the head to see the first six rows of the newly created dataset for the SVR:

```
> create_dataset_2in <- function(dataset){
+   look_back <- 1
+   dataA <- c(); dataB <- c(); data_out <- c()
+   for(i in look_back:(length(dataset)-look_back)){
+
+     dataA <- append(dataA, dataset[i])
+     dataB <- append(dataB, dataset[i + look_back])
+     data_out <- append(data_out, dataset[i + look_back + 1])
+   }
+   result <- as.data.frame(cbind(dataA, dataB, data_out))
+   result
+ }
> usd_eur_checker_2 <- create_dataset_2in(usd_eur)
> head(usd_eur_checker_2, 10)
    dataA  dataB data_out
1  1.0621 1.0791   1.0927
2  1.0791 1.0927   1.0906
3  1.0927 1.0906   1.0986
4  1.0906 1.0986   1.0918
5  1.0986 1.0918   1.0891
6  1.0918 1.0891   1.0817
7  1.0891 1.0817   1.0741
8  1.0817 1.0741   1.0767
9  1.0741 1.0767   1.0876
10 1.0767 1.0876   1.1006
>
```

Let's check the tail to see if we have any NA:

```
> tail(usd_eur_checker_2)
     dataA  dataB data_out
384 1.1200 1.1245   1.1238
385 1.1245 1.1238   1.1211
386 1.1238 1.1211   1.1213
387 1.1211 1.1213   1.1194
388 1.1213 1.1194   1.1157
389 1.1194 1.1157       NA
>
```

We have one NA which need to be dealt with.

Let's remove the NA:

```
> usd_eur_checker_2 <- drop_na(usd_eur_checker_2)
> summary(usd_eur_checker_2)
     dataA            dataB           data_out
 Min.   :1.056   Min.   :1.056   Min.   :1.056
 1st Qu.:1.095   1st Qu.:1.095   1st Qu.:1.095
 Median :1.114   Median :1.114   Median :1.114
 Mean   :1.110   Mean   :1.110   Mean   :1.110
 3rd Qu.:1.126   3rd Qu.:1.126   3rd Qu.:1.126
 Max.   :1.158   Max.   :1.158   Max.   :1.158
> |
```

As expected, the missing value is no more.

Next is normalize the dataset using the normalize function, this will enable the SVR train very well.

```
> normalize <- function(x) {
+   return((x - min(x)) / (max(x) - min(x)))
+ }
> usd_eur_norm_2 <- as.data.frame(lapply(usd_eur_checker_2, normalize))
> summary(usd_eur_norm_2)
     dataA            dataB           data_out
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
 1st Qu.:0.3825   1st Qu.:0.3850   1st Qu.:0.3854
 Median :0.5698   Median :0.5708   Median :0.5723
 Mean   :0.5285   Mean   :0.5300   Mean   :0.5309
 3rd Qu.:0.6849   3rd Qu.:0.6849   3rd Qu.:0.6849
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
> |
```

Let's see the number of row of our new dataset

```
> nrow(usd_eur_norm_2)
[1] 388
> |
```

Now that we are left with 388 rows, we will divide the data into 320 training set and 68 test set.

```
> usd_eur_train_2 <- usd_eur_norm_2[1:320, ] #320 training set
> usd_eur_test_2 <- usd_eur_norm_2[321:388, ] #68 test set
> nrow(usd_eur_train_2)
[1] 320
> nrow(usd_eur_test_2)
[1] 68
> head(usd_eur_train_2)
       dataA     dataB  data_out
1 0.05899705 0.2261554 0.3598820
2 0.22615536 0.3598820 0.3392330
3 0.35988201 0.3392330 0.4178958
4 0.33923304 0.4178958 0.3510324
5 0.41789577 0.3510324 0.3244838
6 0.35103245 0.3244838 0.2517207
> tail(usd_eur_test_2)
        dataA     dataB  data_out
383 0.6361849 0.6283186 0.6725664
384 0.6283186 0.6725664 0.6656834
385 0.6725664 0.6656834 0.6391347
386 0.6656834 0.6391347 0.6411013
387 0.6391347 0.6411013 0.6224189
388 0.6411013 0.6224189 0.5860374
> |
```

As expected we have 320 training set and 68 test set.

Model Fitting:
Model1 linear for 2inputs:
Let's train the SVR model using. We'll build SVR by experimenting with linear and rbf (Radial based function) kernel and also performing performance tuning for each. Meanwhile rbf might not be explicitly stated in the parameter because it is a default in R. First we look into linear kernel: It is either we use radial or default.

```
> set.seed(3211)
>
```

```
> svr_model_2_1 <- svm(data_out ~ dataA + dataB, data = usd_eur_train_2, kernel='linear')
> summary(svr_model_2_1)

Call:
svm(formula = data_out ~ dataA + dataB, data = usd_eur_train_2, kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.5
    epsilon:  0.1


Number of Support Vectors:  238
```

Our model is using cost of 1, gamma of 0.5 and epsilon of 0.1.
Let's now evaluate our model on the test set:

```
> svr_model_2_1_predict <- predict(svr_model_2_1, usd_eur_test_2[, 1:2])
>
```

```
> svr_model_2_1_predict
        321         322         323         324         325         326         327         328         329
0.5393099993 0.4719190610 0.5613630026 0.5128871628 0.5056173022 0.4949940944 0.4698475390 0.4739079394 0.5004300517
        330         331         332         333         334         335         336         337         338
0.5355423995 0.5348020818 0.4920164502 0.5041092052 0.4516284056 0.4393701048 0.4482642291 0.4074231233 0.4135164961
        339         340         341         342         343         344         345         346         347
0.4177757067 0.4223995322 0.5146938632 0.5873546956 0.6404430139 0.5975578478 0.5607219606 0.5128927068 0.5169641954
        348         349         350         351         352         353         354         355         356
0.5349567982 0.5901665352 0.5920170707 0.5952708834 0.6172409844 0.6884214401 0.6811568648 0.7416589308 0.7406865385
        357         358         359         360         361         362         363         364         365
0.7289087604 0.7236386076 0.6790465343 0.6901283473 0.6601258917 0.6081036973 0.5771235642 0.5714280707 0.6127718754
        366         367         368         369         370         371         372         373         374
0.5841283444 0.6499890096 0.6565296406 0.6739477302 0.6380895207 0.6488785330 0.6595680108 0.6863773797 0.6651143320
        375         376         377         378         379         380         381         382         383
0.5894536791 0.6006957526 0.5960608390 0.5877523160 0.6443486980 0.6438845487 0.6824827838 0.6336755936 0.6224282348
        384         385         386         387         388
0.6608828417 0.6580432816 0.6341231106 0.6339187564 0.6175834229
>
```

Now we'll de-normalize the prediction output and desired output:
The de-normalize function:

```
> minvec <- sapply(usd_eur_checker_2, min)
> maxvec <- sapply(usd_eur_checker_2, max)
> denormalize <- function(x, minval, maxval) {
+   x*(maxval - minval) + minval
+ }
>
```

Let's now apply it to both the output of the test data and the predicted result from neural network:

De-normalize the training set and subset the expected output:

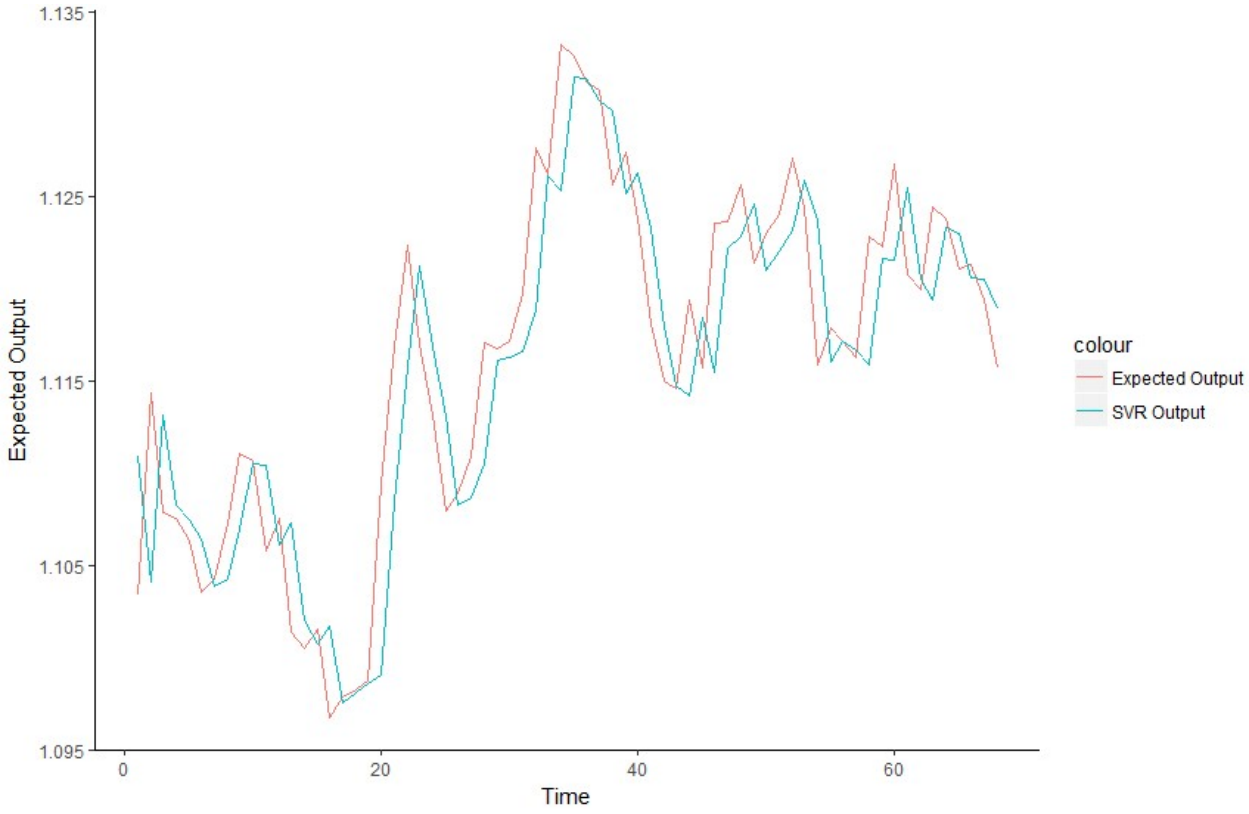
```
> test_denorm_2 <- as.data.frame(Map(denormalize, usd_eur_test_2, minvec, maxvec))
> test_denorm_out_2 <- test_denorm_2$data_out
> head(test_denorm_out_2)
[1] 1.1034 1.1144 1.1079 1.1076 1.1064 1.1036
> length(test_denorm_out_2)
[1] 68
>
```

De-normalize the predicted output:

```
> pred_svr_2_1 <- as.data.frame(svr_model_2_1_predict)
> head(pred_svr_2_1)
    svr_model_2_1_predict
321          0.5393099993
322          0.4719190610
323          0.5613630026
324          0.5128871628
325          0.5056173022
326          0.4949940944
> pred_svr_denorm_2_1 <- as.data.frame(Map(denormalize, pred_svr_2_1, minvec, maxvec))
> pred_svr_denorm_2_1 <- pred_svr_denorm_2_1[, 1]
> head(pred_svr_denorm_2_1)
[1] 1.110947827 1.104094169 1.113190617 1.108260624 1.107521280 1.106440899
> length(pred_svr_denorm_2_1)
[1] 68
> series_svr_2_1 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_svr_denorm_2_1)
> head(series_svr_2_1)
      test_denorm_out_2 pred_svr_denorm_2_1
[1,] 1            1.1034         1.110947827
[2,] 2            1.1144         1.104094169
[3,] 3            1.1079         1.113190617
[4,] 4            1.1076         1.108260624
[5,] 5            1.1064         1.107521280
[6,] 6            1.1036         1.106440899
> colnames(series_svr_2_1) <- c("Time","Expected Output","SVR Output")
> series_svr_2_1 <- as.data.frame(series_svr_2_1)
> head(series_svr_2_1)
  Time Expected Output  SVR Output
1    1          1.1034 1.110947827
2    2          1.1144 1.104094169
3    3          1.1079 1.113190617
4    4          1.1076 1.108260624
5    5          1.1064 1.107521280
6    6          1.1036 1.106440899
>
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_2_1, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



Mean Square Error:

```
> mse <- function(actual, predicted){
+   sum((actual - predicted)^2)/length(actual)
+ }
> mse_svr_2_1 <- mse(series_svr_2_1$`Expected Output`, series_svr_2_1$`SVR Output`)
> mse_svr_2_1
[1] 0.00001779401962
>
```

Root Mean Square Error:

```
> rsme <- function(actual, predicted){
+   sqrt(sum((actual - predicted)^2)/length(actual))
+ }
> rmse_svr_2_1 <- rsme(series_svr_2_1$`Expected Output`, series_svr_2_1$`SVR Output`)
> rmse_svr_2_1
[1] 0.004218295819
>
```

Mean Percentage Absolute Error:

```
> mape <- function(actual, predicted){
+   paste0(round(mean(100*abs((actual - predicted)/ actual)), 6) , "%")
+ }
> mape_svr_2_1 <- mape(series_svr_2_1$`Expected Output`, series_svr_2_1$`SVR Output`)
> mape_svr_2_1
[1] "0.286406%"
>
```

Next we'll try to optimize the model by tuning the performance of the linear kernel.
Model2 tuned linear for 2inputs:
Tune model:

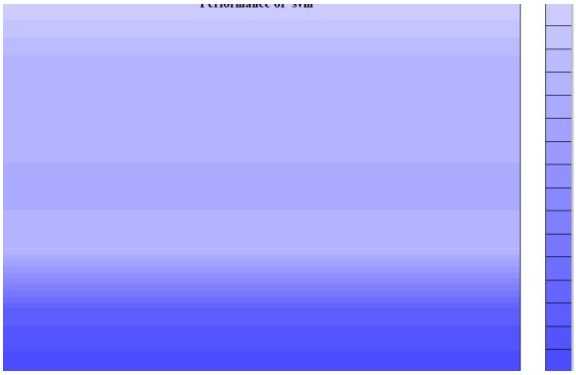
```
> svr_model_2_2_tune <- tune(svm, data_out ~ dataA + dataB, data = usd_eur_train_2, kernel='linear',
+                     ranges = list(gamma = seq(.05,.11,.01), cost = seq(1,4,0.5)),
+                     tunecontrol = tune.control(sampling = "cross"))
>
```

Let's summarize the out output

```
> summary(svr_model_2_2_tune)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  0.05    1

- best performance: 0.006098064812

- Detailed performance results:
   gamma cost           error     dispersion
1   0.05  1.0 0.006098064812 0.001883992744
2   0.06  1.0 0.006098064812 0.001883992744
3   0.07  1.0 0.006098064812 0.001883992744
4   0.08  1.0 0.006098064812 0.001883992744
5   0.09  1.0 0.006098064812 0.001883992744
6   0.10  1.0 0.006098064812 0.001883992744
7   0.11  1.0 0.006098064812 0.001883992744
8   0.05  1.5 0.006099324369 0.001883304133
9   0.06  1.5 0.006099324369 0.001883304133
10  0.07  1.5 0.006099324369 0.001883304133
11  0.08  1.5 0.006099324369 0.001883304133
12  0.09  1.5 0.006099324369 0.001883304133
13  0.10  1.5 0.006099324369 0.001883304133
14  0.11  1.5 0.006099324369 0.001883304133
15  0.05  2.0 0.006104512158 0.001878539371
16  0.06  2.0 0.006104512158 0.001878539371
17  0.07  2.0 0.006104512158 0.001878539371
18  0.08  2.0 0.006104512158 0.001878539371
19  0.09  2.0 0.006104512158 0.001878539371
20  0.10  2.0 0.006104512158 0.001878539371
21  0.11  2.0 0.006104512158 0.001878539371
22  0.05  2.5 0.006103712717 0.001878235687
23  0.06  2.5 0.006103712717 0.001878235687
24  0.07  2.5 0.006103712717 0.001878235687
25  0.08  2.5 0.006103712717 0.001878235687
26  0.09  2.5 0.006103712717 0.001878235687
```

```
> ls(svr_model_2_2_tune)
[1] "best.model"       "best.parameters"  "best.performance" "method"           "nparcomb"         "performances"
[7] "sampling"         "train.ind"
>
```

```
> plot(svr_model_2_2_tune)
>
```



Darker regions imply better accuracy. The use of this plot is to determine the possible range where we can narrow down our search to and try further tuning if required. For instance, this plot show that we can run tuning for epsilon in the new range of 0 to 0.2 and while we are doing this, we can also use even lower steps (say 0.002) but going further may lead to over-fitting. Let's see how the best model looks like when plotted

```
> svr_model_2_2_tune$best.parameters
  gamma cost
1  0.05    1
>
```

The tuning suggests that we use gamma of 0.05 and cost of 1.

Let's feed our best parameters into the model:

```
> svr_model_2_2 <- svm(data_out ~ dataA + dataB, data = usd_eur_train_2, kernel='linear',
+                      gamma = 0.05, cost = 1)
> summary(svr_model_2_2)

Call:
svm(formula = data_out ~ dataA + dataB, data = usd_eur_train_2, kernel = "linear", gamma = 0.05, cost = 1)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.05
    epsilon:  0.1


Number of Support Vectors:  238
```

Our model is using cost of 1, gamma of 0.05 and epsilon of 0.1.
Let's now evaluate our model on the test set:

```
> svr_model_2_2_predict <- predict(svr_model_2_2, usd_eur_test_2[, 1:2])
> svr_model_2_2_predict
        321         322         323         324         325         326         327         328         329
0.5393099993 0.4719190610 0.5613630026 0.5128871628 0.5056173022 0.4949940944 0.4698475390 0.4739079394 0.5004300517
        330         331         332         333         334         335         336         337         338
0.5355423995 0.5348020818 0.4920164502 0.5041092052 0.4516284056 0.4393701048 0.4482642291 0.4074231233 0.4135164961
        339         340         341         342         343         344         345         346         347
0.4177757067 0.4223995322 0.5146938632 0.5873546956 0.6404430139 0.5975578478 0.5607219606 0.5128927068 0.5169641954
        348         349         350         351         352         353         354         355         356
0.5349567982 0.5901665352 0.5920170707 0.5952708834 0.6172409844 0.6884214401 0.6811568648 0.7416589308 0.7406865385
        357         358         359         360         361         362         363         364         365
0.7289087604 0.7236386076 0.6790465343 0.6901283473 0.6601258917 0.6081036973 0.5771235642 0.5714280707 0.6127718754
        366         367         368         369         370         371         372         373         374
0.5841283444 0.6499890096 0.6565296406 0.6739477302 0.6380895207 0.6488785330 0.6595680108 0.6863773797 0.6651143320
        375         376         377         378         379         380         381         382         383
0.5894536791 0.6006957526 0.5960608390 0.5877523160 0.6443486980 0.6438845487 0.6824827838 0.6336755936 0.6224282348
        384         385         386         387         388
0.6608828417 0.6580432816 0.6341231106 0.6339187564 0.6175834229
>
```

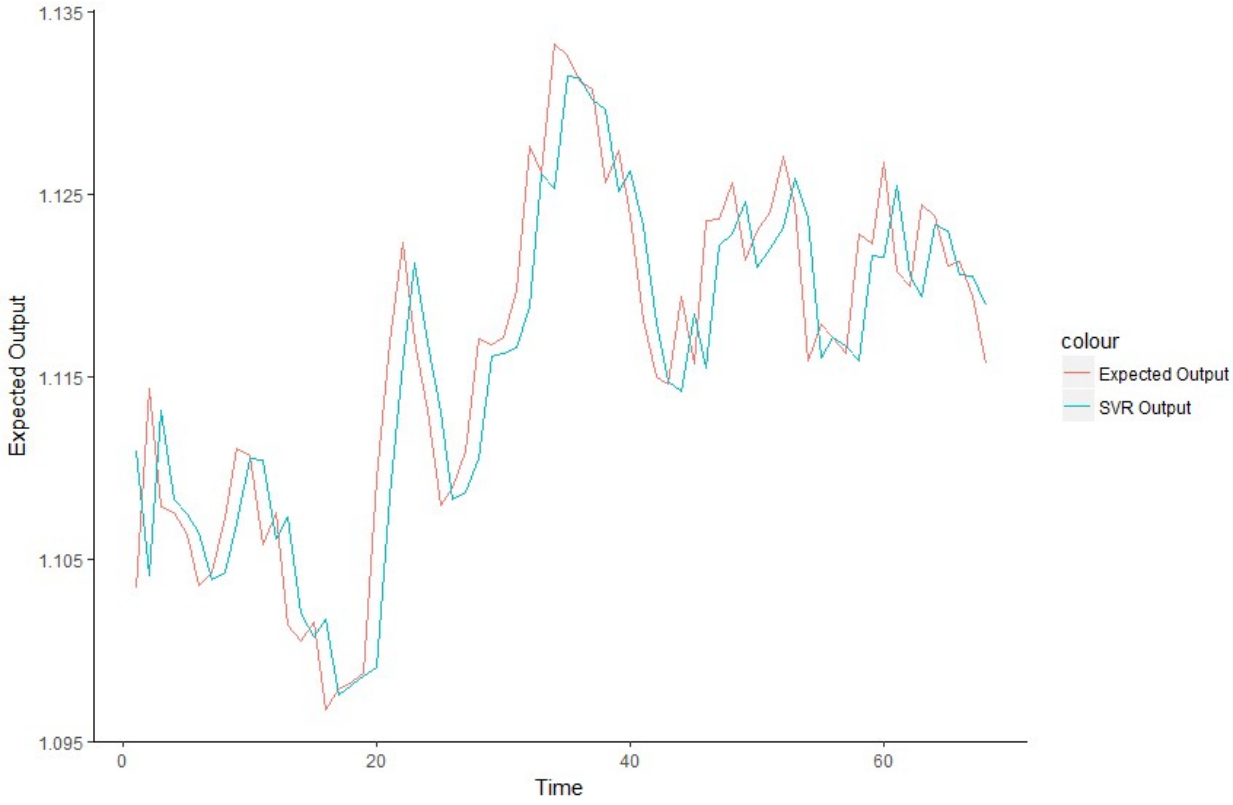
Now we'll de-normalize the prediction output and desired output:
The de-normalize function:

De-normalize the predicted output:

```
> pred_svr_2_2 <- as.data.frame(svr_model_2_2_predict)
> head(pred_svr_2_2)
    svr_model_2_2_predict
321          0.5393099993
322          0.4719190610
323          0.5613630026
324          0.5128871628
325          0.5056173022
326          0.4949940944
> pred_svr_denorm_2_2 <- as.data.frame(Map(denormalize, pred_svr_2_2, minvec, maxvec))
> pred_svr_denorm_2_2 <- pred_svr_denorm_2_2[, 1]
> head(pred_svr_denorm_2_2)
[1] 1.110947827 1.104094169 1.113190617 1.108260624 1.107521280 1.106440899
> length(pred_svr_denorm_2_2)
[1] 68
> series_svr_2_2 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_svr_denorm_2_2)
> head(series_svr_2_2)
      test_denorm_out_2 pred_svr_denorm_2_2
[1,] 1            1.1034         1.110947827
[2,] 2            1.1144         1.104094169
[3,] 3            1.1079         1.113190617
[4,] 4            1.1076         1.108260624
[5,] 5            1.1064         1.107521280
[6,] 6            1.1036         1.106440899
> colnames(series_svr_2_2) <- c("Time","Expected Output","SVR Output")
> series_svr_2_2 <- as.data.frame(series_svr_2_2)
> head(series_svr_2_2)
  Time Expected Output  SVR Output
1    1          1.1034 1.110947827
2    2          1.1144 1.104094169
3    3          1.1079 1.113190617
4    4          1.1076 1.108260624
5    5          1.1064 1.107521280
6    6          1.1036 1.106440899
>
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_2_2, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



Mean Square Error:

```
> mse_svr_2_2 <- mse(series_svr_2_2$`Expected Output`, series_svr_2_2$`SVR Output`)
> mse_svr_2_2
[1] 0.00001779401962
```

Root Mean Square Error:

```
> rmse_svr_2_2 <- rsme(series_svr_2_2$`Expected Output`, series_svr_2_2$`SVR Output`)
> rmse_svr_2_2
[1] 0.004218295819
```

Mean Percentage Absolute Error:

```
> mape_svr_2_2 <- mape(series_svr_2_2$`Expected Output`, series_svr_2_2$`SVR Output`)
> mape_svr_2_2
[1] "0.286406%"
>
```

Model3, 2inputs
Next, we'll apply a default nonlinear kernel called rbf (Here we don't need to specify rbf):

```
> svr_model_2_3 <- svm(data_out ~ dataA + dataB, data = usd_eur_train_2)
> summary(svr_model_2_3)

Call:
svm(formula = data_out ~ dataA + dataB, data = usd_eur_train_2)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.5
    epsilon:  0.1


Number of Support Vectors:  244



>
```

Our model is using cost of 1, gamma of 0.5 and epsilon of 0.1.
Let's now evaluate our model on the test set:

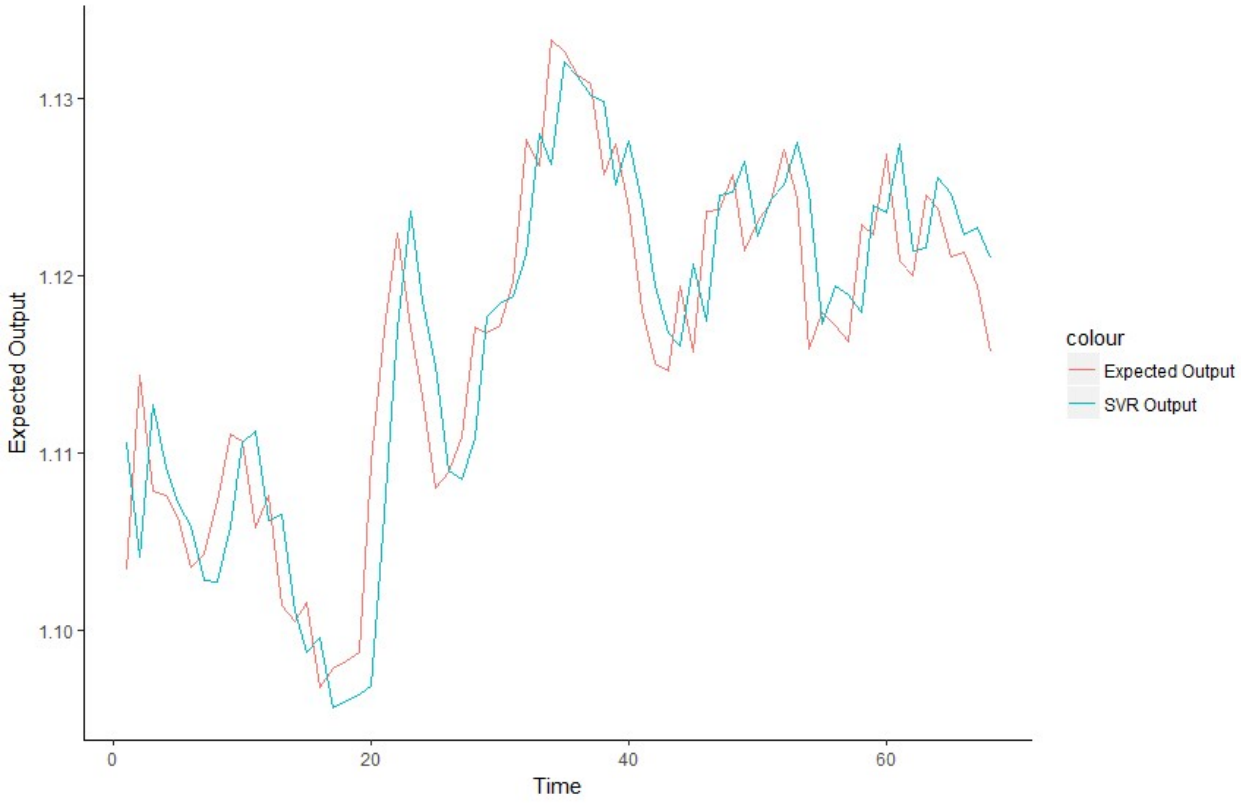
```
> svr_model_2_3_predict <- predict(svr_model_2_3, usd_eur_test_2[, 1:2])
> svr_model_2_3_predict
        321         322         323         324         325         326         327         328         329
0.5358724052 0.4726148324 0.5566226877 0.5218429209 0.5022743803 0.4895919925 0.4594190181 0.4587175843 0.4893191583
        330         331         332         333         334         335         336         337         338
0.5357177600 0.5420767375 0.4924177176 0.4963703896 0.4426979885 0.4197167285 0.4277330883 0.3893834207 0.3926432030
        339         340         341         342         343         344         345         346         347
0.3966009872 0.4010437116 0.4960780484 0.5993491714 0.6639213579 0.6143151045 0.5778869988 0.5201917891 0.5155417662
        348         349         350         351         352         353         354         355         356
0.5382450426 0.6052731139 0.6130446255 0.6166226758 0.6403450967 0.7071779176 0.6899229342 0.7466437473 0.7386365287
        357         358         359         360         361         362         363         364         365
0.7277009435 0.7248412901 0.6786360014 0.7031405310 0.6688094292 0.6228666771 0.5963031114 0.5891847592 0.6345427741
        366         367         368         369         370         371         372         373         374
0.6034392788 0.6724169625 0.6742639110 0.6911727818 0.6502421246 0.6700289400 0.6783339609 0.7022320595 0.6749103324
        375         376         377         378         379         380         381         382         383
0.6011701703 0.6223530098 0.6173970887 0.6083114156 0.6674762474 0.6627065791 0.7008621053 0.6422687127 0.6435924411
        384         385         386         387         388
0.6829148316 0.6739977152 0.6507450337 0.6552222994 0.6377776450
>
```

De-normalize the predicted output:

```
> pred_svr_2_3 <- as.data.frame(svr_model_2_3_predict)
> head(pred_svr_2_3)
    svr_model_2_3_predict
321          0.5358724052
322          0.4726148324
323          0.5566226877
324          0.5218429209
325          0.5022743803
326          0.4895919925
> pred_svr_denorm_2_3 <- as.data.frame(Map(denormalize, pred_svr_2_3, minvec, maxvec))
> pred_svr_denorm_2_3 <- pred_svr_denorm_2_3[, 1]
> head(pred_svr_denorm_2_3)
[1] 1.110598224 1.104164928 1.112708527 1.109171425 1.107181304 1.105891506
> length(pred_svr_denorm_2_3)
[1] 68
> series_svr_2_3 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_svr_denorm_2_3)
> head(series_svr_2_3)
      test_denorm_out_2 pred_svr_denorm_2_3
[1,] 1            1.1034         1.110598224
[2,] 2            1.1144         1.104164928
[3,] 3            1.1079         1.112708527
[4,] 4            1.1076         1.109171425
[5,] 5            1.1064         1.107181304
[6,] 6            1.1036         1.105891506
> colnames(series_svr_2_3) <- c("Time","Expected Output","SVR Output")
> series_svr_2_3 <- as.data.frame(series_svr_2_3)
> head(series_svr_2_3)
  Time Expected Output  SVR Output
1    1          1.1034 1.110598224
2    2          1.1144 1.104164928
3    3          1.1079 1.112708527
4    4          1.1076 1.109171425
5    5          1.1064 1.107181304
6    6          1.1036 1.105891506
>
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_2_3, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



Mean Square Error:

```
> mse_svr_2_3 <- mse(series_svr_2_3$`Expected Output`, series_svr_2_3$`SVR Output`)
> mse_svr_2_3
[1] 0.00001960551493
```

Root Mean Square Error:

```
> rmse_svr_2_3 <- rsme(series_svr_2_3$`Expected Output`, series_svr_2_3$`SVR Output`)
> rmse_svr_2_3
[1] 0.004427811529
```

Mean Percentage Absolute Error:

```
> mape_svr_2_3 <- mape(series_svr_2_3$`Expected Output`, series_svr_2_3$`SVR Output`)
> mape_svr_2_3
[1] "0.31499%"
>
```

Next we'll try to optimize the model by tuning the performance of the rbf nonlinear kernel.

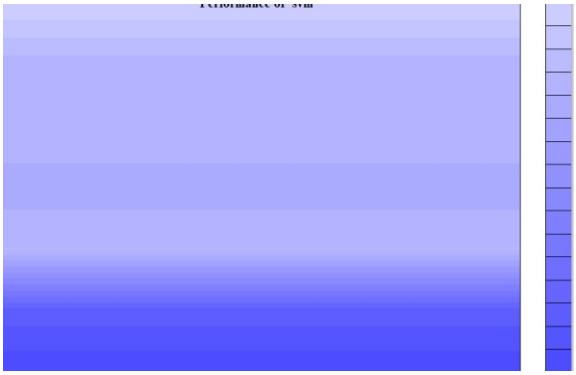
Model4 tuned linear for 2inputs:

Tune model:

```
> svr_model_2_4_tune <- tune(svm, data_out ~ dataA + dataB, data = usd_eur_train_2,
+                            ranges = list(gamma = seq(.05,.11,.01), cost = seq(1,4,0.5)),
+                            tunecontrol = tune.control(sampling = "cross"))
>
```

Let’s summarize the out output

```
> summary(svr_model_2_4_tune)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  0.06    4

- best performance: 0.006038359844

- Detailed performance results:
   gamma cost          error     dispersion
1   0.05  1.0 0.006113789958 0.001328096229
2   0.06  1.0 0.006090950406 0.001333156439
3   0.07  1.0 0.006101621349 0.001373361927
4   0.08  1.0 0.006126403786 0.001410889684
5   0.09  1.0 0.006138167689 0.001411791075
6   0.10  1.0 0.006142941802 0.001404509574
7   0.11  1.0 0.006151899781 0.001413769560
8   0.05  1.5 0.006109661794 0.001427072887
9   0.06  1.5 0.006112173297 0.001426142630
10  0.07  1.5 0.006107491446 0.001434709324
11  0.08  1.5 0.006113188507 0.001442381624
12  0.09  1.5 0.006118153051 0.001433448826
13  0.10  1.5 0.006117770801 0.001446461186
14  0.11  1.5 0.006117916990 0.001475119849
15  0.05  2.0 0.006092376565 0.001427225418
16  0.06  2.0 0.006095671496 0.001431939882
17  0.07  2.0 0.006099705375 0.001435313308
18  0.08  2.0 0.006088477798 0.001464746844
19  0.09  2.0 0.006097881898 0.001489499664
20  0.10  2.0 0.006109422275 0.001508500456
21  0.11  2.0 0.006117137133 0.001518116243
22  0.05  2.5 0.006081242542 0.001429594011
23  0.06  2.5 0.006076711871 0.001457660537
24  0.07  2.5 0.006069937187 0.001482712751
25  0.08  2.5 0.006081908282 0.001496641243
26  0.09  2.5 0.006088222059 0.001506962503
27  0.10  2.5 0.006091424214 0.001513381630
28  0.11  2.5 0.006100492879 0.001530865802
29  0.05  3.0 0.006068529096 0.001459772255
30  0.06  3.0 0.006058747987 0.001477093511
31  0.07  3.0 0.006061177771 0.001493943733
32  0.08  3.0 0.006075764263 0.001507862964
33  0.09  3.0 0.006074405637 0.001521046032

> plot(svr_model_2_2_tune)
>
```



Darker regions imply better accuracy. The use of this plot is to determine the possible range where we can narrow down our search to and try further tuning if required. For instance, this plot show that we can run tuning for epsilon in the new range of 0 to 0.2 and while we are doing this, we can also use even lower steps (say 0.002) but going further may lead to over-fitting. Let's see how the best model looks like when plotted.

```
> svr_model_2_4_tune$best.parameters
   gamma cost
44  0.06    4
>
```

The tuning suggests that we use gamma of 0.06 and cost of 4.

Let's feed our best parameters into the model:

```
> svr_model_2_4 <- svm(data_out ~ dataA + dataB, data = usd_eur_train_2, gamma = 0.06, cost = 4)
> summary(svr_model_2_4)

Call:
svm(formula = data_out ~ dataA + dataB, data = usd_eur_train_2, gamma = 0.06, cost = 4)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  4
      gamma:  0.06
    epsilon:  0.1


Number of Support Vectors:  240



>
```

Our model is using cost of 4, gamma of 0.06 and epsilon of 0.1.
Let's now evaluate our model on the test set:

```
> svr_model_2_4_predict <- predict(svr_model_2_4, usd_eur_test_2[, 1:2])
> svr_model_2_4_predict
        321         322         323         324         325         326         327         328         329
0.5500734481 0.4759840228 0.5724015332 0.5204232609 0.5140641613 0.5025474266 0.4751977348 0.4800782171 0.5087697842
        330         331         332         333         334         335         336         337         338
0.5462046140 0.5451214157 0.4985830899 0.5126561249 0.4547510960 0.4425136786 0.4523620891 0.4072470758 0.4146344157
        339         340         341         342         343         344         345         346         347
0.4192086490 0.4242394204 0.5233236638 0.6000095579 0.6536057330 0.6090727870 0.5716651653 0.5208220378 0.5263233901
        348         349         350         351         352         353         354         355         356
0.5455287294 0.6030518907 0.6047476753 0.6081730989 0.6305997282 0.6991379705 0.6917719015 0.7465721472 0.7445309576
        357         358         359         360         361         362         363         364         365
0.7343772844 0.7301424612 0.6884310568 0.7007730559 0.6713271489 0.6196063608 0.5888641442 0.5835203822 0.6261234772
        366         367         368         369         370         371         372         373         374
0.5959132215 0.6626380564 0.6689157405 0.6857654591 0.6498362467 0.6617552218 0.6719964069 0.6974648325 0.6763721837
        375         376         377         378         379         380         381         382         383
0.5995571323 0.6138608880 0.6088096639 0.6002788283 0.6573297864 0.6565473375 0.6939088674 0.6448462274 0.6353685820
        384         385         386         387         388
0.6734803010 0.6702004664 0.6464760010 0.6469322330 0.6302942299
> |
```

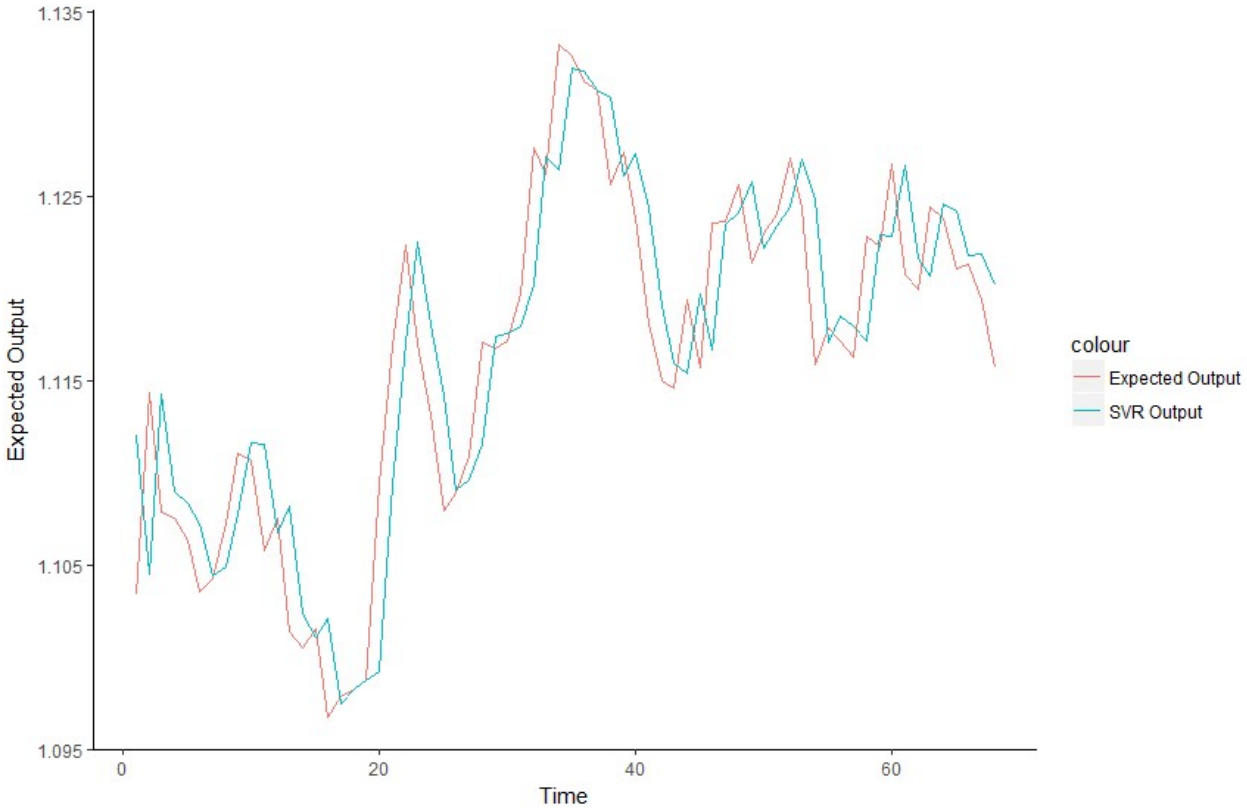
Now we'll de-normalize the prediction output and desired output:
The de-normalize function:

De-normalize the predicted output:

```
> pred_svr_2_4 <- as.data.frame(svr_model_2_4_predict)
> head(pred_svr_2_4)
    svr_model_2_4_predict
321          0.5500734481
322          0.4759840228
323          0.5724015332
324          0.5204232609
325          0.5140641613
326          0.5025474266
> pred_svr_denorm_2_4 <- as.data.frame(Map(denormalize, pred_svr_2_4, minvec, maxvec))
> pred_svr_denorm_2_4 <- pred_svr_denorm_2_4[, 1]
> head(pred_svr_denorm_2_4)
[1] 1.112042470 1.104507575 1.114313236 1.109027046 1.108380325 1.107209073
> length(pred_svr_denorm_2_4)
[1] 68
> series_svr_2_4 <- cbind(seq_along(test_denorm_out_2), test_denorm_out_2, pred_svr_denorm_2_4)
> head(series_svr_2_4)
      test_denorm_out_2 pred_svr_denorm_2_4
[1,] 1            1.1034         1.112042470
[2,] 2            1.1144         1.104507575
[3,] 3            1.1079         1.114313236
[4,] 4            1.1076         1.109027046
[5,] 5            1.1064         1.108380325
[6,] 6            1.1036         1.107209073
> colnames(series_svr_2_4) <- c("Time","Expected Output","SVR Output")
> series_svr_2_4 <- as.data.frame(series_svr_2_4)
> head(series_svr_2_4)
  Time Expected Output  SVR Output
1    1          1.1034 1.112042470
2    2          1.1144 1.104507575
3    3          1.1079 1.114313236
4    4          1.1076 1.109027046
5    5          1.1064 1.108380325
6    6          1.1036 1.107209073
> |
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_2_4, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



Mean Square Error:

```
> mse_svr_2_4 <- mse(series_svr_2_4$`Expected Output`, series_svr_2_4$`SVR Output`)
> mse_svr_2_4
[1] 0.00001793379063
>
```

Root Mean Square Error:

```
> rmse_svr_2_4 <- rsme(series_svr_2_4$`Expected Output`, series_svr_2_4$`SVR Output`)
> rmse_svr_2_4
[1] 0.004234830649
```

Mean Percentage Absolute Error:

```
> mape_svr_2_4 <- mape(series_svr_2_4$`Expected Output`, series_svr_2_4$`SVR Output`)
> mape_svr_2_4
[1] "0.291581%"
>
```

3 INPUTS
Data preparation and input selection
Let's write function that will convert the usd_eur into the required dataset matrix for SVR
First now experiment with three inputs and one output.

```
> create_dataset_3in <- function(dataset){
+   look_back <- 1
+   dataA <- c(); dataB <- c(); dataC <- c(); data_out <- c()
+   for(i in look_back:(length(dataset)-look_back)){
+
+     dataA <- append(dataA, dataset[i])
+     dataB <- append(dataB, dataset[i + look_back])
+     dataC <- append(dataC, dataset[i + look_back + 1])
+     data_out <- append(data_out, dataset[i + look_back + 2])
+   }
+   result <- as.data.frame(cbind(dataA, dataB, dataC, data_out))
+   result
+ }
> usd_eur_checker_3 <- create_dataset_3in(usd_eur)
> head(usd_eur_checker_3, 10)
    dataA  dataB  dataC data_out
1  1.0621 1.0791 1.0927   1.0906
2  1.0791 1.0927 1.0906   1.0986
3  1.0927 1.0906 1.0986   1.0918
4  1.0906 1.0986 1.0918   1.0891
5  1.0986 1.0918 1.0891   1.0817
6  1.0918 1.0891 1.0817   1.0741
7  1.0891 1.0817 1.0741   1.0767
8  1.0817 1.0741 1.0767   1.0876
9  1.0741 1.0767 1.0876   1.1006
10 1.0767 1.0876 1.1006   1.0850
>
```

Let's check the tail to see if we have any NA:

```
> tail(usd_eur_checker_3)
     dataA  dataB  dataC data_out
384 1.1200 1.1245 1.1238   1.1211
385 1.1245 1.1238 1.1211   1.1213
386 1.1238 1.1211 1.1213   1.1194
387 1.1211 1.1213 1.1194   1.1157
388 1.1213 1.1194 1.1157       NA
389 1.1194 1.1157     NA       NA
> str(usd_eur_checker_3)
'data.frame':   389 obs. of  4 variables:
 $ dataA   : num  1.06 1.08 1.09 1.09 1.1 ...
 $ dataB   : num  1.08 1.09 1.09 1.1 1.09 ...
 $ dataC   : num  1.09 1.09 1.1 1.09 1.09 ...
 $ data_out: num  1.09 1.1 1.09 1.09 1.08 ...
>
```

We have one NA which need to be dealt with.

Let's remove the NA:

```
> usd_eur_checker_3 <- drop_na(usd_eur_checker_3)
> summary(usd_eur_checker_3)
     dataA              dataB              dataC             data_out
 Min.   :1.056100   Min.   :1.056100   Min.   :1.056100   Min.   :1.05610
 1st Qu.:1.094900   1st Qu.:1.095200   1st Qu.:1.095300   1st Qu.:1.09530
 Median :1.114000   Median :1.114100   Median :1.114200   Median :1.11440
 Mean   :1.109824   Mean   :1.109976   Mean   :1.110081   Mean   :1.11014
 3rd Qu.:1.125800   3rd Qu.:1.125800   3rd Qu.:1.125800   3rd Qu.:1.12580
 Max.   :1.157800   Max.   :1.157800   Max.   :1.157800   Max.   :1.15780
>
```

As expected, the missing value is no more.

Next is normalize the dataset using the normalize function

```
> normalize <- function(x) {
+   return((x - min(x)) / (max(x) - min(x)))
+ }
> usd_eur_norm_3 <- as.data.frame(lapply(usd_eur_checker_3, normalize))
> summary(usd_eur_norm_3)
     dataA              dataB              dataC             data_out
 Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000   Min.   :0.0000000
 1st Qu.:0.3815143   1st Qu.:0.3844641   1st Qu.:0.3854474   1st Qu.:0.3854474
 Median :0.5693215   Median :0.5703048   Median :0.5712881   Median :0.5732547
 Mean   :0.5282548   Mean   :0.5297590   Mean   :0.5307829   Mean   :0.5313673
 3rd Qu.:0.6853491   3rd Qu.:0.6853491   3rd Qu.:0.6853491   3rd Qu.:0.6853491
 Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000   Max.   :1.0000000
>
```

Let's see the number of row of our new dataset

```
> nrow(usd_eur_norm_3)
[1] 387
>
```

Now that we are left with 387 rows, we will divide the data into 320 training set and 67 test set.

```
> usd_eur_train_3 <- usd_eur_norm_3[1:320, ]
> usd_eur_test_3 <- usd_eur_norm_3[321:387, ]
> nrow(usd_eur_train_3)
[1] 320
> nrow(usd_eur_test_3)
[1] 67
> head(usd_eur_train_3)
          dataA        dataB        dataC     data_out
1 0.05899705015 0.2261553589 0.3598820059 0.3392330383
2 0.22615535890 0.3598820059 0.3392330383 0.4178957719
3 0.35988200590 0.3392330383 0.4178957719 0.3510324484
4 0.33923303835 0.4178957719 0.3510324484 0.3244837758
5 0.41789577188 0.3510324484 0.3244837758 0.2517207473
6 0.35103244838 0.3244837758 0.2517207473 0.1769911504
> tail(usd_eur_test_3)
           dataA        dataB        dataC     data_out
382 0.6951819076 0.6361848574 0.6283185841 0.6725663717
383 0.6361848574 0.6283185841 0.6725663717 0.6656833825
384 0.6283185841 0.6725663717 0.6656833825 0.6391347099
385 0.6725663717 0.6656833825 0.6391347099 0.6411012783
386 0.6656833825 0.6391347099 0.6411012783 0.6224188791
387 0.6391347099 0.6411012783 0.6224188791 0.5860373648
>
```

As expected we have 320 training set and 67 test set.

Model Fitting:
Model1 linear for 3inputs:
Let's train the SVR model using. We'll build SVR by experimenting with linear and rbf kernel and also performing performance tuning for each. Meanwhile rbf might not be explicitly stated in the parameter because it is a default in R.
First we look into linear kernel:

```
> svr_model_3_1 <- svm(data_out ~ dataA + dataB + dataC, data = usd_eur_train_3, kernel='linear')
> summary(svr_model_3_1)

Call:
svm(formula = data_out ~ dataA + dataB + dataC, data = usd_eur_train_3, kernel = "linear")


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  1
      gamma:  0.3333333333
    epsilon:  0.1


Number of Support Vectors:  237
```

Our model is using cost of 1, gamma of 0.5 and epsilon of 0.1.
Let's now evaluate our model on the test set:

```
> svr_model_3_1_predict <- predict(svr_model_3_1, usd_eur_test_3[, 1:3])
> svr_model_3_1_predict
        321         322         323         324         325         326         327         328         329
0.4688138604 0.5601960975 0.5077739749 0.5060751113 0.4930539833 0.4684752328 0.4723558115 0.4970834122 0.5313309207
        330         331         332         333         334         335         336         337         338
0.5312971173 0.4910099458 0.5033804734 0.4497274226 0.4393355825 0.4456348698 0.4051163669 0.4122116575 0.4142626116
        339         340         341         342         343         344         345         346         347
0.4191843495 0.5096591935 0.5795217828 0.6347892084 0.5950856380 0.5620269321 0.5135153411 0.5166815853 0.5320458683
        348         349         350         351         352         353         354         355         356
0.5861973959 0.5879524780 0.5937188803 0.6150544161 0.6845704366 0.6773497106 0.7400842555 0.7375152492 0.7289864201
        357         358         359         360         361         362         363         364         365
0.7237225853 0.6795931131 0.6909609667 0.6592527133 0.6095128348 0.5785063010 0.5710336875 0.6103084126 0.5814167715
        366         367         368         369         370         371         372         373         374
0.6483998513 0.6521747302 0.6724511970 0.6370945111 0.6490858819 0.6575678785 0.6843211955 0.6635252763 0.5910417603
        375         376         377         378         379         380         381         382         383
0.6021463216 0.5938459427 0.5866270590 0.6418912222 0.6401361264 0.6807206119 0.6320436696 0.6237186771 0.6590336100
        384         385         386         387
0.6552898662 0.6338060935 0.6337008191 0.6165706456
>
```

Now we'll de-normalize the prediction output and desired output:
The denormalize function:

```
> minvec <- sapply(usd_eur_checker_3, min)
> maxvec <- sapply(usd_eur_checker_3, max)
> denormalize <- function(x, minval, maxval) {
+   x*(maxval - minval) + minval
+ }
```

Let's now apply it to both the output of the test data and the predicted result from SVR:
De-normalize the training set and subset the expected output:

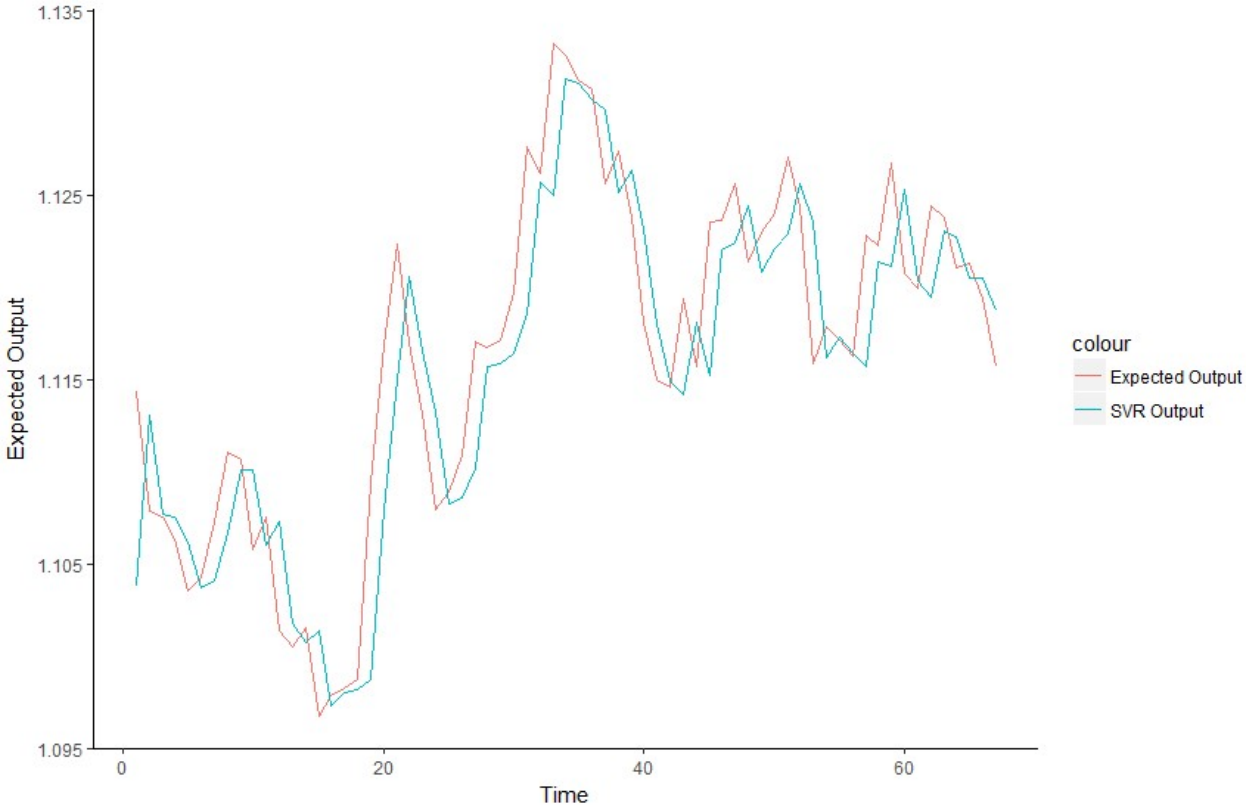
```
> test_denorm_3 <- as.data.frame(Map(denormalize, usd_eur_test_3, minvec, maxvec))
> head(test_denorm_3)
   dataA  dataB  dataC data_out
1 1.1053 1.1117 1.1034   1.1144
2 1.1117 1.1034 1.1144   1.1079
3 1.1034 1.1144 1.1079   1.1076
4 1.1144 1.1079 1.1076   1.1064
5 1.1079 1.1076 1.1064   1.1036
6 1.1076 1.1064 1.1036   1.1043
> test_denorm_out_3 <- test_denorm_3$data_out
> head(test_denorm_out_3)
[1] 1.1144 1.1079 1.1076 1.1064 1.1036 1.1043
> length(test_denorm_out_3)
[1] 67
>
```

De-normalize the predicted output:

```
> pred_svr_3_1 <- as.data.frame(svr_model_3_1_predict)
> head(pred_svr_3_1)
    svr_model_3_1_predict
321          0.4688138604
322          0.5601960975
323          0.5077739749
324          0.5060751113
325          0.4930539833
326          0.4684752328
> pred_svr_denorm_3_1 <- as.data.frame(Map(denormalize, pred_svr_3_1, minvec, maxvec))
> pred_svr_denorm_3_1 <- pred_svr_denorm_3_1[, 1]
> head(pred_svr_denorm_3_1)
[1] 1.103778370 1.113071943 1.107740613 1.107567839 1.106243590 1.103743931
> length(pred_svr_denorm_3_1)
[1] 67
> series_svr_3_1 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_svr_denorm_3_1)
> head(series_svr_3_1)
      test_denorm_out_3 pred_svr_denorm_3_1
[1,] 1            1.1144         1.103778370
[2,] 2            1.1079         1.113071943
[3,] 3            1.1076         1.107740613
[4,] 4            1.1064         1.107567839
[5,] 5            1.1036         1.106243590
[6,] 6            1.1043         1.103743931
> colnames(series_svr_3_1) <- c("Time","Expected Output","SVR Output")
> series_svr_3_1 <- as.data.frame(series_svr_3_1)
> head(series_svr_3_1)
  Time Expected Output  SVR Output
1    1          1.1144 1.103778370
2    2          1.1079 1.113071943
3    3          1.1076 1.107740613
4    4          1.1064 1.107567839
5    5          1.1036 1.106243590
6    6          1.1043 1.103743931
>
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_3_1, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
> |
```



Mean Square Error:

```
> mse_svr_3_1 <- mse(series_svr_3_1$`Expected Output`, series_svr_3_1$`SVR Output`)
> mse_svr_3_1
[1] 0.00001791677062
```

Root Mean Square Error:

```
> rmse_svr_3_1 <- rsme(series_svr_3_1$`Expected Output`, series_svr_3_1$`SVR Output`)
> rmse_svr_3_1
[1] 0.004232820646
```

Mean Percentage Absolute Error:

```
> mape_svr_3_1 <- mape(series_svr_3_1$`Expected Output`, series_svr_3_1$`SVR Output`)
> mape_svr_3_1
[1] "0.286345%"
> |
```

Next we'll try to optimize the model by tuning the performance of the linear kernel.
Tune Model1 for linear (3inputs):
Tune model:

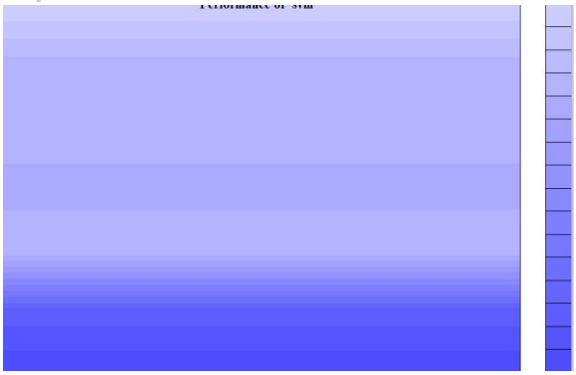
```
> svr_model_3_2_tune <- tune(svm, data_out ~ dataA + dataB + dataC, data = usd_eur_train_3, kernel= "linear",
+                         ranges = list(gamma = seq(.05,.11,.01), cost = seq(1,4,0.5)),
+                         tunecontrol = tune.control(sampling = "cross"))
> |
```

Let's summarize the out output

```
> summary(svr_model_3_2_tune)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  0.05  2.5

- best performance: 0.006180186327

- Detailed performance results:
   gamma cost           error      dispersion
1   0.05  1.0 0.006190413165 0.001420679642
2   0.06  1.0 0.006190413165 0.001420679642
3   0.07  1.0 0.006190413165 0.001420679642
4   0.08  1.0 0.006190413165 0.001420679642
5   0.09  1.0 0.006190413165 0.001420679642
6   0.10  1.0 0.006190413165 0.001420679642
7   0.11  1.0 0.006190413165 0.001420679642
8   0.05  1.5 0.006189517055 0.001422508930
9   0.06  1.5 0.006189517055 0.001422508930
10  0.07  1.5 0.006189517055 0.001422508930
11  0.08  1.5 0.006189517055 0.001422508930
12  0.09  1.5 0.006189517055 0.001422508930
13  0.10  1.5 0.006189517055 0.001422508930
14  0.11  1.5 0.006189517055 0.001422508930
15  0.05  2.0 0.006184377644 0.001421000263
16  0.06  2.0 0.006184377644 0.001421000263
17  0.07  2.0 0.006184377644 0.001421000263
18  0.08  2.0 0.006184377644 0.001421000263
19  0.09  2.0 0.006184377644 0.001421000263
20  0.10  2.0 0.006184377644 0.001421000263
21  0.11  2.0 0.006184377644 0.001421000263
22  0.05  2.5 0.006180186327 0.001424148260
23  0.06  2.5 0.006180186327 0.001424148260
24  0.07  2.5 0.006180186327 0.001424148260
25  0.08  2.5 0.006180186327 0.001424148260
26  0.09  2.5 0.006180186327 0.001424148260
27  0.10  2.5 0.006180186327 0.001424148260
28  0.11  2.5 0.006180186327 0.001424148260
29  0.05  3.0 0.006182793531 0.001420317358
30  0.06  3.0 0.006182793531 0.001420317358
31  0.07  3.0 0.006182793531 0.001420317358
32  0.08  3.0 0.006182793531 0.001420317358
33  0.09  3.0 0.006182793531 0.001420317358
34  0.10  3.0 0.006182793531 0.001420317358
35  0.11  3.0 0.006182793531 0.001420317358
36  0.05  3.5 0.006185934407 0.001416709715
37  0.06  3.5 0.006185934407 0.001416709715
```

```
> ls(svr_model_2_2_tune)
[1] "best.model"       "best.parameters"  "best.performance" "method"           "nparcomb"         "performances"
[7] "sampling"         "train.ind"
> |
```

```
> plot(svr_model_2_2_tune)
>
```



Darker regions imply better accuracy. The use of this plot is to determine the possible range where we can narrow down our search to and try further tuning if required. For instance, this plot show that we can run tuning for epsilon in the new range of 0 to 0.2 and while we are doing this, we can also use even lower steps (say 0.002) but going further may lead to over-fitting. Let's see how the best model looks like when plotted

```
> svr_model_3_2_tune$best.parameters
   gamma cost
22  0.05  2.5
>
```

The tuning suggests that we use gamma of 0.05 and cost of 2.5.

Let's feed our best parameters into the model:

```
> svr_model_3_2 <- svm(data_out ~ dataA + dataB + dataC, data = usd_eur_train_3,kernel = "linear",
+                      gamma = 0.05, cost = 2.5)
> summary(svr_model_3_2)

Call:
svm(formula = data_out ~ dataA + dataB + dataC, data = usd_eur_train_3, kernel = "linear", gamma = 0.05,
    cost = 2.5)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  linear
       cost:  2.5
      gamma:  0.05
    epsilon:  0.1


Number of Support Vectors:  236
```

Our model is using cost of 2.5, gamma of 0.05 and epsilon of 0.1.
Let's now evaluate our model on the test set:

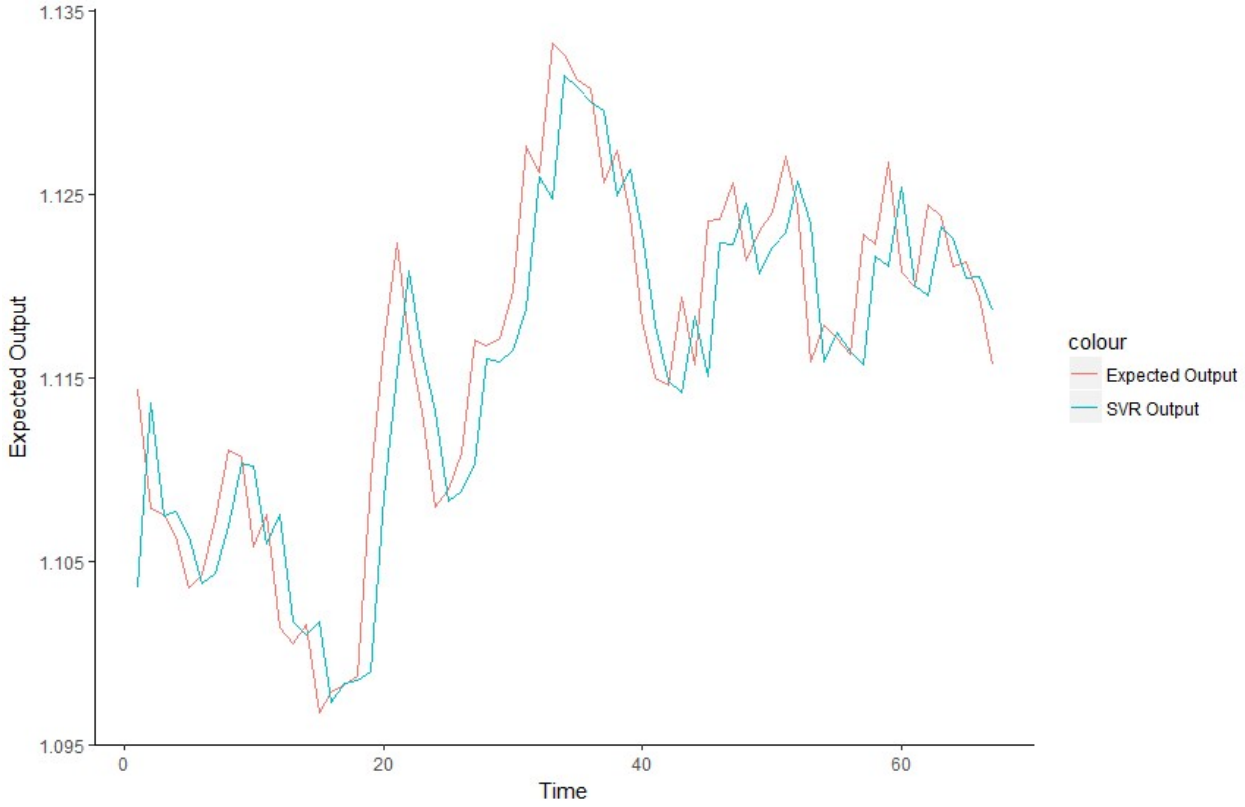
```
> svr_model_3_2_predict <- predict(svr_model_3_2, usd_eur_test_3[, 1:3])
> svr_model_3_2_predict
        321         322         323         324         325         326         327         328         329
0.4666362447 0.5662769573 0.5055918503 0.5077816212 0.4941166919 0.4692050481 0.4746695453 0.4998601461 0.5338788518
        330         331         332         333         334         335         336         337         338
0.5319616531 0.4904920193 0.5059290921 0.4490078751 0.4415874902 0.4483274799 0.4056063306 0.4155780792 0.4169056399
        339         340         341         342         343         344         345         346         347
0.4218565977 0.5156058427 0.5825791648 0.6364965939 0.5926674332 0.5614415153 0.5128142399 0.5186931265 0.5339402672
        348         349         350         351         352         353         354         355         356
0.5891781642 0.5877988357 0.5942317321 0.6161715833 0.6870836161 0.6754455882 0.7417837459 0.7352743024 0.7271090543
        357         358         359         360         361         362         363         364         365
0.7222990663 0.6766581926 0.6911552577 0.6570770660 0.6073911157 0.5780417778 0.5716653846 0.6125400937 0.5799554670
        366         367         368         369         370         371         372         373         374
0.6517304586 0.6512865258 0.6726592927 0.6348683290 0.6497021338 0.6574682271 0.6847514768 0.6616023824 0.5878685987
        375         376         377         378         379         380         381         382         383
0.6037981630 0.5937246313 0.5866878169 0.6445494160 0.6391785558 0.6819624741 0.6289726528 0.6237164036 0.6605574856
        384         385         386         387
0.6542408045 0.6324978010 0.6337909596 0.6157646364
>
```

De-normalize the predicted output:

```
> pred_svr_3_2 <- as.data.frame(svr_model_3_2_predict)
> head(pred_svr_3_2)
    svr_model_3_2_predict
321          0.4666362447
322          0.5662769573
323          0.5055918503
324          0.5077816212
325          0.4941166919
326          0.4692050481
> pred_svr_denorm_3_2 <- as.data.frame(Map(denormalize, pred_svr_3_2, minvec, maxvec))
> pred_svr_denorm_3_2 <- pred_svr_denorm_3_2[, 1]
> head(pred_svr_denorm_3_2)
[1] 1.103556906 1.113690367 1.107518691 1.107741391 1.106351668 1.103818153
> length(pred_svr_denorm_3_2)
[1] 67
> series_svr_3_2 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_svr_denorm_3_2)
> head(series_svr_3_2)
      test_denorm_out_3 pred_svr_denorm_3_2
[1,] 1            1.1144         1.103556906
[2,] 2            1.1079         1.113690367
[3,] 3            1.1076         1.107518691
[4,] 4            1.1064         1.107741391
[5,] 5            1.1036         1.106351668
[6,] 6            1.1043         1.103818153
> colnames(series_svr_3_2) <- c("Time","Expected Output","SVR Output")
> series_svr_3_2 <- as.data.frame(series_svr_3_2)
> head(series_svr_3_2)
  Time Expected Output  SVR Output
1    1          1.1144 1.103556906
2    2          1.1079 1.113690367
3    3          1.1076 1.107518691
4    4          1.1064 1.107741391
5    5          1.1036 1.106351668
6    6          1.1043 1.103818153
>
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_3_2, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



Mean Square Error:

```
> mse_svr_3_2 <- mse(series_svr_3_2$`Expected Output`, series_svr_3_2$`SVR Output`)
> mse_svr_3_2
[1] 0.00001774937813
```

Root Mean Square Error:

```
> rmse_svr_3_2 <- rsme(series_svr_3_2$`Expected Output`, series_svr_3_2$`SVR Output`)
> rmse_svr_3_2
[1] 0.004213001083
```

Mean Percentage Absolute Error:

```
> mape_svr_3_2 <- mape(series_svr_3_2$`Expected Output`, series_svr_3_2$`SVR Output`)
> mape_svr_3_2
[1] "0.283426%"
>
```

Model3, 3inputs
Next, we'll apply a default nonlinear kernel called rbf (Here we don't need to specify rbf):

```
> svr_model_3_3 <- svm(data_out ~ dataA + dataB + dataC, data = usd_eur_train_3)
> summary(svr_model_3_3)

Call:
svm(formula = data_out ~ dataA + dataB + dataC, data = usd_eur_train_3)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.3333333333
    epsilon:  0.1


Number of Support Vectors:  244
```

Our model is using cost of 1, gamma of 0.5 and epsilon of 0.1.
Let's now evaluate our model on the test set:

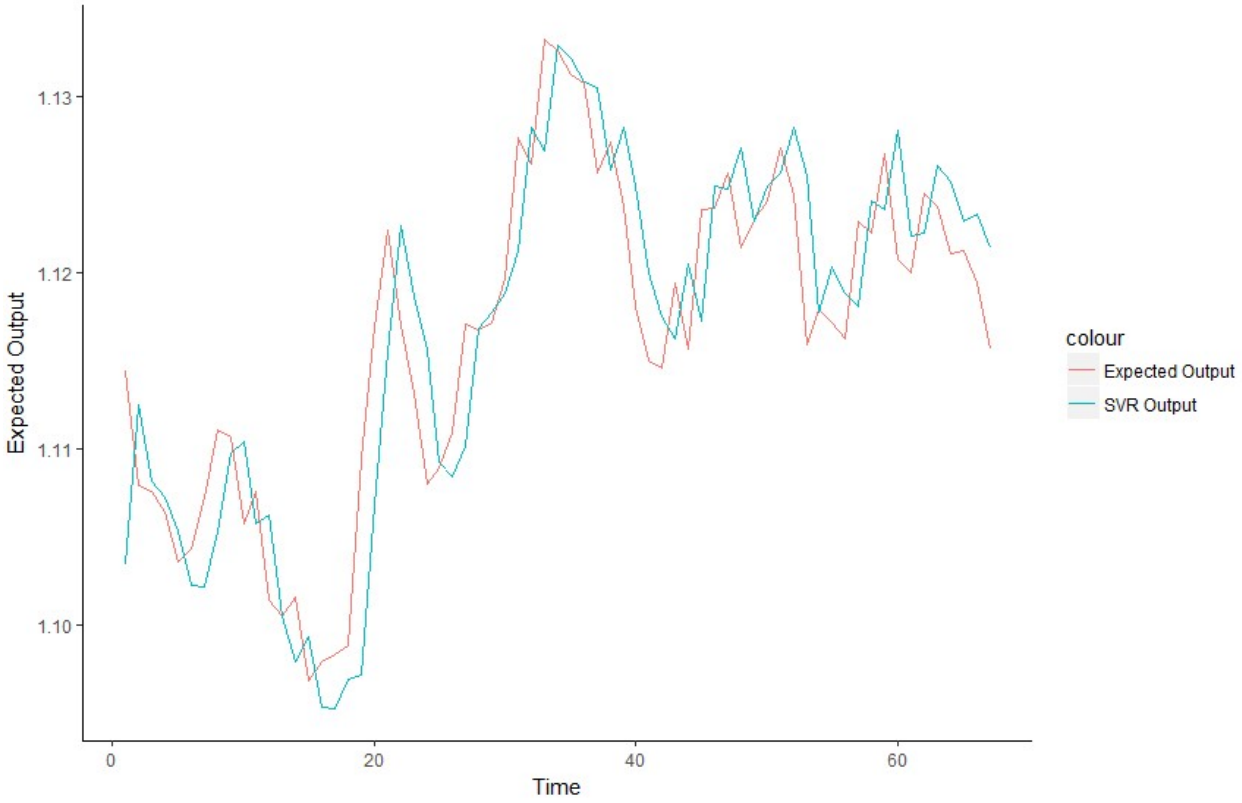
```
> svr_model_3_3_predict <- predict(svr_model_3_3, usd_eur_test_3[, 1:3])
> svr_model_3_3_predict
        321         322         323         324         325         326         327         328         329
0.4652660241 0.5548488910 0.5124114754 0.5032526693 0.4839192093 0.4536598391 0.4530909200 0.4845193270 0.5279583688
        330         331         332         333         334         335         336         337         338
0.5340013438 0.4883405159 0.4930332276 0.4362676396 0.4114289542 0.4251684070 0.3856755495 0.3854025568 0.4015119189
        339         340         341         342         343         344         345         346         347
0.4036073082 0.4990559379 0.5840844017 0.6549152745 0.6151608862 0.5850709938 0.5229758372 0.5147432699 0.5314687798
        348         349         350         351         352         353         354         355         356
0.5968932614 0.6063017210 0.6173573571 0.6407496564 0.7096091960 0.6964175242 0.7555749498 0.7477980870 0.7349491552
        357         358         359         360         361         362         363         364         365
0.7321272728 0.6859875270 0.7093685369 0.6766326372 0.6288353395 0.6038383063 0.5919554136 0.6329799856 0.6012107835
        366         367         368         369         370         371         372         373         374
0.6764145587 0.6755330733 0.6982742693 0.6567694306 0.6770366590 0.6840041558 0.7094944498 0.6827128434 0.6064097035
        375         376         377         378         379         380         381         382         383
0.6315655189 0.6165697998 0.6099208335 0.6687711416 0.6640722779 0.7077335580 0.6486359377 0.6507950294 0.6879124230
        384         385         386         387
0.6791897090 0.6570806250 0.6614795007 0.6421508216
>
```

De-normalize the predicted output:

```
> pred_svr_3_3 <- as.data.frame(svr_model_3_3_predict)
> head(pred_svr_3_3)
    svr_model_3_3_predict
321          0.4652660241
322          0.5548488910
323          0.5124114754
324          0.5032526693
325          0.4839192093
326          0.4536598391
> pred_svr_denorm_3_3 <- as.data.frame(Map(denormalize, pred_svr_3_3, minvec, maxvec))
> pred_svr_denorm_3_3 <- pred_svr_denorm_3_3[, 1]
> head(pred_svr_denorm_3_3)
[1] 1.103417555 1.112528132 1.108212247 1.107280796 1.105314584 1.102237206
> length(pred_svr_denorm_3_3)
[1] 67
> series_svr_3_3 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_svr_denorm_3_3)
> head(series_svr_3_3)
      test_denorm_out_3 pred_svr_denorm_3_3
[1,] 1            1.1144         1.103417555
[2,] 2            1.1079         1.112528132
[3,] 3            1.1076         1.108212247
[4,] 4            1.1064         1.107280796
[5,] 5            1.1036         1.105314584
[6,] 6            1.1043         1.102237206
> colnames(series_svr_3_3) <- c("Time","Expected Output","SVR Output")
> series_svr_3_3 <- as.data.frame(series_svr_3_3)
> head(series_svr_3_3)
  Time Expected Output  SVR Output
1    1          1.1144 1.103417555
2    2          1.1079 1.112528132
3    3          1.1076 1.108212247
4    4          1.1064 1.107280796
5    5          1.1036 1.105314584
6    6          1.1043 1.102237206
>
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_3_3, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



Mean Square Error:

```
> mse_svr_3_3 <- mse(series_svr_3_3$`Expected Output`, series_svr_3_3$`SVR Output`)
> mse_svr_3_3
[1] 0.00002074377469
```

Root Mean Square Error:

```
> rmse_svr_3_3 <- rsme(series_svr_3_3$`Expected Output`, series_svr_3_3$`SVR Output`)
> rmse_svr_3_3
[1] 0.004554533422
```

Mean Percentage Absolute Error:

```
> mape_svr_3_3 <- mape(series_svr_3_3$`Expected Output`, series_svr_3_3$`SVR Output`)
> mape_svr_3_3
[1] "0.322395%"
>
```

Next we'll try to optimize the model by tuning the performance of the rbf nonlinear kernel.

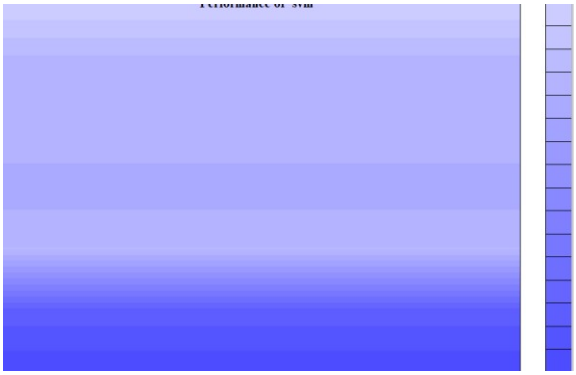
Model4 tuned linear for 2inputs:

Tune model:

```
> svr_model_3_4_tune <- tune(svm, data_out ~ dataA + dataB + dataC, data = usd_eur_train_3,
+                            ranges = list(gamma = seq(.05,.11,.01), cost = seq(1,4,0.5)),
+                            tunecontrol = tune.control(sampling = "cross"))
>
```

Let's summarize the out output

```
> summary(svr_model_3_4_tune)

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
  0.05    3

- best performance: 0.006167574213

- Detailed performance results:
   gamma cost          error     dispersion
1   0.05  1.0 0.006220504106 0.002293246550
2   0.06  1.0 0.006227915947 0.002307664849
3   0.07  1.0 0.006228558004 0.002302857934
4   0.08  1.0 0.006221057142 0.002297404028
5   0.09  1.0 0.006258051588 0.002332534803
6   0.10  1.0 0.006278795993 0.002335172772
7   0.11  1.0 0.006300797921 0.002346520614
8   0.05  1.5 0.006211993886 0.002305928191
9   0.06  1.5 0.006234524729 0.002350560334
10  0.07  1.5 0.006243395056 0.002337276810
11  0.08  1.5 0.006250018954 0.002327511166
12  0.09  1.5 0.006252721194 0.002307488516
13  0.10  1.5 0.006243477112 0.002299499840
14  0.11  1.5 0.006230815371 0.002288970481
15  0.05  2.0 0.006216288021 0.002358850737
16  0.06  2.0 0.006203849986 0.002322710146
17  0.07  2.0 0.006219990843 0.002325322854
18  0.08  2.0 0.006236944966 0.002320529536
19  0.09  2.0 0.006220046329 0.002324472602
20  0.10  2.0 0.006219355151 0.002326507812
21  0.11  2.0 0.006206660924 0.002323484803
22  0.05  2.5 0.006170747054 0.002309696591
23  0.06  2.5 0.006199468247 0.002336378171
24  0.07  2.5 0.006210430491 0.002343494801
25  0.08  2.5 0.006209727859 0.002344860739
26  0.09  2.5 0.006213838996 0.002355511770
27  0.10  2.5 0.006191560425 0.002328426924
28  0.11  2.5 0.006183401893 0.002303409932
29  0.05  3.0 0.006167574213 0.002337977919
30  0.06  3.0 0.006200450496 0.002364845007
31  0.07  3.0 0.006211087465 0.002355664568
32  0.08  3.0 0.006209355821 0.002359898796
33  0.09  3.0 0.006201447427 0.002347238942


> plot(svr_model_3_4_tune)
```



Darker regions imply better accuracy. The use of this plot is to determine the possible range where we can narrow down our search to and try further tuning if required. For instance, this plot show that we can run tuning for epsilon in the new range of 0 to 0.2 and while we are doing this, we can also use even lower steps (say 0.002) but going further may lead to over-fitting. Let's see how the best model looks like when plotted

```
> svr_model_3_4_tune$best.parameters
   gamma cost
29  0.05    3
>
```

The tuning suggests that we use gamma of 0.05 and cost of 3.

Let's feed our best parameters into the model:

```
> svr_model_3_4 <- svm(data_out ~ dataA + dataB, data = usd_eur_train_3, gamma = 0.05, cost = 3)
> summary(svr_model_3_4)

Call:
svm(formula = data_out ~ dataA + dataB, data = usd_eur_train_3, gamma = 0.05, cost = 3)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  3
      gamma:  0.05
    epsilon:  0.1


Number of Support Vectors:  273
```

Our model is using cost of 4, gamma of 0.06 and epsilon of 0.1.
Let's now evaluate our model on the test set:

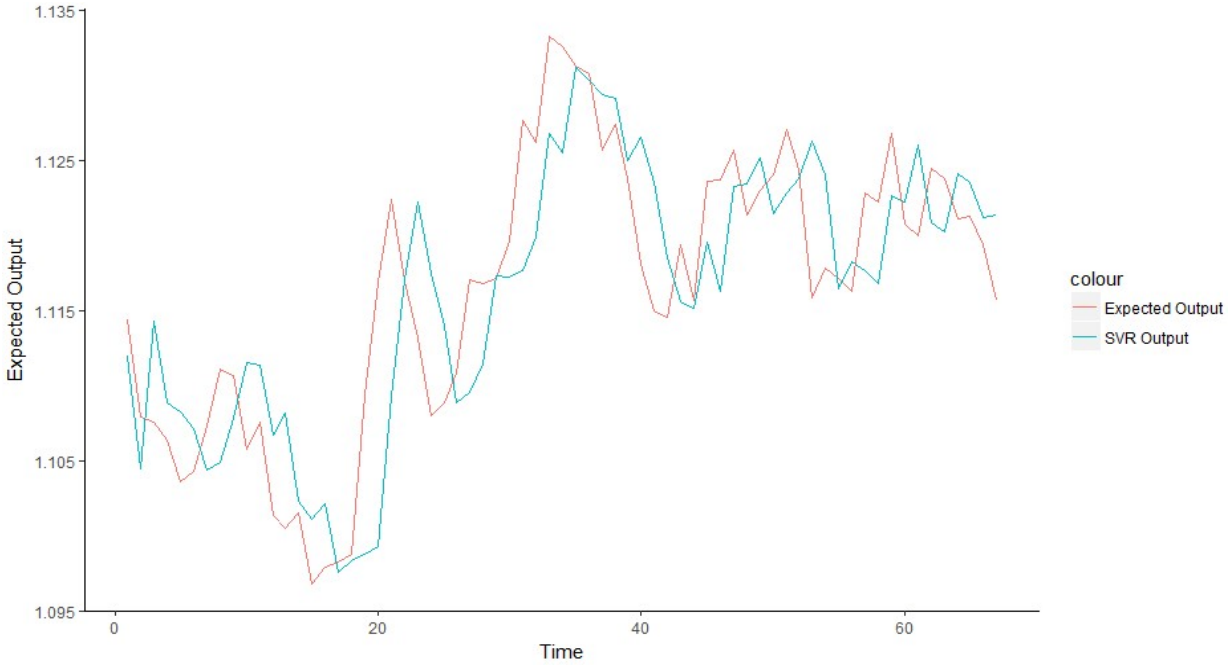
```
> svr_model_3_4_predict <- predict(svr_model_3_4, usd_eur_test_3[, 1:3])
> svr_model_3_4_predict
        321         322         323         324         325         326         327         328
0.5497811664 0.4754112964 0.5727349280 0.5188453187 0.5133995089 0.5019917401 0.4749945502 0.4799882669
        329         330         331         332         333         334         335         336
0.5084951643 0.5455655885 0.5438336807 0.4977612712 0.5122253699 0.4547677634 0.4428866114 0.4526333722
        337         338         339         340         341         342         343         344
0.4082048408 0.4154348277 0.4199409145 0.4248934475 0.5236143964 0.5993081534 0.6506535374 0.6040944893
        345         346         347         348         349         350         351         352
0.5688951256 0.5194136945 0.5255902528 0.5446218998 0.6019396968 0.6016666676 0.6051774932 0.6274194521
        353         354         355         356         357         358         359         360
0.6951474591 0.6827736834 0.7383756157 0.7305341329 0.7211499461 0.7179966918 0.6775834564 0.6929396104
        361         362         363         364         365         366         367         368
0.6628792612 0.6138982409 0.5855974237 0.5811808365 0.6239433606 0.5921616101 0.6603691753 0.6625816108
        369         370         371         372         373         374         375         376
0.6792609442 0.6426993319 0.6566177761 0.6659644320 0.6906737229 0.6679969912 0.5940649560 0.6111654143
        377         378         379         380         381         382         383         384
0.6054334493 0.5971914078 0.6546985237 0.6506902370 0.6882451047 0.6372865190 0.6306747834 0.6690807326
        385         386         387
0.6633662786 0.6403059512 0.6420177464
>
```

De-normalize the predicted output:

```
> pred_svr_3_4 <- as.data.frame(svr_model_3_4_predict)
> head(pred_svr_3_4)
    svr_model_3_4_predict
321          0.5497811664
322          0.4754112964
323          0.5727349280
324          0.5188453187
325          0.5133995089
326          0.5019917401
> pred_svr_denorm_3_4 <- as.data.frame(Map(denormalize, pred_svr_3_4, minvec, maxvec))
> pred_svr_denorm_3_4 <- pred_svr_denorm_3_4[, 1]
> head(pred_svr_denorm_3_4)
[1] 1.112012745 1.104449329 1.114347142 1.108866569 1.108312730 1.107152560
> length(pred_svr_denorm_3_4)
[1] 67
> series_svr_3_4 <- cbind(seq_along(test_denorm_out_3), test_denorm_out_3, pred_svr_denorm_3_4)
> head(series_svr_3_4)
      test_denorm_out_3 pred_svr_denorm_3_4
[1,] 1            1.1144         1.112012745
[2,] 2            1.1079         1.104449329
[3,] 3            1.1076         1.114347142
[4,] 4            1.1064         1.108866569
[5,] 5            1.1036         1.108312730
[6,] 6            1.1043         1.107152560
> colnames(series_svr_3_4) <- c("Time","Expected Output","SVR Output")
> series_svr_3_4 <- as.data.frame(series_svr_3_4)
> head(series_svr_3_4)
  Time Expected Output  SVR Output
1    1          1.1144 1.112012745
2    2          1.1079 1.104449329
3    3          1.1076 1.114347142
4    4          1.1064 1.108866569
5    5          1.1036 1.108312730
6    6          1.1043 1.107152560
>
```

We have finally de-normalized both the expected output and the predicted output.

```
> ggplot(series_svr_3_4, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



Mean Square Error:

```
> mse_svr_3_4 <- mse(series_svr_3_4$`Expected Output`, series_svr_3_4$`SVR Output`)
> mse_svr_3_4
[1] 0.00003119379192
>
```

Root Mean Square Error:

```
> rmse_svr_3_4 <- rsme(series_svr_3_4$`Expected Output`, series_svr_3_4$`SVR Output`)
> rmse_svr_3_4
[1] 0.005585140277
>
```

Mean Percentage Absolute Error:

```
> mape_svr_3_4 <- mape(series_svr_3_4$`Expected Output`, series_svr_3_4$`SVR Output`)
> mape_svr_3_4
[1] "0.400218%"
>
```

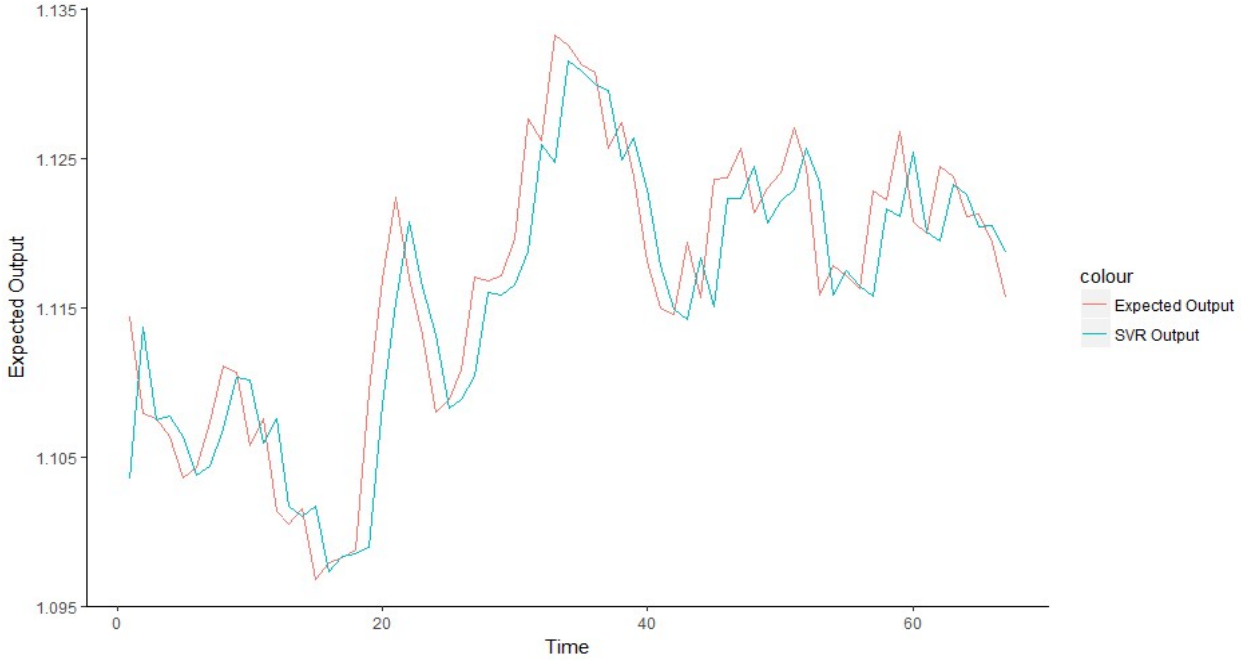
Collating and tabulating the errors to be able to figure out which model so far is the best:

```
> error_svr_2_1 <- c(mse_svr_2_1, rmse_svr_2_1, mape_svr_2_1)
> error_svr_2_2 <- c(mse_svr_2_2, rmse_svr_2_2, mape_svr_2_2)
> error_svr_2_3 <- c(mse_svr_2_3, rmse_svr_2_3, mape_svr_2_3)
> error_svr_2_4 <- c(mse_svr_2_4, rmse_svr_2_4, mape_svr_2_4)
> error_svr_3_1 <- c(mse_svr_3_1, rmse_svr_3_1, mape_svr_3_1)
> error_svr_3_2 <- c(mse_svr_3_2, rmse_svr_3_2, mape_svr_3_2)
> error_svr_3_3 <- c(mse_svr_3_3, rmse_svr_3_3, mape_svr_3_3)
> error_svr_3_4 <- c(mse_svr_3_4, rmse_svr_3_4, mape_svr_3_4)
> error_svr <- cbind(error_svr_2_1, error_svr_2_2, error_svr_2_3, error_svr_2_4,
+                    error_svr_3_1, error_svr_3_2, error_svr_3_3, error_svr_3_4)
> colnames(error_svr) <- c("model1_2", "model2_2", "model3_2", "model4_2",
+                          "model1_3", "model2_3", "model3_3", "model4_3")
> error_svr <-  as.data.frame(error_svr)
> error_svr
                 model1_2              model2_2              model3_2              model4_2              model1_3
1 0.0000177940196187942 0.0000177940196187942 0.0000196055149340963 0.0000179337906287679 0.0000179167706211149
2   0.00421829581926093   0.00421829581926093   0.00442781152874603   0.00423483064936108   0.00423282064598949
3             0.286406%             0.286406%             0.31499%             0.291581%             0.286345%
                 model2_3              model3_3              model4_3
1 0.0000177493781267606 0.0000207437746923295 0.000031193791919109
2   0.00421300108316632   0.00455453342202355  0.00558514027747818
3             0.283426%             0.322395%             0.400218%
>
```

So far, we can observe that model 2 of three input has the lowest error (MSE, RMSE, and MAPE). Therefore, we will choose as the best model. This improvement suggest that data behavior is possibly nonlinear and that parameter tuning works well for this

model. Perhaps, if we get more data, we might be able to improve the model better leveraging parameter tuning.
Let visualize our best model:

```
> ggplot(series_svr_3_2, aes(Time)) +
+   geom_line(aes(y = `Expected Output`, colour = "Expected Output")) +
+   geom_line(aes(y = `SVR Output`, colour = "SVR Output"))+
+   theme(panel.grid.major = element_blank(),
+         panel.grid.minor = element_blank(),
+         panel.background = element_blank(),
+         axis.line = element_line(colour = "black"))
>
```



Compare nn best with svr best:

```
> nn_best <- as.data.frame(error_nn$model1_3)
> nn_best
      error_nn$model1_3
1 0.0000161682224969046
2   0.00402097282966506
3                0.27865%
> svm_best <- as.data.frame(error_svr$model2_3)
> svm_best
     error_svr$model2_3
1 0.0000177493781267606
2   0.00421300108316632
3               0.283426%
> compare_svm_nn_best <- cbind(nn_best, svm_best)
> compare_svm_nn_best
      error_nn$model1_3    error_svr$model2_3
1 0.0000161682224969046 0.0000177493781267606
2   0.00402097282966506   0.00421300108316632
3                0.27865%              0.283426%
> colnames(compare_svm_nn_best) <- c("neural net Error", "Support Vector Regression Error")
> compare_svm_nn_best
       neural net Error Support Vector Regression Error
1 0.0000161682224969046           0.0000177493781267606
2   0.00402097282966506             0.00421300108316632
3                0.27865%                        0.283426%
>
```

It is evident that the best neural net model performed better than the best SVR model in all performance rating. This is deduced from the analysis so far.