7BUIS008W Data Mining & Machine Learning - Coursework 1

Andrew Keats

22 November 2017

Table of Contents

Question 1: White Wine clustering

Starting off

> You need to conduct the k-means clustering analysis of the white wine sheet. Find the ideal number of clusters (please justify your answer). Choose the best two possible numbers of clusters and perform the k-means algorithm for both candidates. Validate which clustering test is more accurate. For the winning test, get the mean of the each attribute of each group. Before conducting the k-means, please investigate if you need to add in your code any pre-processing task (justify your answer). Write a code in R Studio to address all the above issues. In your report, check the consistency of those produced clusters, with information obtained from column 12.

In the White Wine dataset provided, column 12 is labelled Quality; this is a qualitative value assigned by a human through the subjective means of tasting. Essentially, by try to cluster against all variables apart from Quality and then comparing against this variable, we are trying to look for some correlation between all the variables in combination and the subjective quality of wine.

Firstly we need to load the data…

```
#going to import the Excel spreadsheet WhiteWine dataset
wine.raw <- read_excel("../data/Whitewine.xlsx")
```

Here's a glance at the dataset

```
head(wine.raw)
```

```
## # A tibble: 6 x 12
## `fixed acidity` `volatile acidity` `citric acid` `residual
sugar`
```

```
## <dbl> <dbl> <dbl> <dbl>
## 1 7.0 0.27 0.36 20.7
## 2 6.3 0.30 0.34 1.6
## 3 8.1 0.28 0.40 6.9
## 4 7.2 0.23 0.32 8.5
## 5 7.2 0.23 0.32 8.5
## 6 8.1 0.28 0.40 6.9
## # ... with 8 more variables: chlorides <dbl>, `free sulfur
dioxide` <dbl>,
## # `total sulfur dioxide` <dbl>, density <dbl>, pH <dbl>,
## # sulphates <dbl>, alcohol <dbl>, quality <dbl>
```

```
str(wine.raw)
```

```
## 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 ...
```

We want to scale the data to allow all attributes to be compared more easily. First of all let’s split our data so we have two tables, one with all the attributes of wine and the other just with the humanly perceived quality.

```
wine.all_but_q <- wine.raw[1:11]
wine.q <- wine.raw$quality

#Wine properties
str(wine.all_but_q)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame': 4898 obs. of 11
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 ...
```

```
#Wine quality values
str(wine.q)
```

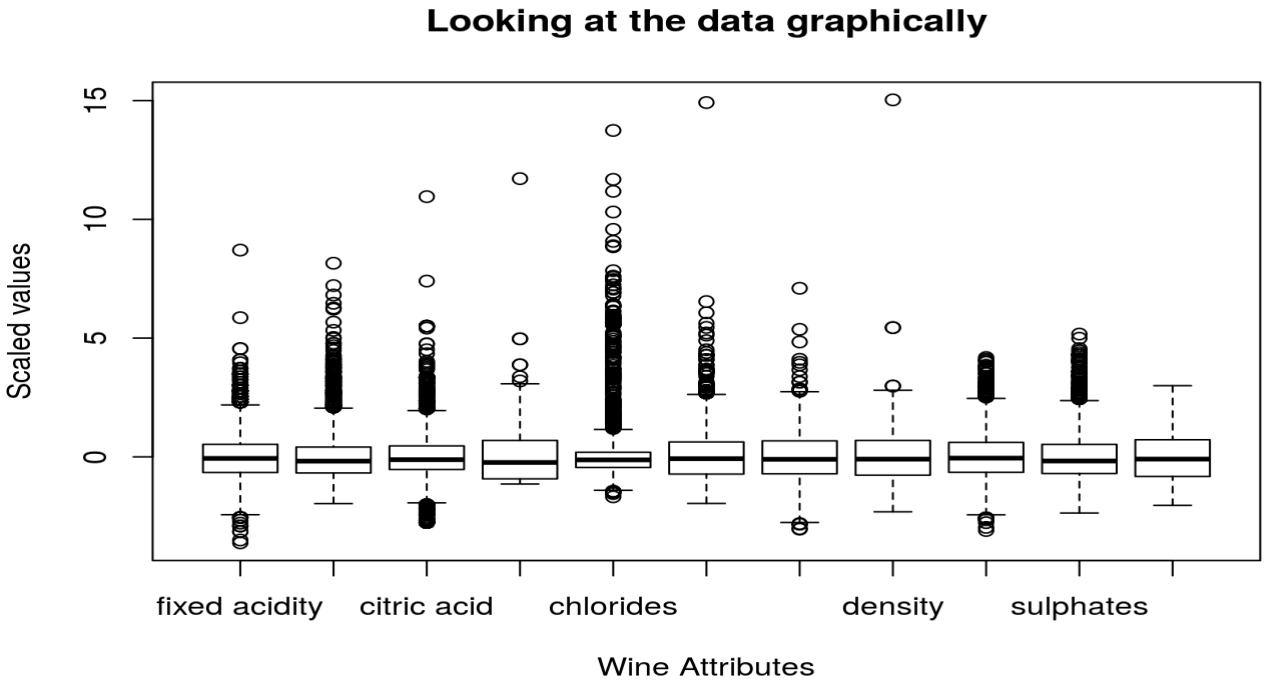
```
## num [1:4898] 6 6 6 6 6 6 6 6 6 6 ...
```

Now we scale the data

```
wine.scaled <- as.data.frame(scale(wine.all_but_q))

#Summary of scaled wine data
summary(wine.scaled)
```

```
## fixed acidity volatile acidity citric acid residual sugar
## Min. :-3.61998 Min. :-1.9668 Min. :-2.7615 Min. :-1.1418
## 1st Qu.:-0.65743 1st Qu.:-0.6770 1st Qu.:-0.5304 1st
Qu.:-0.9250
## Median :-0.06492 Median :-0.1810 Median :-0.1173 Median
:-0.2349
## Mean : 0.00000 Mean : 0.0000 Mean : 0.0000 Mean : 0.0000
## 3rd Qu.: 0.52758 3rd Qu.: 0.4143 3rd Qu.: 0.4612 3rd Qu.:
0.6917
## Max. : 8.70422 Max. : 8.1528 Max. :10.9553 Max. :11.7129
## chlorides free sulfur dioxide total sulfur dioxide
## Min. :-1.6831 Min. :-1.95848 Min. :-3.0439
## 1st Qu.:-0.4473 1st Qu.:-0.72370 1st Qu.:-0.7144
## Median :-0.1269 Median :-0.07691 Median :-0.1026
## Mean : 0.0000 Mean : 0.00000 Mean : 0.0000
## 3rd Qu.: 0.1935 3rd Qu.: 0.62867 3rd Qu.: 0.6739
## Max. :13.7417 Max. :14.91679 Max. : 7.0977
## density pH sulphates
## Min. :-2.31280 Min. :-3.10109 Min. :-2.3645
## 1st Qu.:-0.77063 1st Qu.:-0.65077 1st Qu.:-0.6996
## Median :-0.09608 Median :-0.05475 Median :-0.1739
## Mean : 0.00000 Mean : 0.00000 Mean : 0.0000
## 3rd Qu.: 0.69298 3rd Qu.: 0.60750 3rd Qu.: 0.5271
## Max. :15.02976 Max. : 4.18365 Max. : 5.1711
## alcohol
## Min. :-2.04309
## 1st Qu.:-0.82419
## Median :-0.09285
## Mean : 0.00000
## 3rd Qu.: 0.71974
## Max. : 2.99502
```

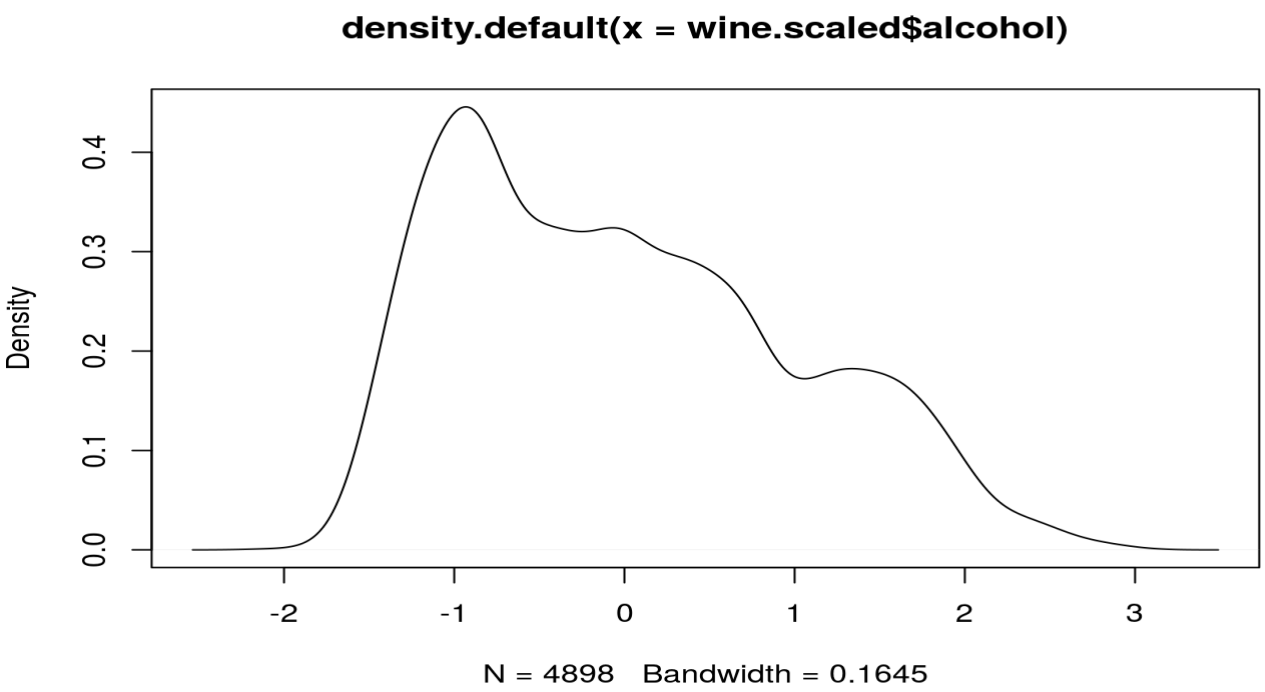
```
boxplot(wine.scaled, main="Looking at the data graphically",
xlab="Wine Attributes", ylab="Scaled values")
```



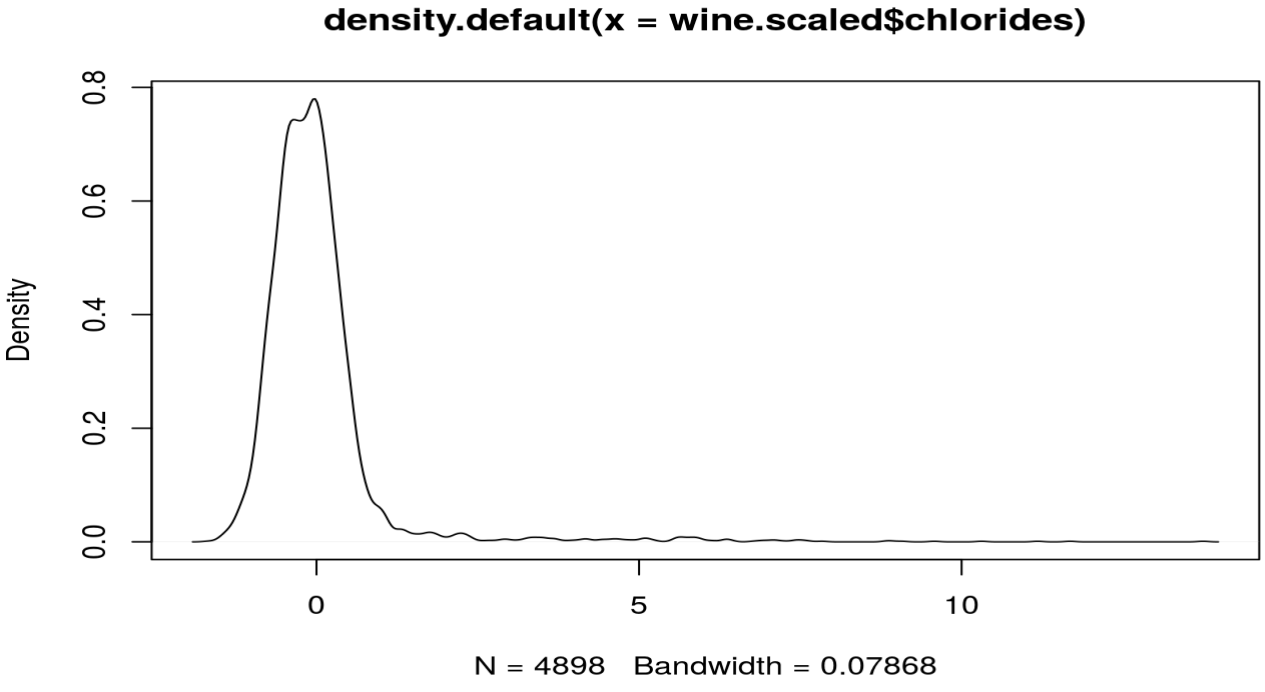
We can see from these box-plots that some attributes seem to have some clear outliers that would suggest erroneous data and not just natural extremes (**Prabhakaran, 2016**). As such, we can decide that it's worth cleansing the data a little by removing these outliers from the dataset. For example, Alcohol on the most right column seems to have very clear boundaries as we'd expect from wine; when that is compared with some other attributes, they seem to tell a different story: Chlorides seems to have a lot of values that are in the upper quartile, and a large distance between min and max values but when you look it you can see there's a gradient that suggests a normal distribution; in contrast to this, the columns Residual Sugar, Free Sulfur Dioxide and Density all seem to not only have relatively large min and max distances but there seem to be uppermost values that with nearest neighbour values that are a relatively large distance away.

Below are density line graphs to demonstrate the difference between attributes that don't seem to have outliers compared to those that do.

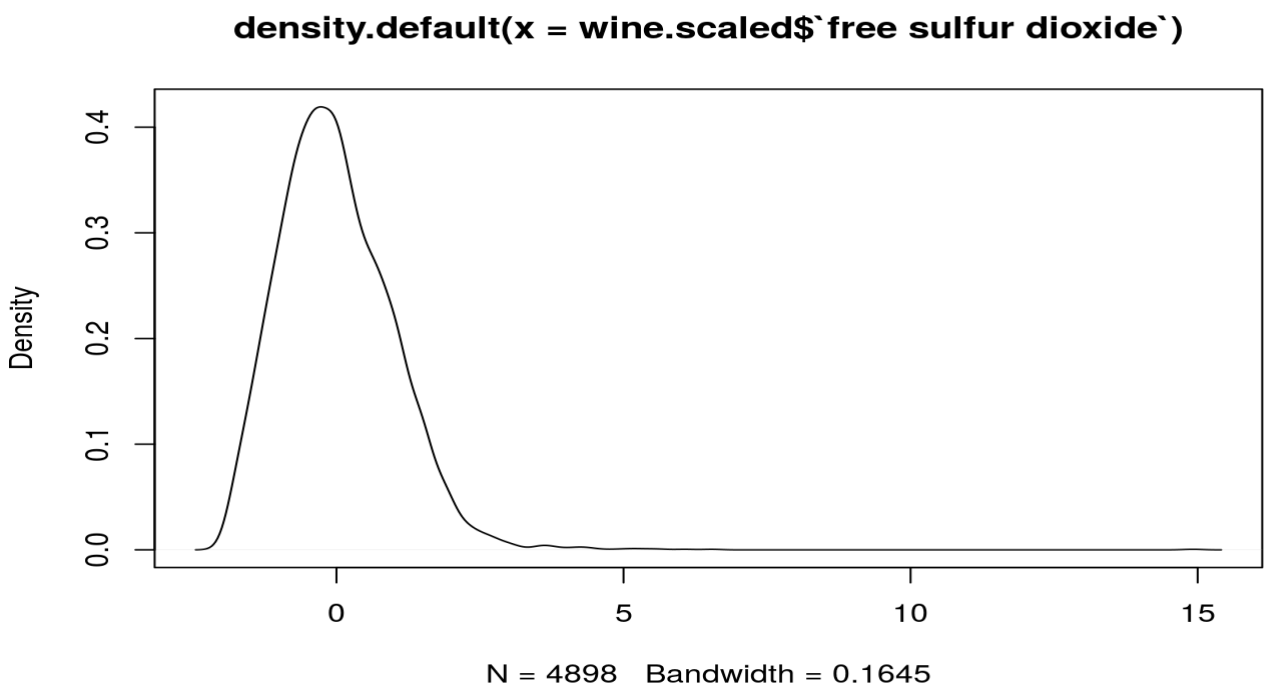
```
plot(density(wine.scaled$`alcohol`))
```



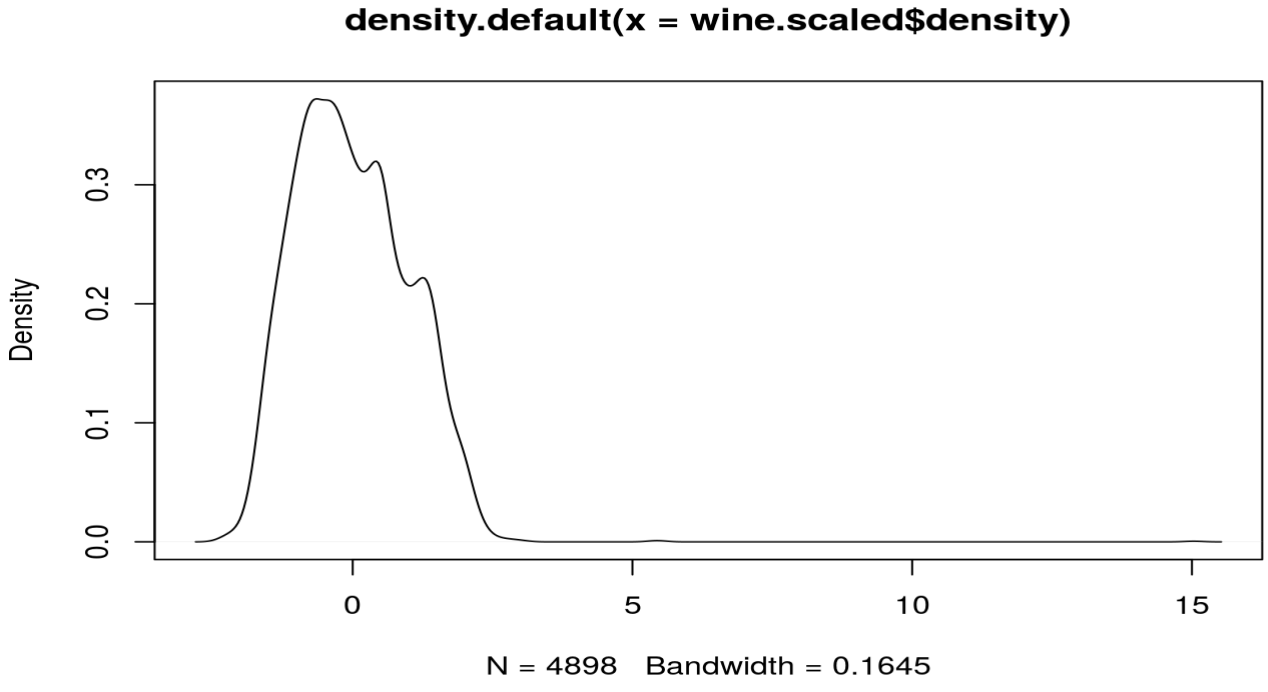

```
plot(density(wine.scaled$`chlorides`))
```




```
plot(density(wine.scaled$`free sulfur dioxide`))
```




```
plot(density(wine.scaled$`density`))
```



In order to work out which attributes should be considered to have valid outliers, I've gone with a heuristic approach, choosing to look at the distance between the uppermost outliers for each attribute and it's nearest neighbour.

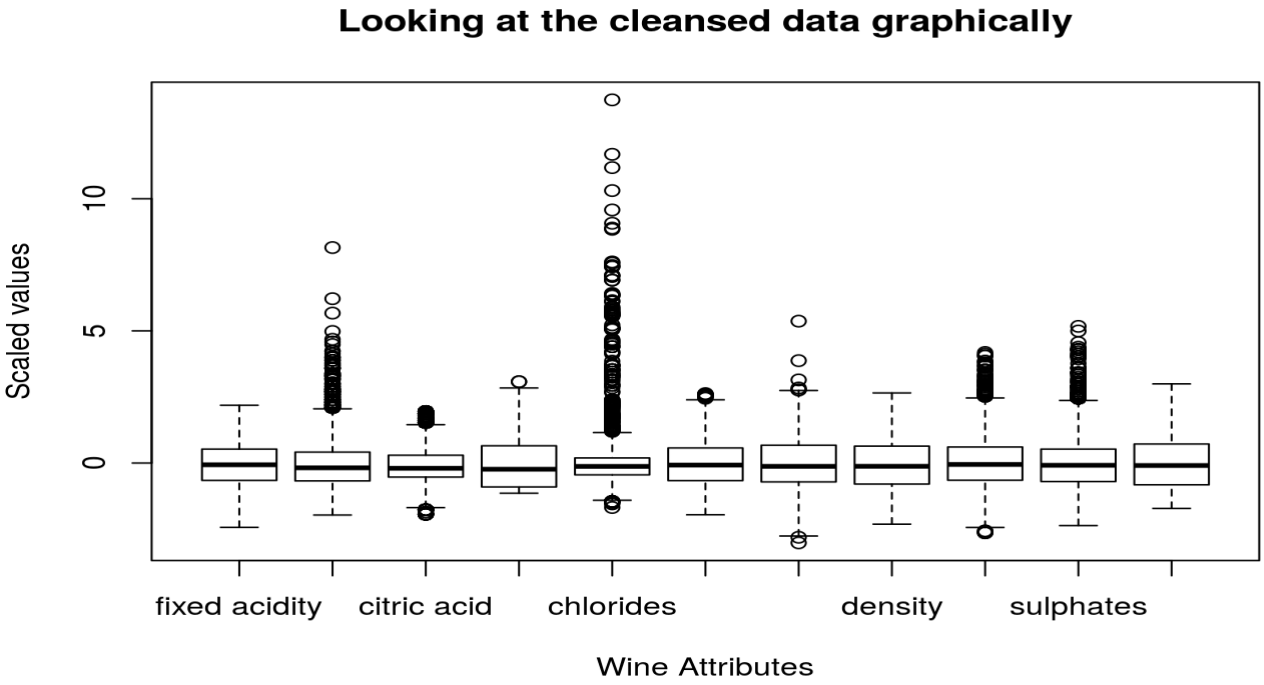
```
#Create a list to populate with our tail neighbour distances
tail_deltas <- c()

for (attrib in wine.scaled) {
#get the last two values
data_tails <- tail(sort(attrib),2)
#push the delta on to our list
tail_deltas <- c(tail_deltas, diff(data_tails))
}

#grab out attribute keys to include in our new table/frame
attributes <- names(wine.scaled)

#make a new dataframe from
dataframe <- data.frame(attributes = attributes,
tail_neighbour_d=tail_deltas)

#get the order for the nearest neighbour starting with the
greatest distance and descending
neighbour_order <- order(dataframe$tail_neighbour_d,
decreasing=TRUE)

#now apply the order to the frame
sorted_attributes_by_neighbour_d <- dataframe[ neighbour_order,
]
sorted_attributes_by_neighbour_d



## attributes tail_neighbour_d
## 8 density 9.5890647
## 6 free sulfur dioxide 8.3788351
## 4 residual sugar 6.7428254
## 3 citric acid 3.5531375
## 1 fixed acidity 2.8440459
## 5 chlorides 2.0596881
## 7 total sulfur dioxide 1.7294905
## 2 volatile acidity 0.9425113
## 10 sulphates 0.1752452
## 11 alcohol 0.1218897
## 9 pH 0.0662249
```

Given the findings, I think we can just consider the top five attributes in the above list as ones to cleanse for outliers. A lot of sources online warn against arbitrarily getting rid of outliers because it might be the case that valid information is being lost when what you really want to account for is bad data.

To clarify, the attributes to be processed are: - density
- free sulfur dioxide
- residual sugar
- citric acid
- fixed acidity

Boxplot has an outlier property that we can use to collect values that we might want to remove, so this is the one option we will look at for cleansing data.

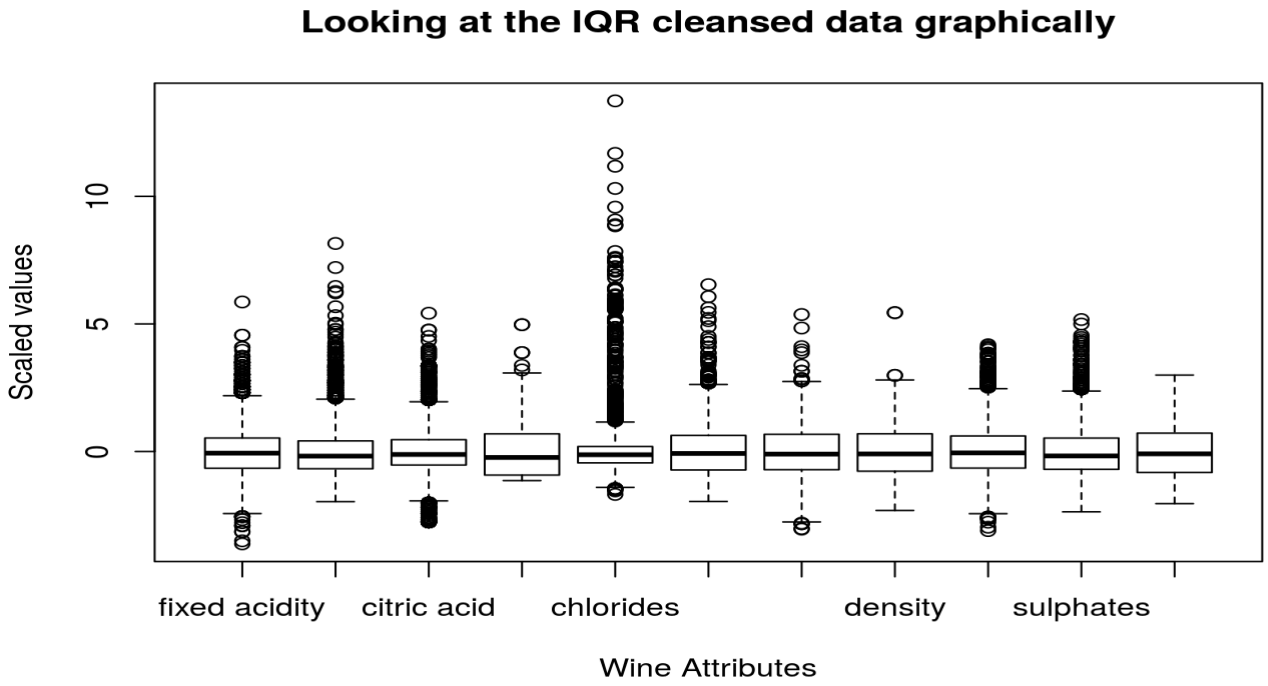
```
wine.scaled_cleansed_bp <- wine.scaled[ !(wine.scaled$density
%in% boxplot(wine.scaled$density, plot=FALSE)$out), ]
wine.scaled_cleansed_bp <- wine.scaled_cleansed_bp[ !
(wine.scaled_cleansed_bp$`free sulfur dioxide` %in%
boxplot(wine.scaled$`free sulfur dioxide`, plot=FALSE)$out), ]
wine.scaled_cleansed_bp <- wine.scaled_cleansed_bp[ !
(wine.scaled_cleansed_bp$`residual sugar` %in%
boxplot(wine.scaled_cleansed_bp$`residual sugar`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp <- wine.scaled_cleansed_bp[ !
(wine.scaled_cleansed_bp$`citric acid` %in%
boxplot(wine.scaled_cleansed_bp$`citric acid`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp <- wine.scaled_cleansed_bp[ !
(wine.scaled_cleansed_bp$`fixed acidity` %in%
boxplot(wine.scaled_cleansed_bp$`fixed acidity`,
plot=FALSE)$out), ]

boxplot(wine.scaled_cleansed_bp, main="Looking at the cleansed
data graphically", xlab="Wine Attributes", ylab="Scaled
values")
```



While this new set of data is now has no values beyond the outermost quartile ranges, this is arguably too harsh a treatment. An alternative option is to arbitrarily work with the interquartile ranges; what have done is to tweak the multiplier of the interquartile range until it successfully meant that only the most extreme outliers were discard. In the end a value 5 times that of the IQR worked well to pick off only values at the very tips of the tails.

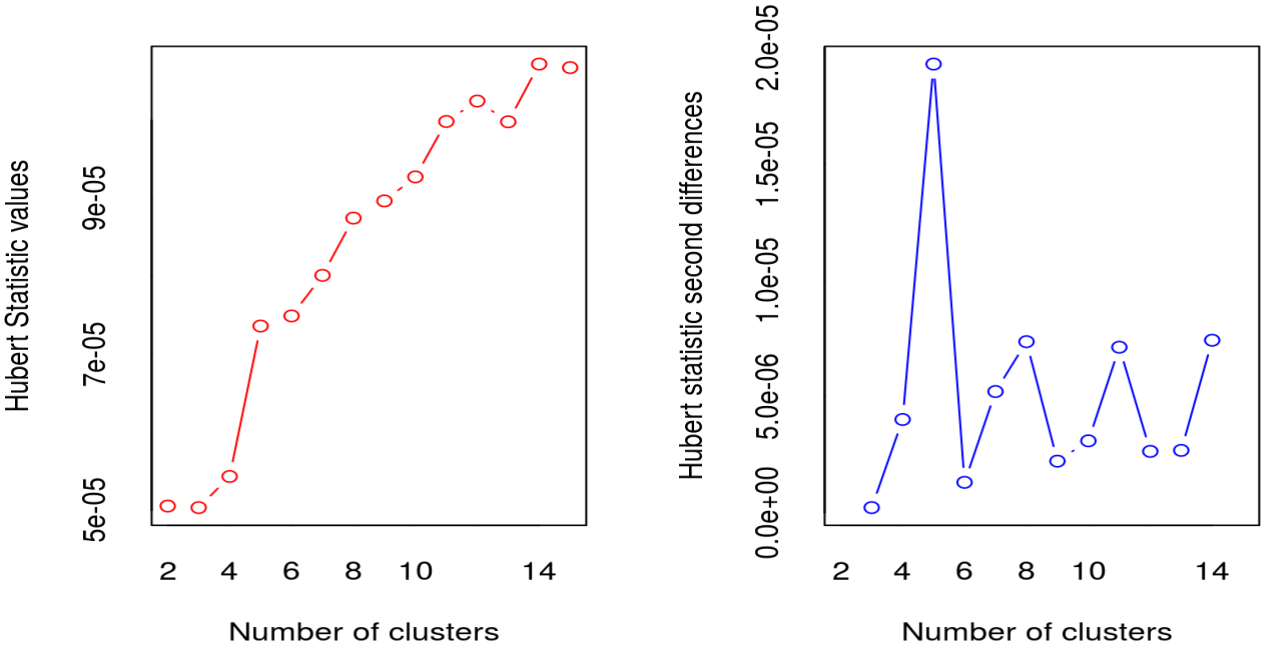
```
#Get the top 5 variables with the highest outlier distance
worst_outliers <-
head(sorted_attributes_by_neighbour_d$attributes, n=5)

wine.scaled_cleansed_iqr <- wine.scaled

# Create a variable to store the row id's to be removed
iqr_outliers <- c()
quartile_multiplier = 5

# Loop through the list of columns you specified
for(i in worst_outliers){

# Get the Min/Max values
max <- quantile(wine.scaled_cleansed_iqr[,i],0.75, na.rm=FALSE)
+ (IQR(wine.scaled_cleansed_iqr[,i], na.rm=FALSE) *
quartile_multiplier )
min <- quantile(wine.scaled_cleansed_iqr[,i],0.25, na.rm=FALSE)
- (IQR(wine.scaled_cleansed_iqr[,i], na.rm=FALSE) *
```

```
quartile_multiplier )

# Get the id's using which
idx <- which(wine.scaled_cleansed_iqr[,i] < min |
wine.scaled_cleansed_iqr[,i] > max)

# Output the number of outliers in each variable
#print(paste(i, length(idx), sep=' - removing: '))

# Append the outliers list
iqr_outliers <- c(iqr_outliers, idx)
}

# Sorting outliers
iqr_outliers <- sort(iqr_outliers)

# Remove the outliers
wine.scaled_cleansed_iqr <- wine.scaled_cleansed_iqr[-
iqr_outliers,]

boxplot(wine.scaled_cleansed_iqr, main="Looking at the IQR
cleansed data graphically", xlab="Wine Attributes",
ylab="Scaled values")
```

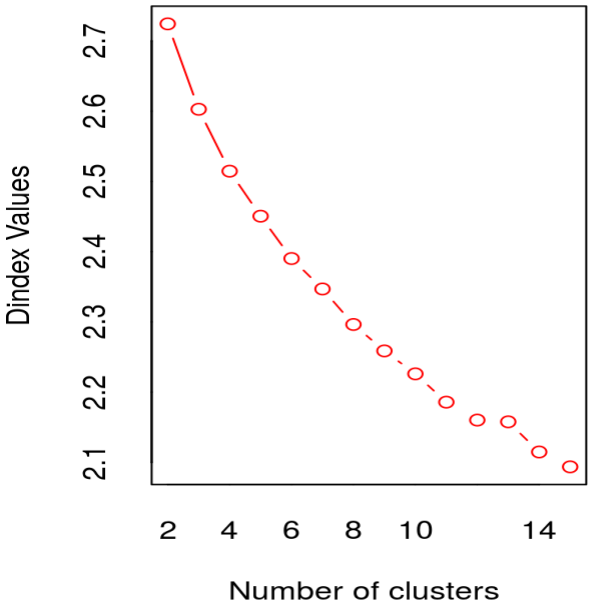


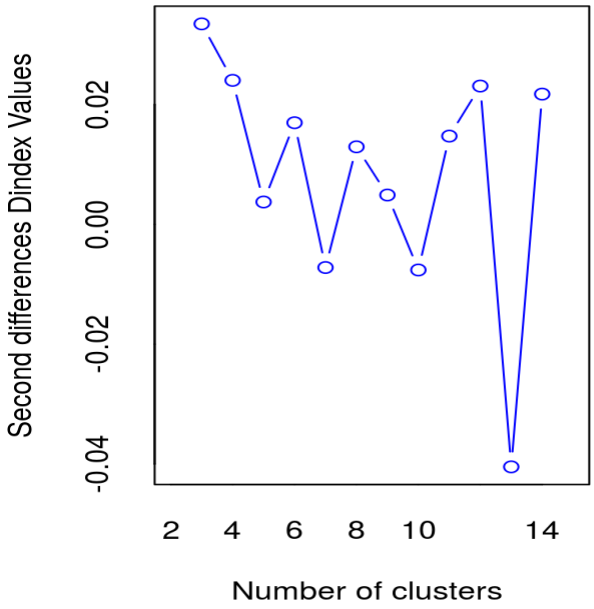
Now that the data looks a lot cleaner, it's time to start working with the data to try and find the best clustering. To begin with, nbclust will be used to see if that produces anything useful (**Using k-means to cluster wine dataset, 2015**).

```
number_of_clusters <- NbClust(wine.scaled_cleansed_iqr,
min.nc=2, max.nc=15,
method="kmeans")


## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```




```
## *** : 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
## * 2 proposed 4 as the best number of clusters
## * 3 proposed 5 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 1 proposed 8 as the best number of clusters
## * 1 proposed 13 as the best number of clusters
## * 3 proposed 14 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
##
##
```

```
##
*****************************************************************
```

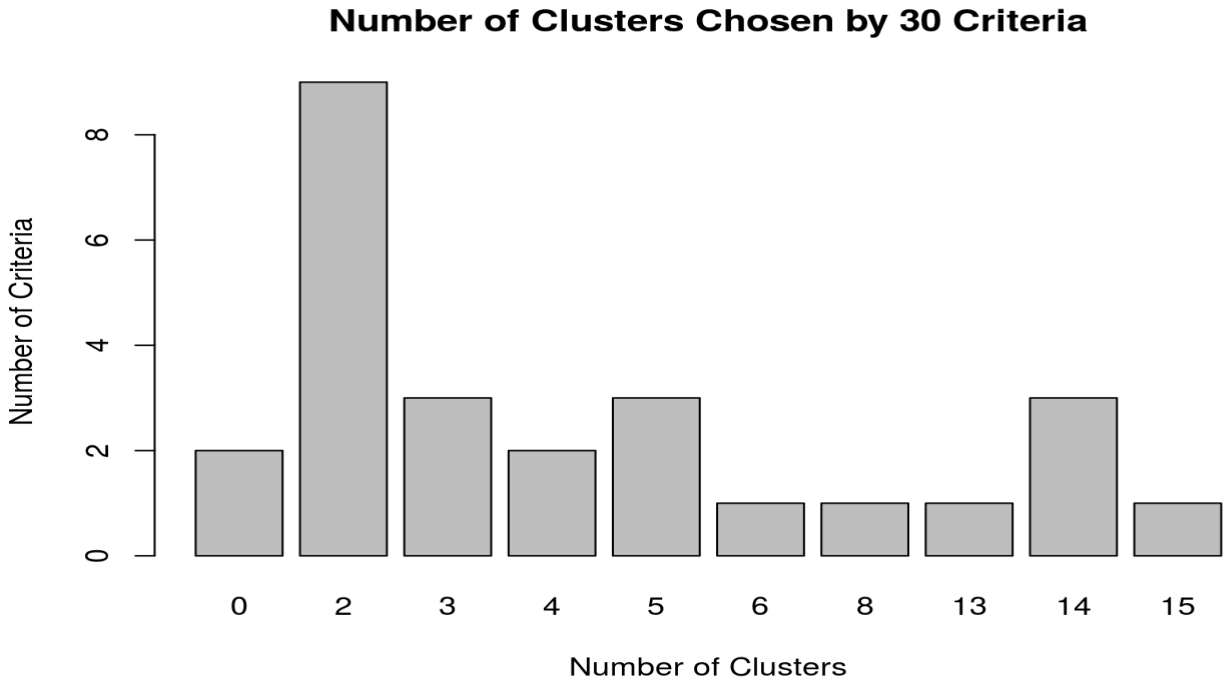
The following table displays the results recommending potential values for k

```
table(number_of_clusters$Best.n[1,])
```

```
##
## 0 2 3 4 5 6 8 13 14 15
## 2 9 3 2 3 1 1 1 3 1
```

The bar chart more easily conveys this.

```
barplot(table(number_of_clusters$Best.n[1,]),
xlab="Number of Clusters",
ylab="Number of Criteria",
main="Number of Clusters Chosen by 30 Criteria")
```



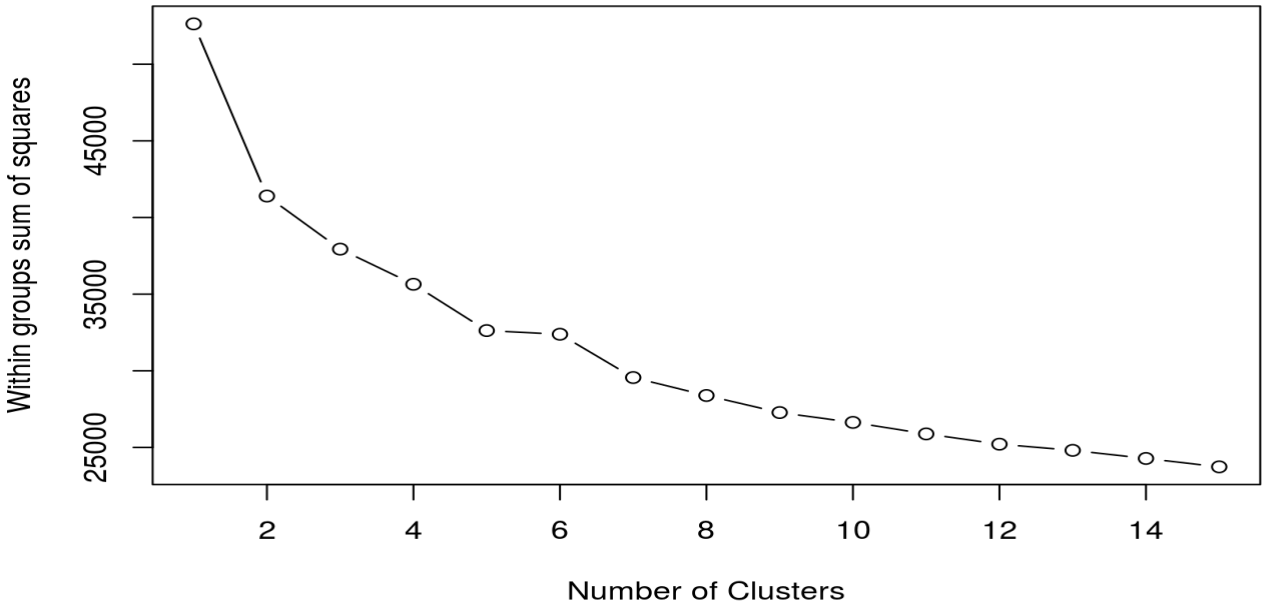
From the bar graph above we can see that there seems to be an clear leader in terms of suggested number of clusters, being k = 2. There are however other values that should be explored to see how they compare: 3, 5 and 14. To confirm that the accuracy of this

result in terms of the best contender for, we can plot the sum of square errors and looks for a pronounced bend in the graph. Where the most pronounced bend is, this is a contender for the value for k (**Kulma, 2017**).

```
sse_list <- 0
for (i in 1:15){
sse_list[i] <- sum(kmeans(wine.scaled_cleansed_iqr,
centers=i)$withinss)
}
```

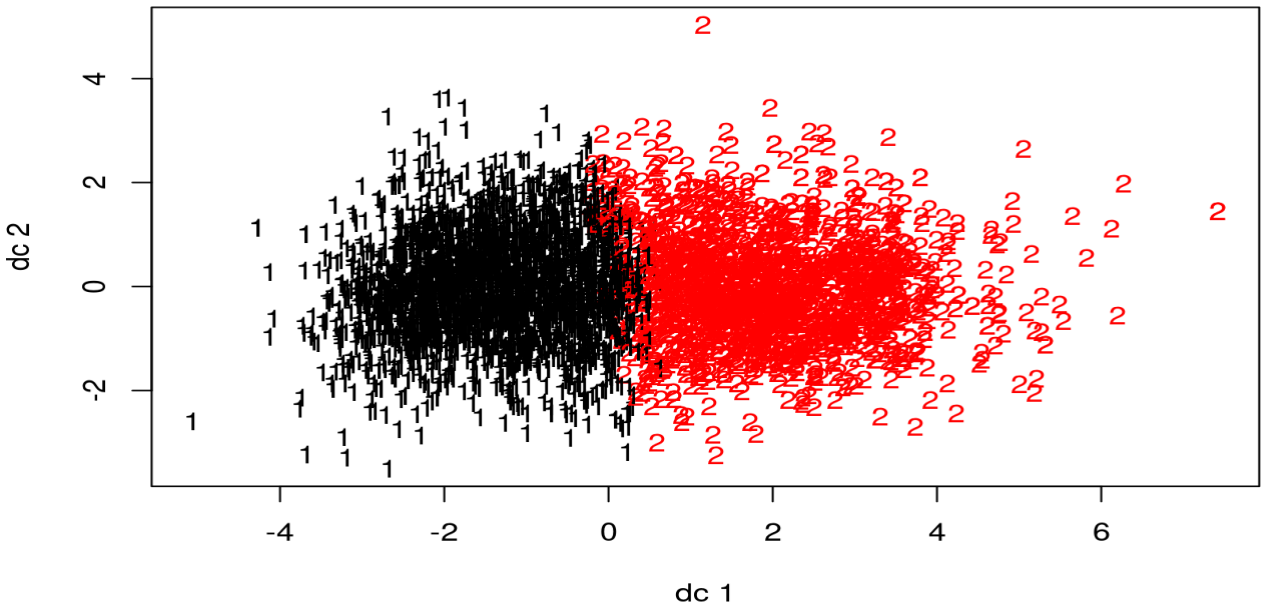
```
## Warning: did not converge in 10 iterations
```

```
plot(1:15,
sse_list,
type="b",
xlab="Number of Clusters",
ylab="Within groups sum of squares")
```

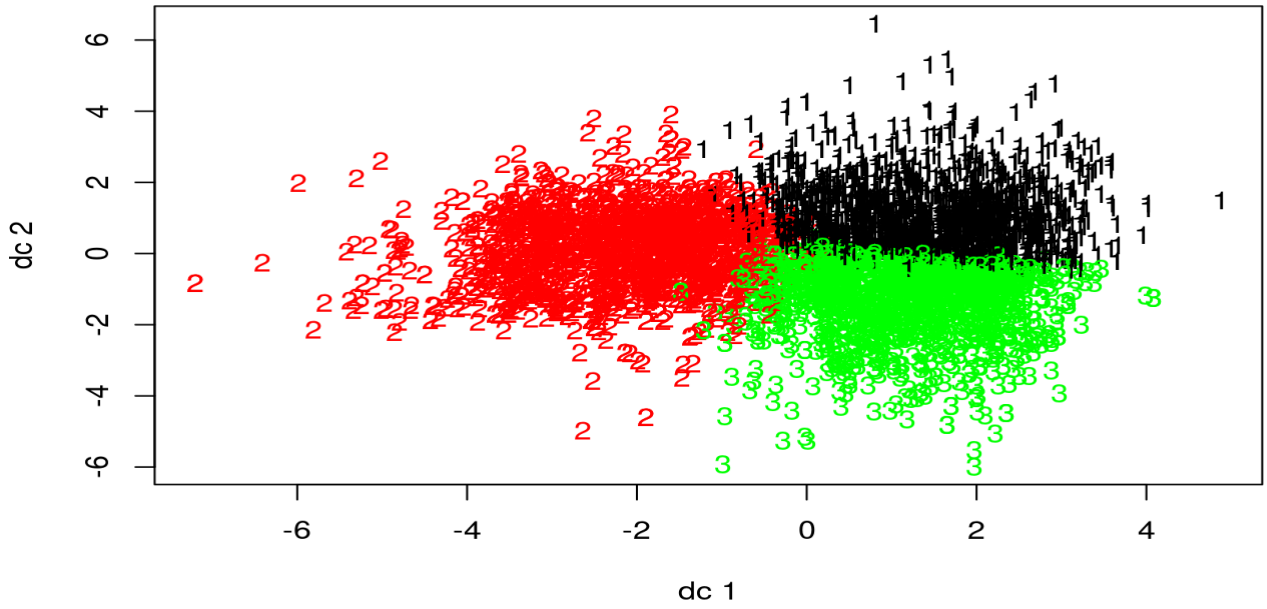


The histogram for the Sum of Square Errors partially backs up the results of nbclust seeing as there is ‘elbow’ (**Using k-means to cluster wine dataset, 2015**) on the line at 2 on the Number of Clusters. Having said that, the kink between 5 and 7 suggest that this range should also be tested for k.

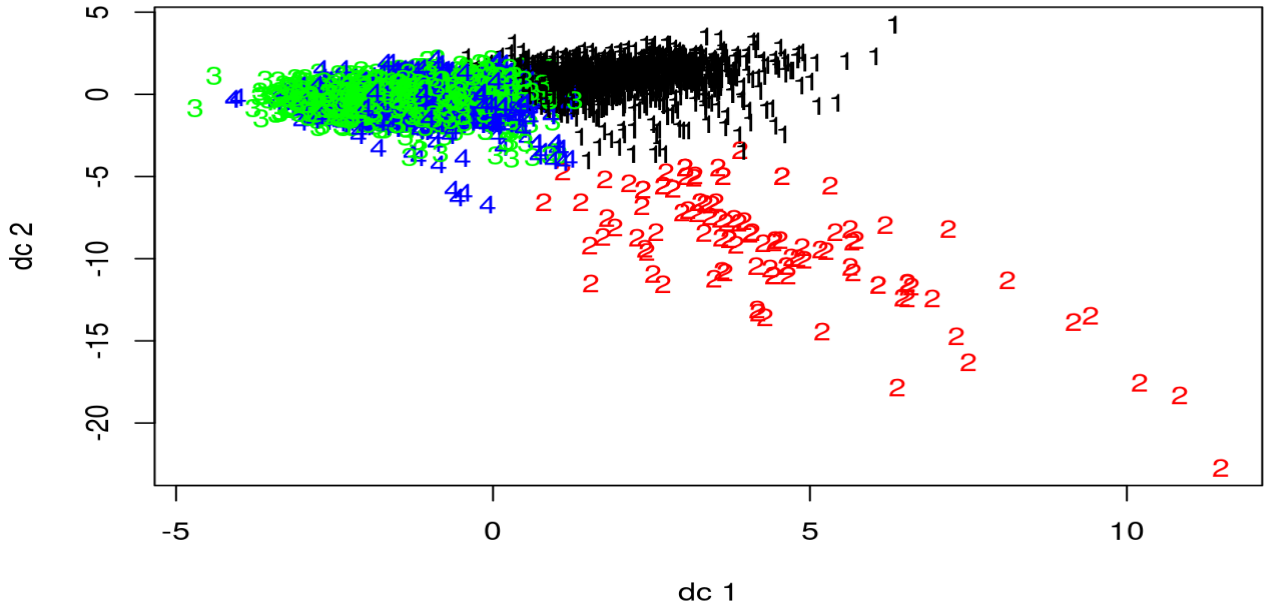
```
#If we're going to run tests on the k-means against the data we
need to remove the outliers from our quality column too
wine.q_cleansed <- wine.q[-iqr_outliers]

fit.km2 <- kmeans(wine.scaled_cleansed_iqr, 2)
fit.km3 <- kmeans(wine.scaled_cleansed_iqr, 3)
fit.km4 <- kmeans(wine.scaled_cleansed_iqr, 4)
fit.km5 <- kmeans(wine.scaled_cleansed_iqr, 5)
fit.km6 <- kmeans(wine.scaled_cleansed_iqr, 6)
fit.km7 <- kmeans(wine.scaled_cleansed_iqr, 7)
fit.km11 <- kmeans(wine.scaled_cleansed_iqr, 11)
fit.km14 <- kmeans(wine.scaled_cleansed_iqr, 14)

plotcluster(wine.scaled_cleansed_iqr, fit.km2$cluster)
```



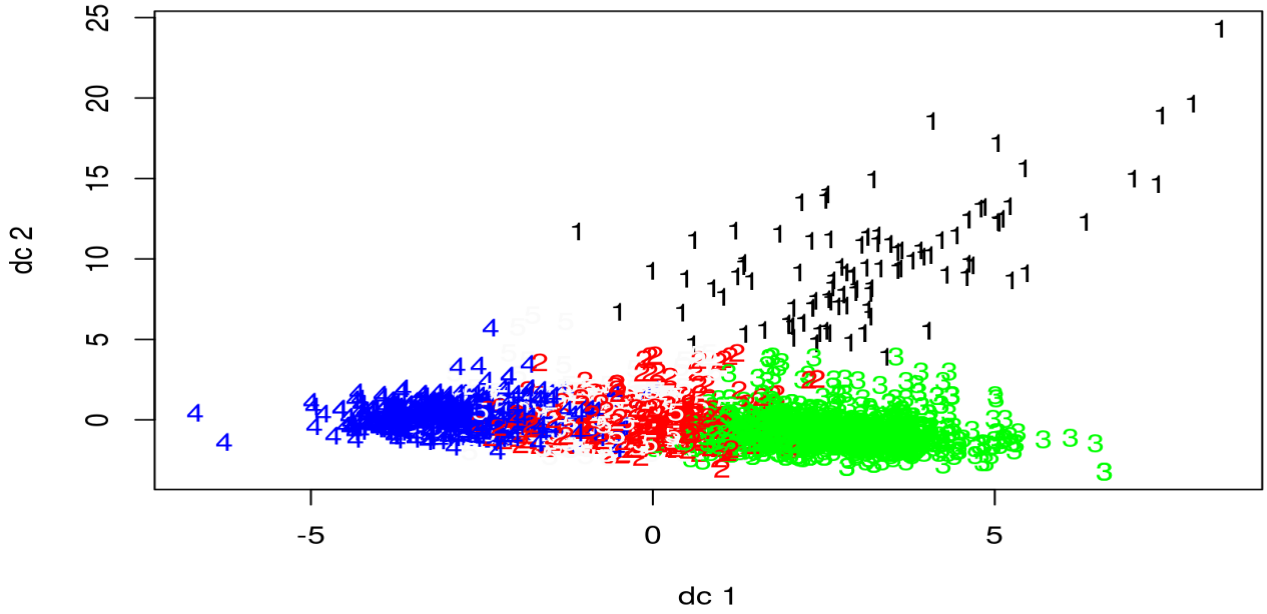

```
plotcluster(wine.scaled_cleansed_iqr, fit.km3$cluster)
```



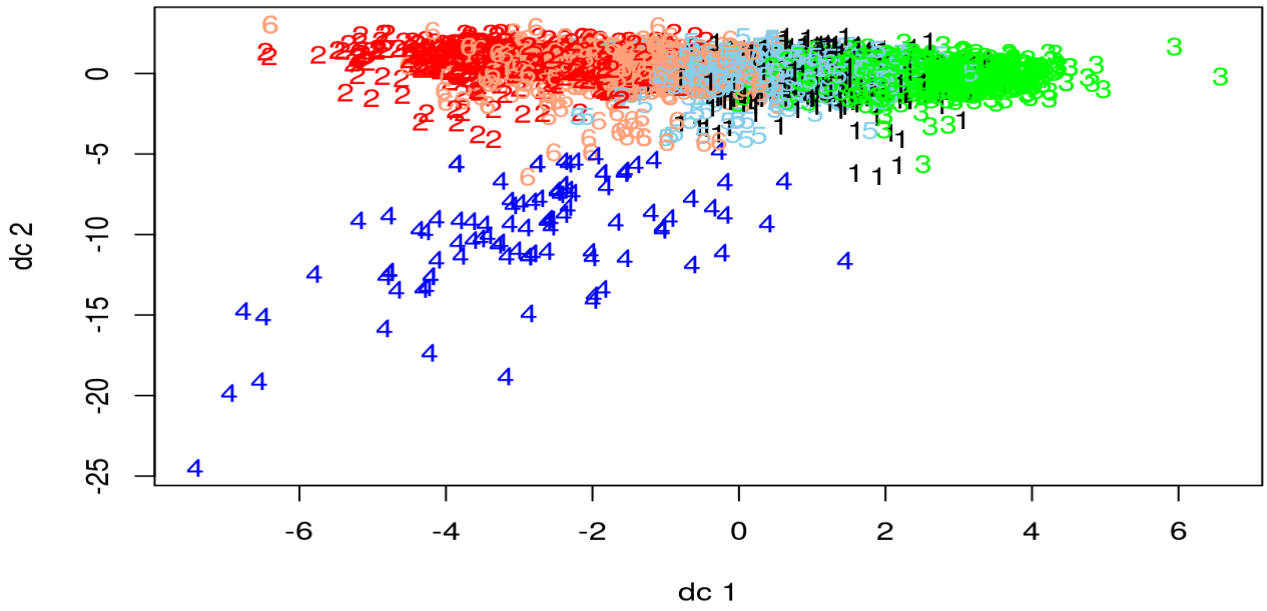

```
plotcluster(wine.scaled_cleansed_iqr, fit.km4$cluster)
```



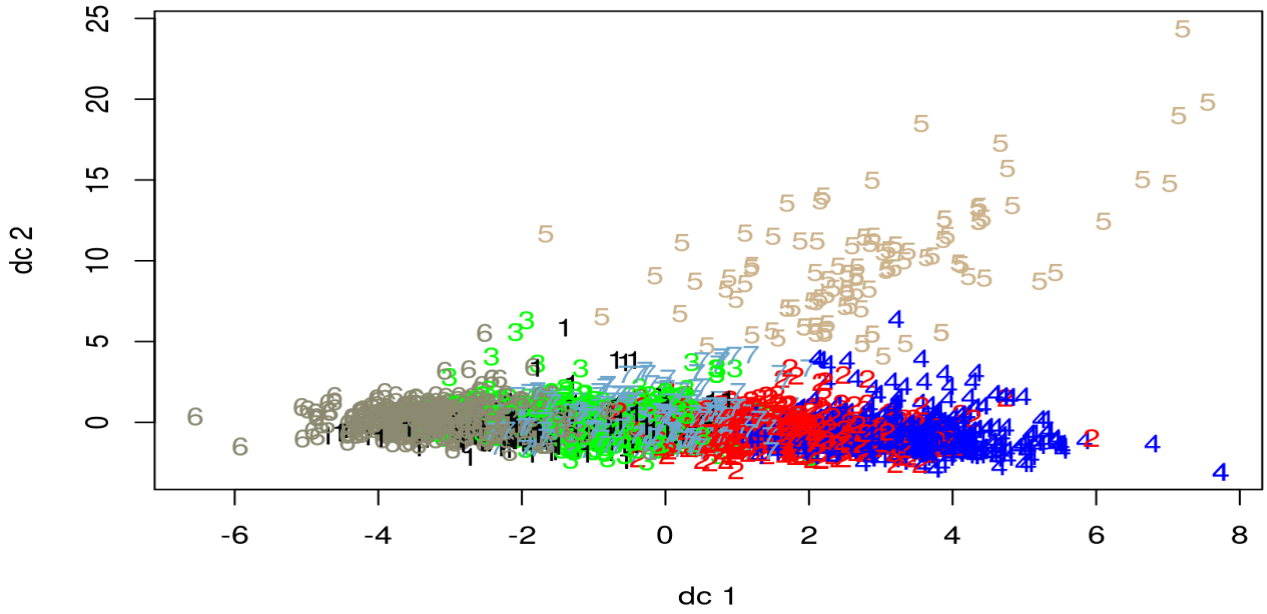

```
plotcluster(wine.scaled_cleansed_iqr, fit.km5$cluster)
```



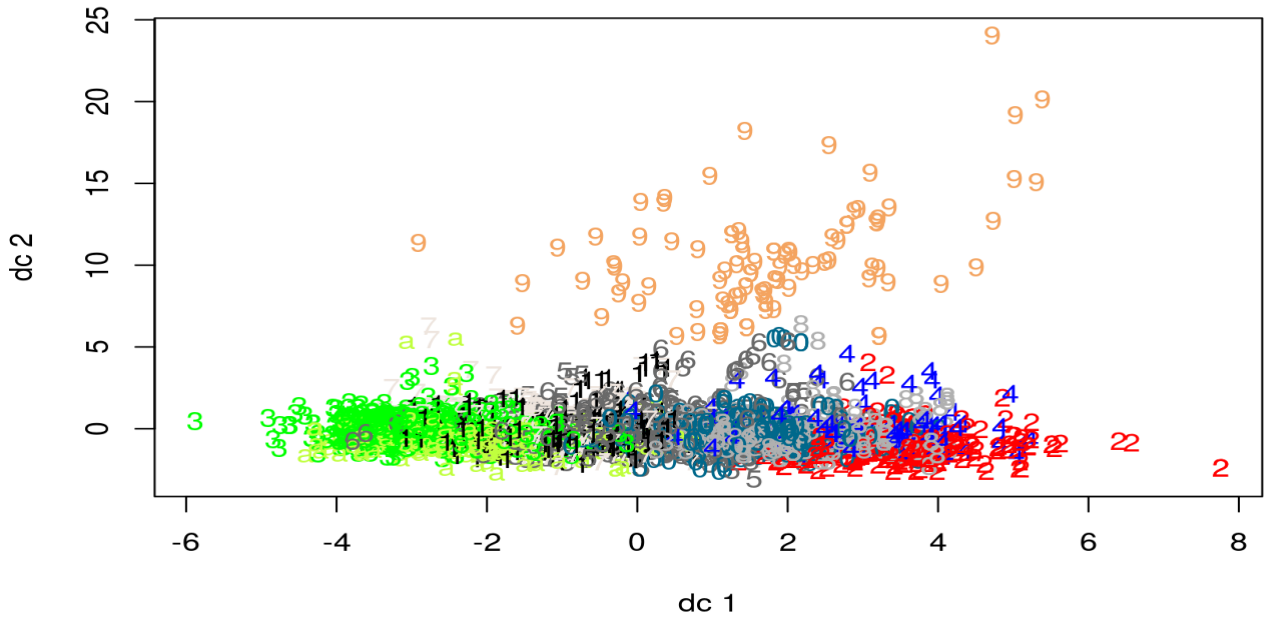

```
plotcluster(wine.scaled_cleansed_iqr, fit.km6$cluster)
```



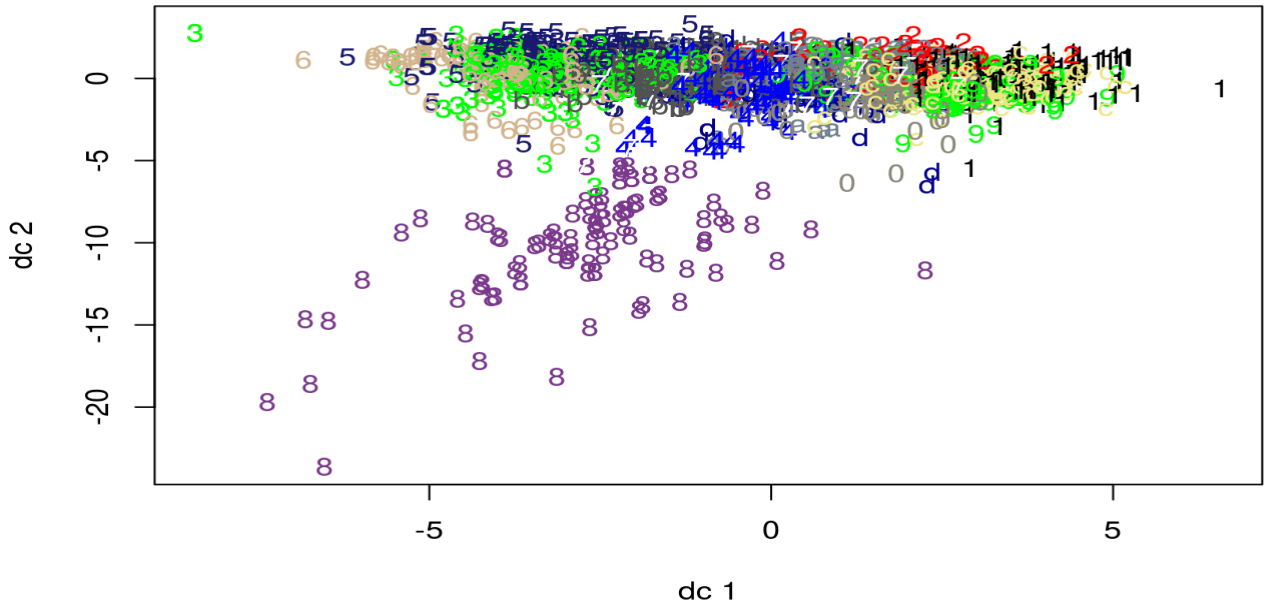

```
plotcluster(wine.scaled_cleansed_iqr, fit.km7$cluster)
```




```
plotcluster(wine.scaled_cleansed_iqr, fit.km11$cluster)
```




```
plotcluster(wine.scaled_cleansed_iqr, fit.km14$cluster)
```



Mapping/Fitting the clusters to the data

Now that we have experimented with various different values for k, when applying k-means clustering to the wine data, it's now time to see if it can be fit to the data that delivers anything obviously meaningful with regards to wine quality. While the strongest cluster option appears to be 2, I thought it was worth looking at how to map the data against quality by looking at the quality values as though they were factors; so, as you can see in the following table, there are only 7 unique values out of 10 possible scores for quality, meaning that trying to fit the data is actually easiest against 7 clusters, one per quality value.

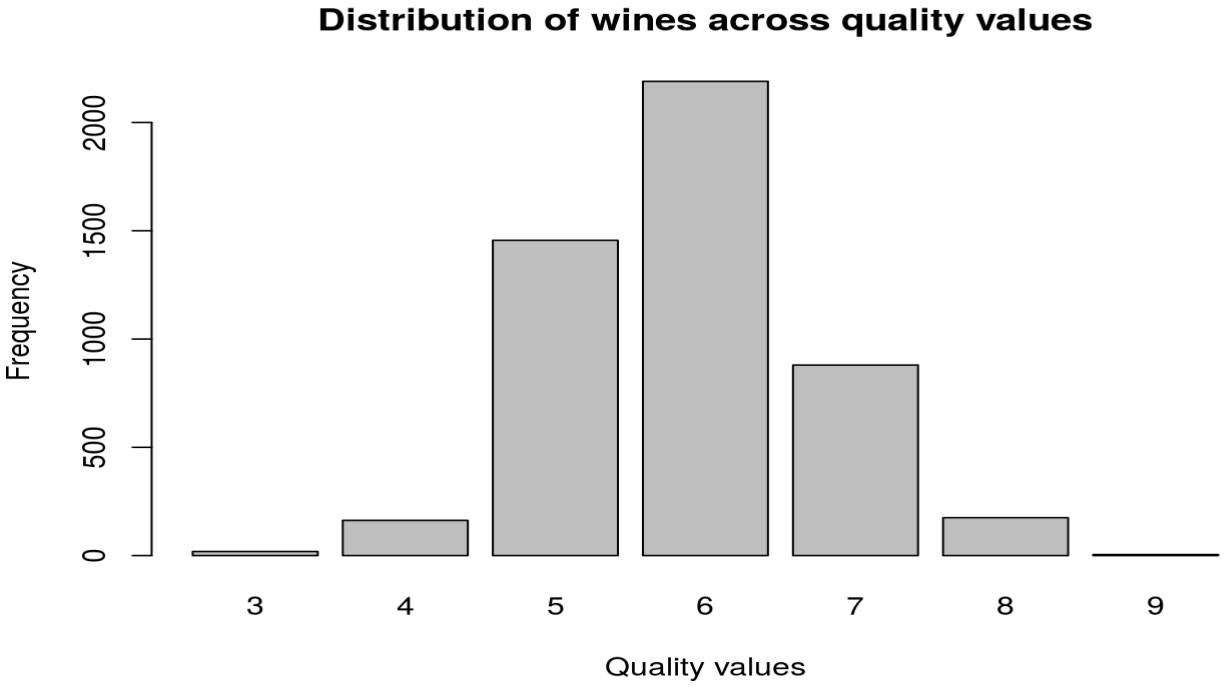
To compare the clusters to the quality scores we use a Confusion matrix to see where the values lie within those 2 sets of data. Further to that, to evaluate the confusion matrix mathematically, we will apply the Rand Index method describes thusly:

> The Rand index computes how similar the clusters (returned by the clustering algorithm) are to the benchmark classifications. One can also view the Rand index as a measure of the percentage of correct decisions made by the algorithm. It can be computed using the following formula RI={} (**Rand, 1971, p846–850**)

```
wine.q_table <- table(wine.q_cleansed)
wine.q_table
```

```
## wine.q_cleansed
## 3 4 5 6 7 8 9
## 19 163 1456 2190 880 175 5
```

```
barplot(wine.q_table,
xlab="Quality values",
ylab="Frequency",
main="Distribution of wines across quality values")
```




```
confuseTable.km7 <- table(wine.q_cleansed, fit.km7$cluster)

names(dimnames(confuseTable.km7)) <- list("Quality",
"Clusters")

confuseTable.km7
```

```
## Clusters
## Quality 1 2 3 4 5 6 7
## 3 0 6 6 2 1 2 2
```

```
## 4 10 18 48 18 4 19 46
## 5 71 409 278 327 49 51 271
## 6 267 366 396 347 47 327 440
## 7 163 34 142 102 2 324 113
## 8 31 6 22 18 2 76 20
## 9 0 0 1 0 0 4 0
```

```
randIndex(confuseTable.km7)
```

```
## ARI
## 0.03058022
```

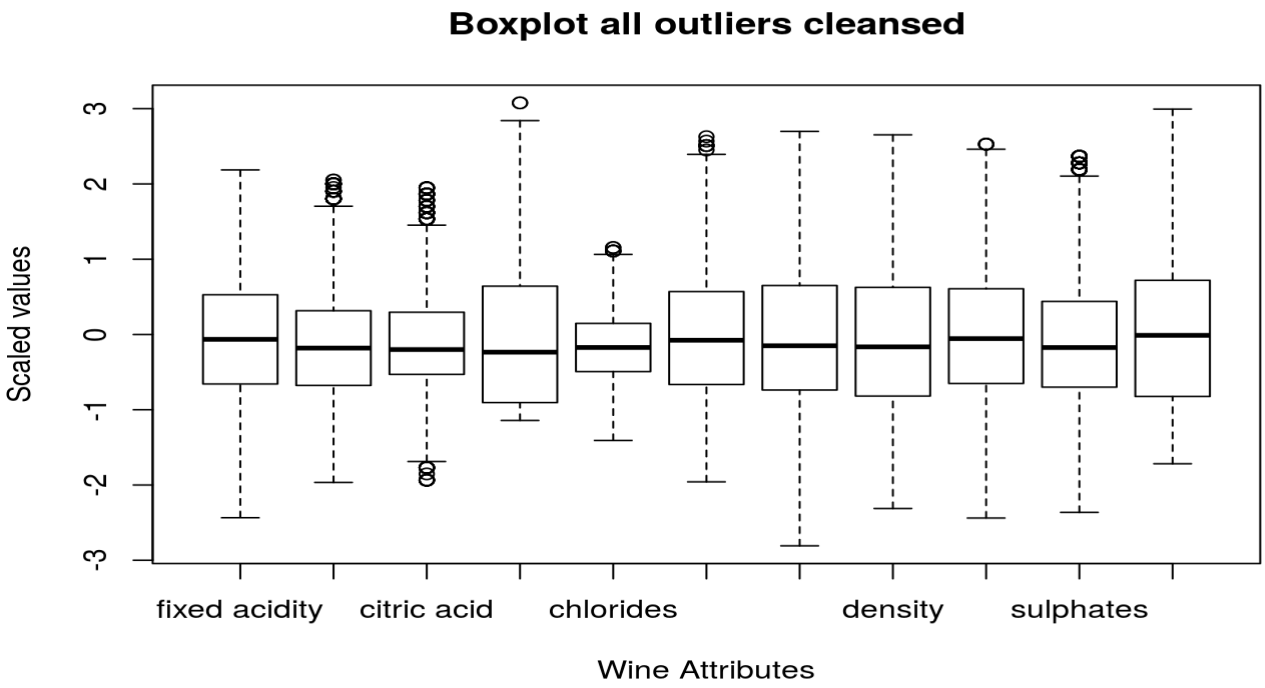
Poor results

Given this low value of 0.03410079, is so far from the ideal, and as can be seen from the matrix, there seems to be a spread across all clusters, we can surmise that either the White Wine dataset was not cleansed thoroughly enough or that k-means clustering i=simply isn't an effective way of determining quality.

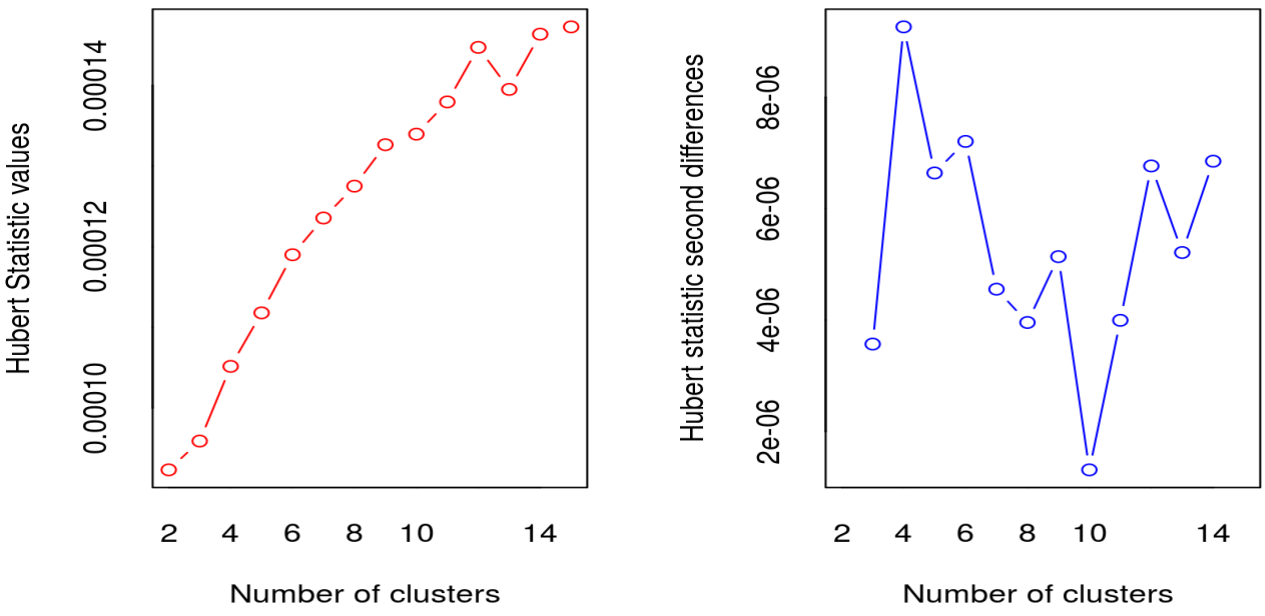
In order to be sure that it is indeed the methodology that is unsuitable rather than the data being insufficiently processed, looking at a more severe form of data cleansing may prove insightful; to that end, removing all boxplot outliers across all variables and running the whole process again is worth it just to see if the results are more conclusive.

```
wine.properties <- names(wine.all_but_q)
#
wine.scaled_cleansed_bp_all <- wine.scaled

wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$density %in%
boxplot(wine.scaled_cleansed_bp_all$density, plot=FALSE)$out),
]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`free sulfur dioxide` %in%
boxplot(wine.scaled_cleansed_bp_all$`free sulfur dioxide`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`residual sugar` %in%
boxplot(wine.scaled_cleansed_bp_all$`residual sugar`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
```

```
(wine.scaled_cleansed_bp_all$`citric acid` %in%
boxplot(wine.scaled_cleansed_bp_all$`citric acid`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`fixed acidity` %in%
boxplot(wine.scaled_cleansed_bp_all$`fixed acidity`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`volatile acidity` %in%
boxplot(wine.scaled_cleansed_bp_all$`volatile acidity`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$chlorides %in%
boxplot(wine.scaled_cleansed_bp_all$chlorides,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`total sulfur dioxide` %in%
boxplot(wine.scaled_cleansed_bp_all$`total sulfur dioxide`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$pH %in%
boxplot(wine.scaled_cleansed_bp_all$pH, plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$sulphates %in%
boxplot(wine.scaled_cleansed_bp_all$sulphates,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$alcohol %in%
boxplot(wine.scaled_cleansed_bp_all$alcohol, plot=FALSE)$out),
]

# for (prop in wine.properties) {
# wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all[prop] %in%
boxplot(wine.scaled_cleansed_bp_all[prop], plot=FALSE)$out), ]
# }

boxplot(wine.scaled_cleansed_bp_all, main="Boxplot all outliers
cleansed", xlab="Wine Attributes", ylab="Scaled values")
```

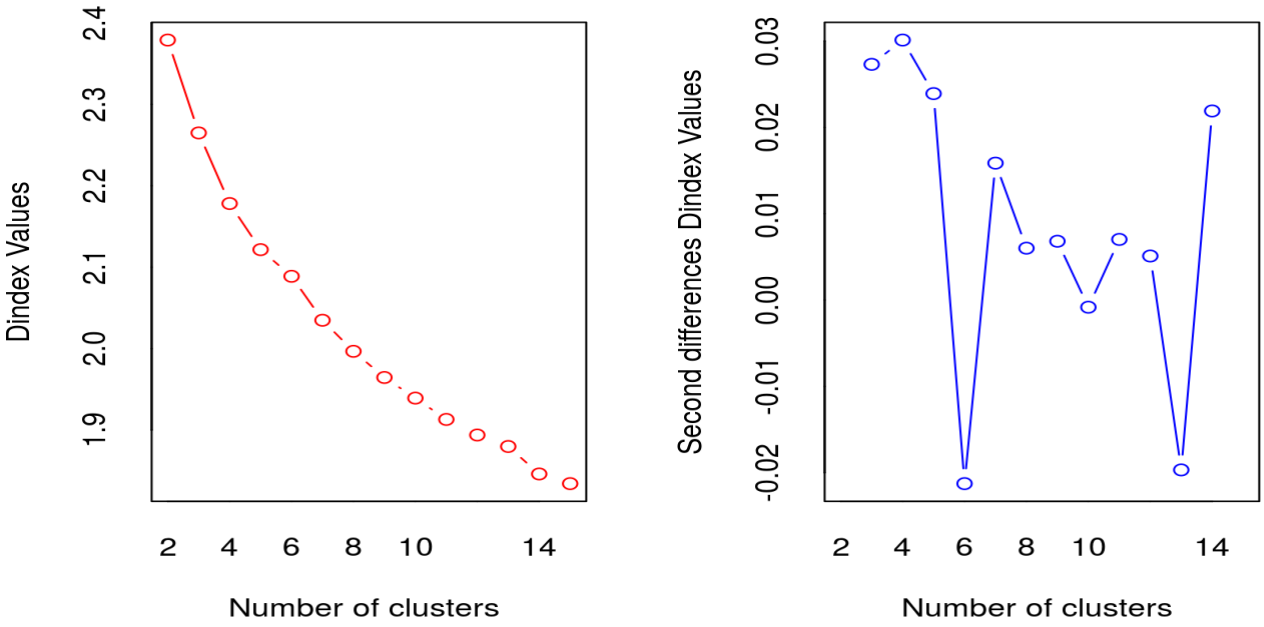



```
number_of_clusters_severe_cleanse <-
NbClust(wine.scaled_cleansed_bp_all,
min.nc=2, max.nc=15,
method="kmeans")
```




```
## *** : 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
## * 5 proposed 3 as the best number of clusters
## * 2 proposed 4 as the best number of clusters
## * 1 proposed 10 as the best number of clusters
## * 4 proposed 14 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
##
##
##
*****************************************************************
```

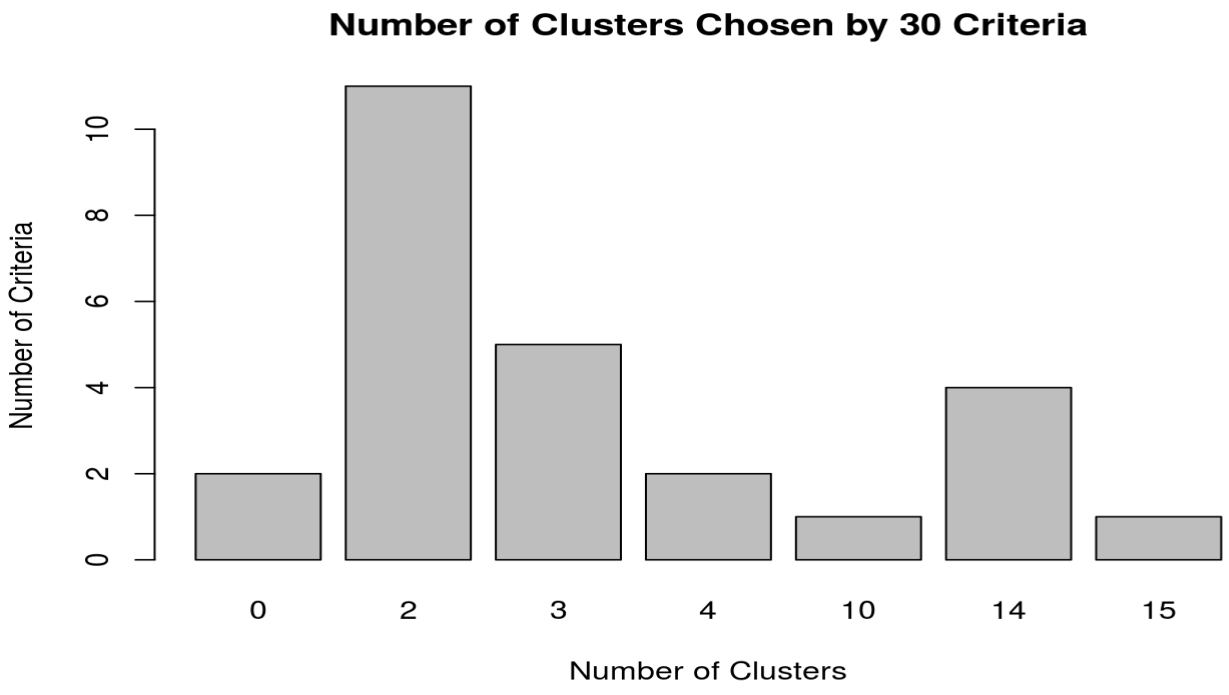
The following table displays the results recommending potential values for k

```
table(number_of_clusters_severe_cleanse$Best.n[1,])
```

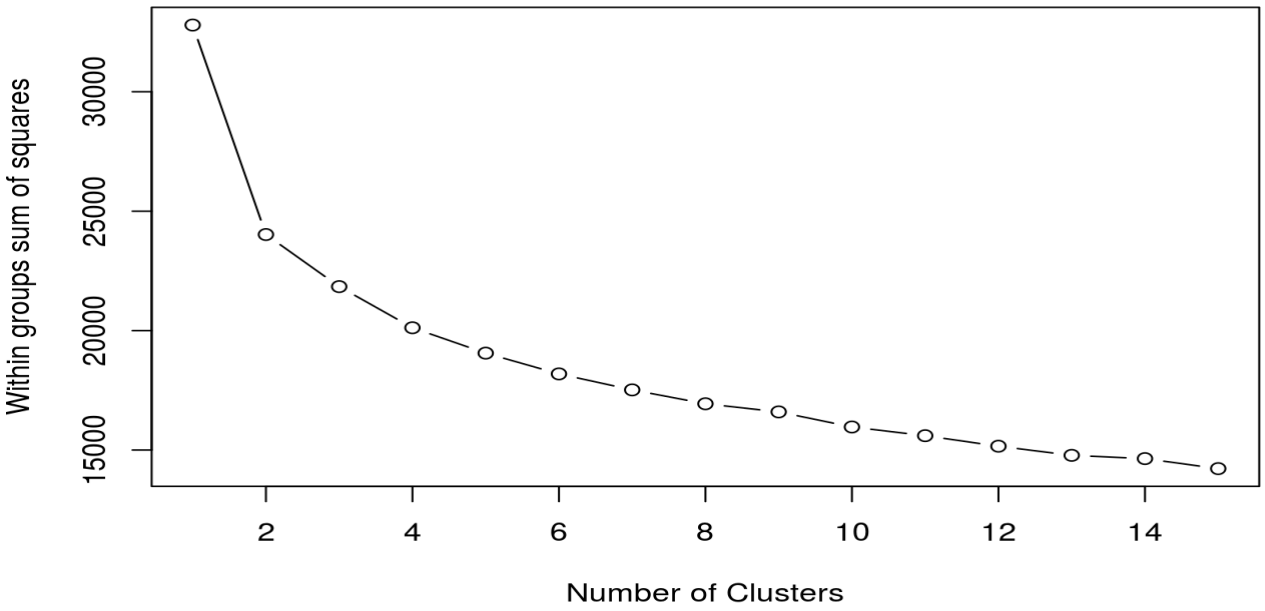
```
##
## 0 2 3 4 10 14 15
## 2 11 5 2 1 4 1
```

The bar chart more easily conveys this.

```
barplot(table(number_of_clusters_severe_cleanse$Best.n[1,]),
xlab="Number of Clusters",
ylab="Number of Criteria",
main="Number of Clusters Chosen by 30 Criteria")
```

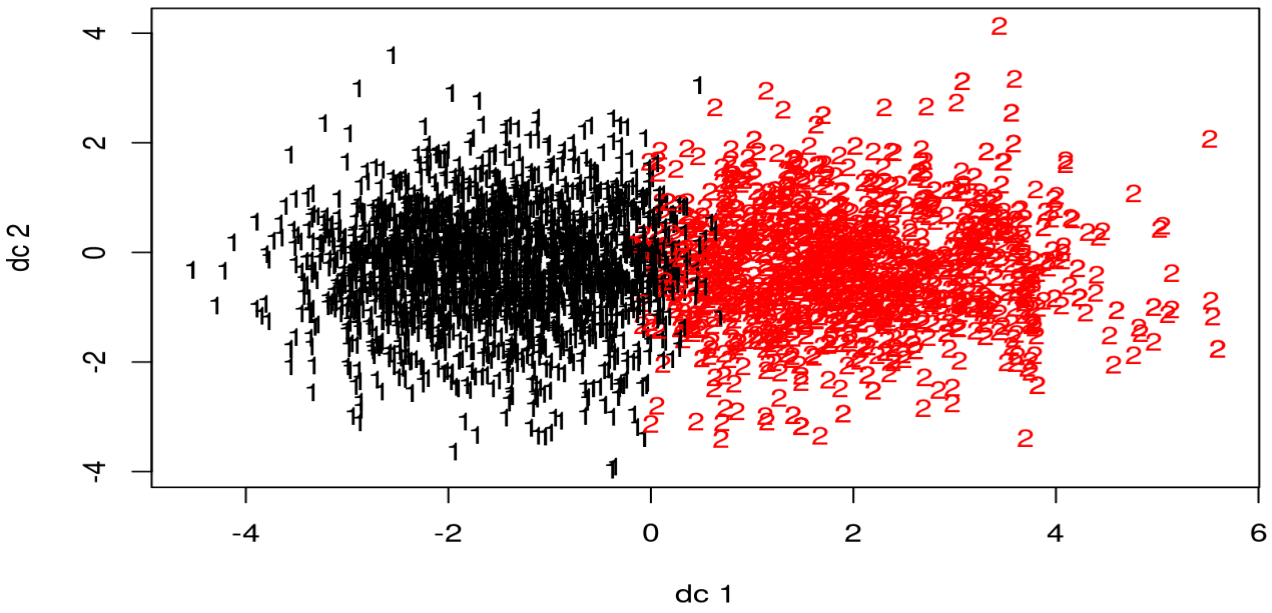


```
sse_list <- 0
for (i in 1:15){
sse_list[i] <- sum(kmeans(wine.scaled_cleansed_bp_all,
centers=i)$withinss)
}

plot(1:15,
sse_list,
type="b",
xlab="Number of Clusters",
ylab="Within groups sum of squares")
```

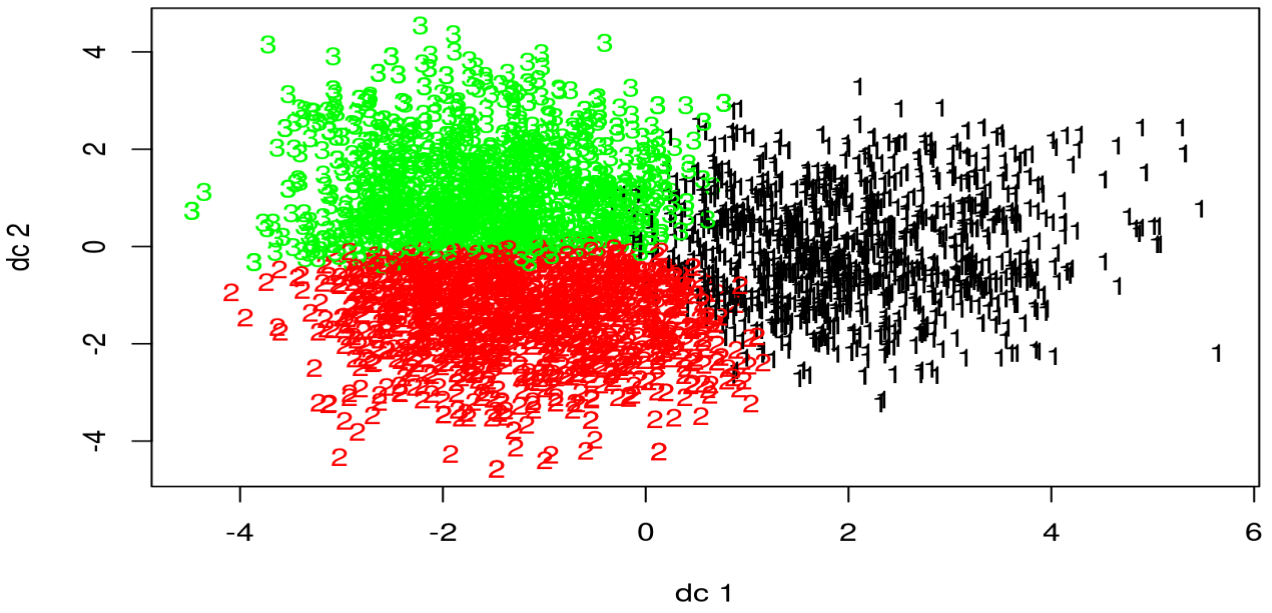



```
#If we're going to run tests on the k-means against the
severely cleansed data we need to remove the outliers from our
quality column too
bp_severe_outliers <- unique(unlist(mapply(function(x, y)
sapply(setdiff(x, y), function(d) which(x==d)), wine.scaled,
wine.scaled_cleansed_bp_all)))

wine.q_cleansed_severe <- wine.q[-bp_severe_outliers]

fit_severe.km2 <- kmeans(wine.scaled_cleansed_bp_all, 2)
fit_severe.km3 <- kmeans(wine.scaled_cleansed_bp_all, 3)
fit_severe.km4 <- kmeans(wine.scaled_cleansed_bp_all, 4)
fit_severe.km5 <- kmeans(wine.scaled_cleansed_bp_all, 5)
fit_severe.km6 <- kmeans(wine.scaled_cleansed_bp_all, 6)
fit_severe.km7 <- kmeans(wine.scaled_cleansed_bp_all, 7)
```

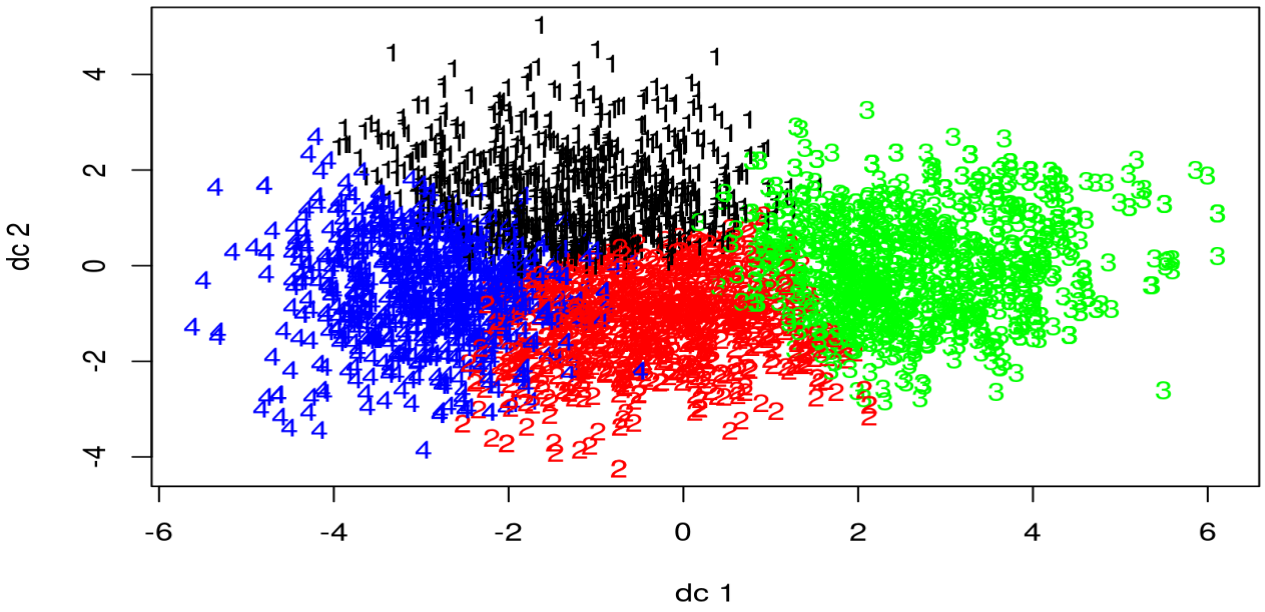
```
fit_severe.km11 <- kmeans(wine.scaled_cleansed_bp_all, 11)
fit_severe.km14 <- kmeans(wine.scaled_cleansed_bp_all, 14)

plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km2$cluster)
```



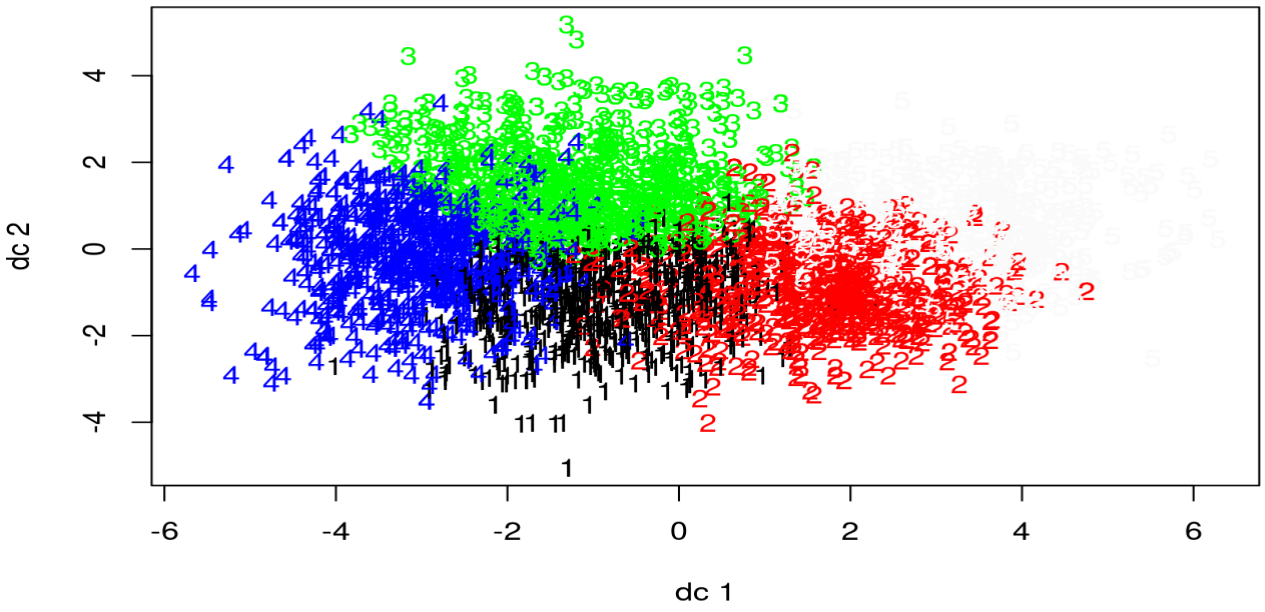

```
plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km3$cluster)
```



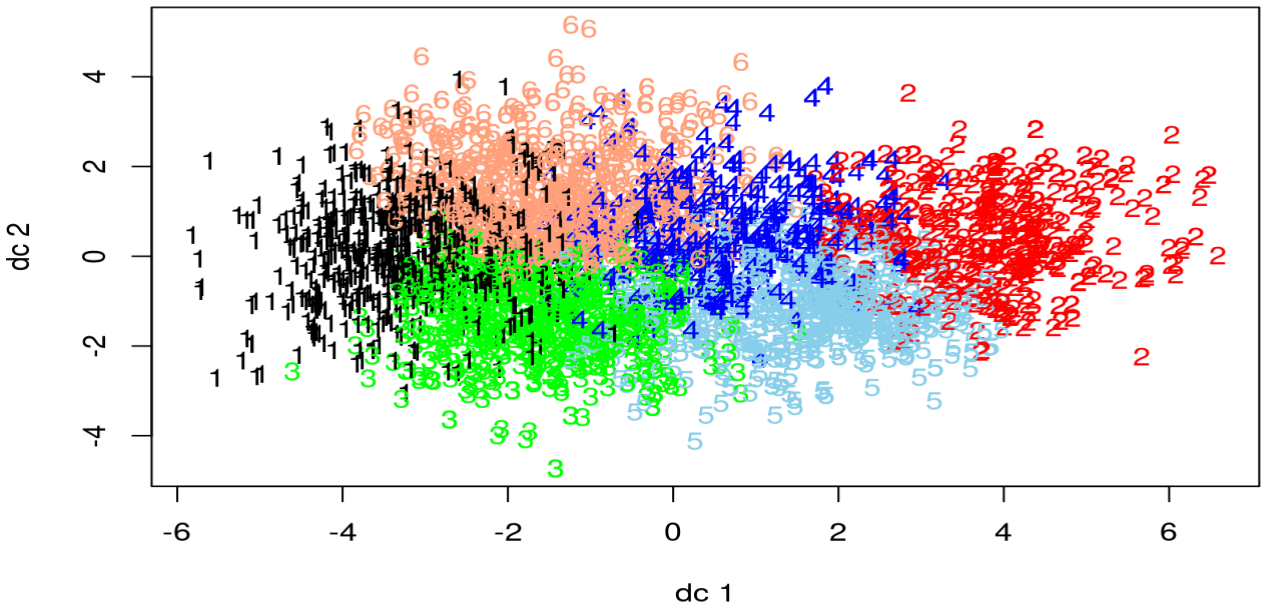
```
plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km4$cluster)
```



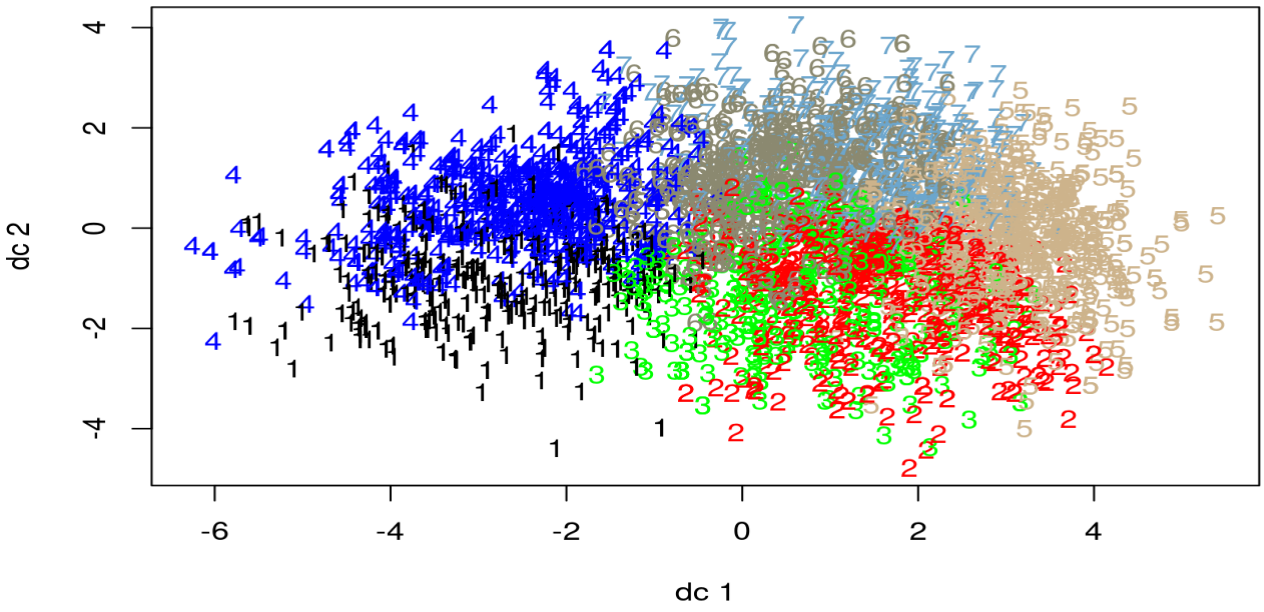

```
plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km5$cluster)
```



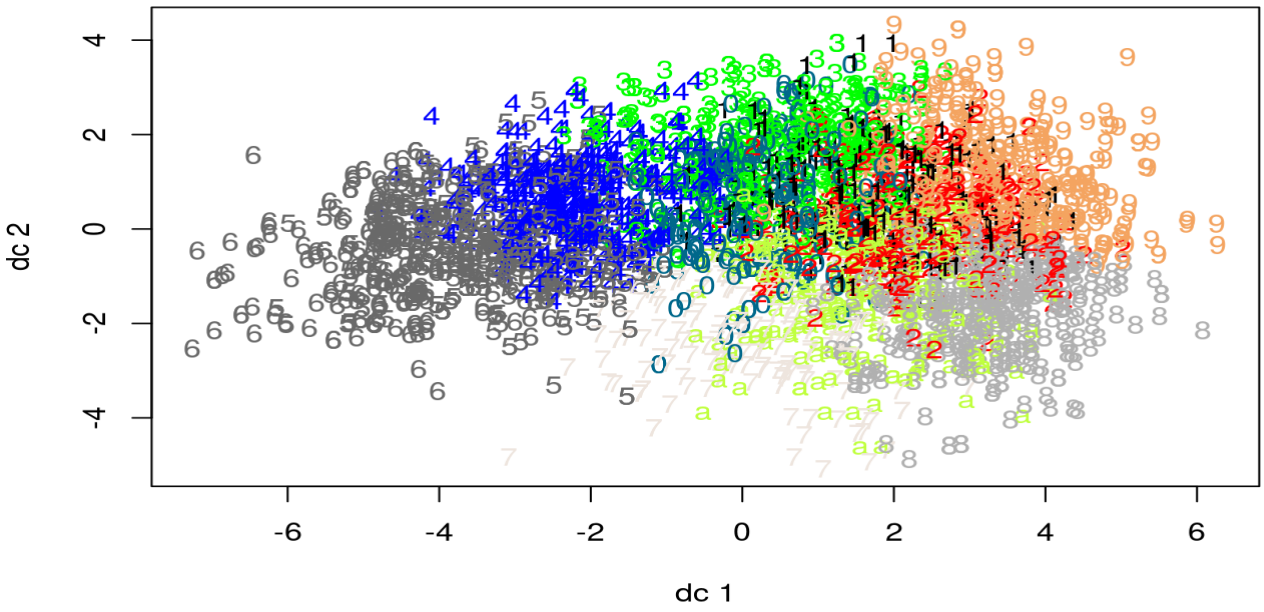
```
plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km6$cluster)
```



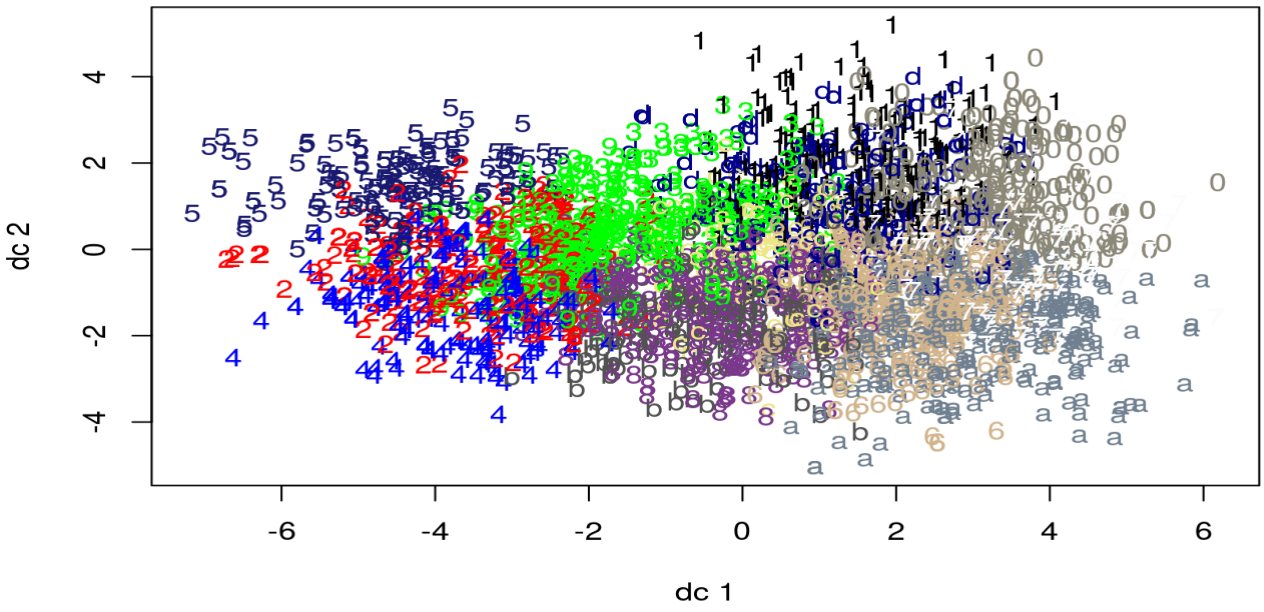

```
plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km7$cluster)
```



```
plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km11$cluster)
```




```
plotcluster(wine.scaled_cleansed_bp_all,
fit_severe.km14$cluster)
```



```
confuseTable_severe.km7 <- table(wine.q_cleansed_severe,
fit_severe.km7$cluster)

names(dimnames(confuseTable_severe.km7)) <- list("Quality",
"Clusters")

confuseTable_severe.km7
```

```
## Clusters
## Quality 1 2 3 4 5 6 7
## 3 1 4 0 1 2 1 0
## 4 10 32 2 9 8 23 4
## 5 193 180 59 372 37 196 64
## 6 239 268 186 329 264 338 240
## 7 77 84 116 40 241 96 141
## 8 14 8 38 7 40 10 28
## 9 0 0 0 0 4 0 0
```

```
randIndex(confuseTable_severe.km7)
```

```
## ARI
## 0.0313044
```

Results so far

Given that the poor value of 0.03410079 for the dataset that was only lightly ‘pruned’ is actually closer to 1 than the 0.02536692 of the last result, the previous dataset actually led to a better set of clusters to act as predictors of Quality.

Alternative clusters

It would appear that the less intensive data cleansing was more appropriate if the ARI value is anything to go by. To refresh what the confusion matrix for that looked like, it is repeated below:

```
confuseTable.km7
```

```
## Clusters
## Quality 1 2 3 4 5 6 7
## 3 0 6 6 2 1 2 2
## 4 10 18 48 18 4 19 46
## 5 71 409 278 327 49 51 271
## 6 267 366 396 347 47 327 440
## 7 163 34 142 102 2 324 113
## 8 31 6 22 18 2 76 20
## 9 0 0 1 0 0 4 0
```

```
randIndex(confuseTable.km7)
```

```
## ARI
## 0.03058022
```

If we look at this more closely, we can see that while there are wines of various qualities spread across all clusters, there are some clusters are weighted in favour of higher quality wines or the middle range. From this observation it can be posited that some more meaningful fitting might be found between factors of quality, “Good”, “Mediocre” and “Bad”. So one final experiment before drawing to a conclusion is to try to fit the data against 3 clusters.

Creating 3 quality factors & attempting one last fit.

```
wine.q_cleansed_f3 <- cut(wine.q_cleansed, 3, labels = c("bad",
"mediocre", "good"))

confuseTable.km3 <- table(wine.q_cleansed, fit.km3$cluster)
names(dimnames(confuseTable.km3)) <- list("Quality",
"Clusters")

confuseTable.km3_f3 <- table(wine.q_cleansed_f3,
fit.km3$cluster)
names(dimnames(confuseTable.km3_f3)) <- list("Quality",
"Clusters")

confuseTable.km3
```

```
## Clusters
## Quality 1 2 3
## 3 2 10 7
## 4 45 42 76
## 5 282 786 388
## 6 731 789 670
## 7 462 141 277
## 8 90 27 58
## 9 3 0 2
```

```
randIndex(confuseTable.km3)
```

```
## ARI
## 0.0349165
```

```
confuseTable.km3_f3
```

```
## Clusters
## Quality 1 2 3
## bad 329 838 471
## mediocre 1193 930 947
## good 93 27 60
```

```
randIndex(confuseTable.km3_f3)
```

```
## ARI
## 0.02601754
```

The results of these other attempts to fit the data against 3 clusters has not yielded (significantly) better results. The very last thing is to see how the suggestion by NbClust works out.

Fitting to NbClust suggested k = 2

```
wine.q_cleansed_f2 <- cut(wine.q_cleansed, 2, labels = c("bad",
"good"))
```

```
confuseTable.km2 <- table(wine.q_cleansed, fit.km2$cluster)
names(dimnames(confuseTable.km2)) <- list("Quality",
"Clusters")

confuseTable.km2_f2 <- table(wine.q_cleansed_f2,
fit.km3$cluster)
names(dimnames(confuseTable.km2_f2)) <- list("Quality",
"Clusters")

confuseTable.km2
```

```
## Clusters
## Quality 1 2
## 3 8 11
## 4 109 54
## 5 608 848
## 6 1329 861
## 7 725 155
## 8 147 28
## 9 4 1
```

```
randIndex(confuseTable.km2)
```

```
## ARI
## 0.02553242
```

```
confuseTable.km2_f2
```

```
## Clusters
## Quality 1 2 3
## bad 1060 1627 1141
## good 555 168 337
```

```
randIndex(confuseTable.km2_f2)
```

```
## ARI
## 0.0346663
```

Writing up for the best results

According to the ARI values the highest being 0.03450436, for 3 clusters with 7 unique quality values, this would be the most successful k-means clustering of those explored, though only marginally. So before we reach out conclusion it’s necessary to display the characteristics of this particular set of clusters for k = 3.

```
#fit.km3

#K-means clustering with 3 clusters of sizes:
fit.km3$size
```

```
## [1] 1615 1795 1478
```

```
#Cluster means:
fit.km3$centers
```

```
## fixed acidity volatile acidity citric acid residual sugar
chlorides
## 1 -0.7518320 -0.03341504 -0.36556862 -0.5949961 -0.2784045
## 2 0.1221039 0.03463949 0.22136042 0.9244884 0.4093015
## 3 0.6629243 -0.01252144 0.09769788 -0.4847543 -0.1947752
## free sulfur dioxide total sulfur dioxide density pH
## 1 -0.2110590 -0.3806105 -0.6686403 0.7951063
## 2 0.6373365 0.7925332 0.9958298 -0.2184079
## 3 -0.5539881 -0.5553075 -0.4902981 -0.6038568
## sulphates alcohol
## 1 0.23936045 0.5421105
## 2 0.06211615 -0.8403982
## 3 -0.33752884 0.4237459
```

Conclusion

By the looks of it, this use of k-means is simply not appropriate against this set of data; it would seem that either using Principal Component Analysis or applying a method for being selective about which variables are used when looking for certain trends would be required in order to reduce the noise that comes from having so many dimensions. One tool in Data Science that could have proved beneficial for this would have been Exploratory Data Analysis; finding relationships between different variables and those connections to Quality might have led to a better understanding of factors affecting wine qualities, resulting some idea of how to select only certain variable to apply k-means to. Additionally, would consider initially testing against a smaller sample of data next time before investing so many CPU cycles to this task!

Question 2: White Wine clustering (Hierarchical)

Premise

> You need to conduct the hierarchical clustering (agglomerative) clustering analysis of the white wine sheet. Investigate the hclust() function for single, complete, average methods. Create the visualization of all methods using a dendrogram. Look at the cophenetic correlation between each clustering result using cor.dendlist. Discuss the produced results after using the coorplot function. Write a code in R Studio to address all the above issues.

Preparation of data

As with Question 1, the data needs to be loaded, partitioned, then scaled. Assuming that this has all been done as before - there's no need to demonstrate the exact same execution of code - we can simply demonstrate we have our data ready to work with.

Important Pre-processing issue

In carrying out the experiments with creating the 3 clusters and then comparing them, I experienced an issue with the R runtime due to the sheer size of the dataset and how was transformed into complexity of the hierarchical clusters; the console reported a `node stack overflow` and after some research online the only option was to use a smaller dataset. In the previous code block you can see the `sample` function used to reduce the dataset to be studied to approximately two-thirds of the original size. Unfortunately, this may well have an adverse impact on results but there's not much I can do about that apart from maybe run the clustering again (multiple times) but with a fresh sample of the same size and then see if there's much variation.

That issue aside, what follows is the summary of the sampled & scaled data.

```
summary(wine.scaled)
```

```
## fixed acidity volatile acidity citric acid residual sugar
## Min. :-3.60000 Min. :-1.9774 Min. :-2.7479 Min. :-1.1431
## 1st Qu.:-0.66330 1st Qu.:-0.6880 1st Qu.:-0.5219 1st
Qu.:-0.9285
## Median :-0.07596 Median :-0.1921 Median :-0.1921 Median
:-0.2261
## Mean : 0.00000 Mean : 0.0000 Mean : 0.0000 Mean : 0.0000
## 3rd Qu.: 0.51138 3rd Qu.: 0.4030 3rd Qu.: 0.3850 3rd Qu.:
0.6715
## Max. : 8.61666 Max. : 8.1396 Max. :10.9377 Max. :11.5783
## chlorides free sulfur dioxide total sulfur dioxide
## Min. :-1.5412 Min. :-1.91571 Min. :-3.0229
## 1st Qu.:-0.4431 1st Qu.:-0.71125 1st Qu.:-0.6913
## Median :-0.1229 Median :-0.08034 Median :-0.1084
## Mean : 0.0000 Mean : 0.00000 Mean : 0.0000
## 3rd Qu.: 0.1974 3rd Qu.: 0.60792 3rd Qu.: 0.6669
## Max. :13.7394 Max. :14.54522 Max. : 7.0264
## density pH sulphates alcohol
## Min. :-2.30592 Min. :-3.1229 Min. :-2.0906 Min. :-2.04129
## 1st Qu.:-0.75613 1st Qu.:-0.6503 1st Qu.:-0.6999 1st
Qu.:-0.81951
## Median :-0.08577 Median :-0.0488 Median :-0.1785 Median
:-0.08645
## Mean : 0.00000 Mean : 0.0000 Mean : 0.0000 Mean : 0.00000
## 3rd Qu.: 0.68415 3rd Qu.: 0.6195 3rd Qu.: 0.5169 3rd Qu.:
0.72807
## Max. :14.90774 Max. : 4.0946 Max. : 5.1233 Max. : 2.84581
```

Hierarchical clustering

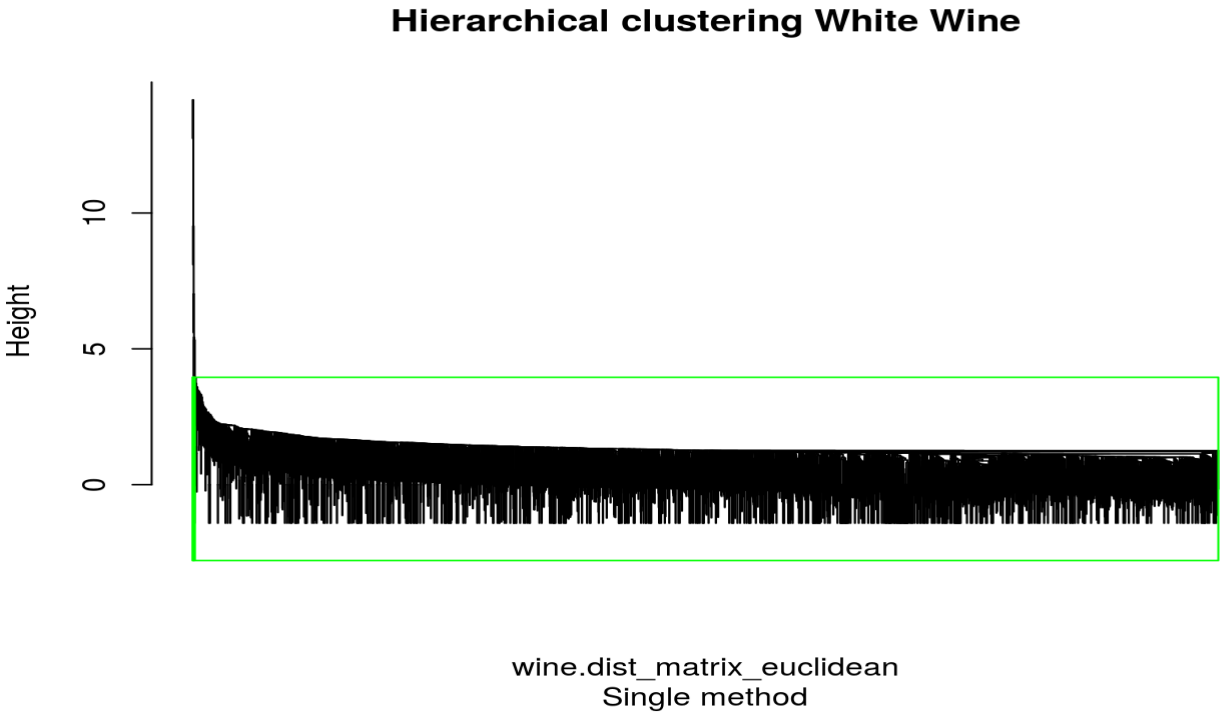
Hierarchical clustering methods use a distance matrix as input for the algorithms; to that end, we need to transform the scaled date into a distance matrix before we pass it into our hierarchical clustering function.

```
wine.dist_matrix_euclidean <- dist(wine.scaled) # Euclidean
distance matrix.
summary(wine.dist_matrix_euclidean)
```

```
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.000 3.378 4.229 4.420 5.206 27.360
```

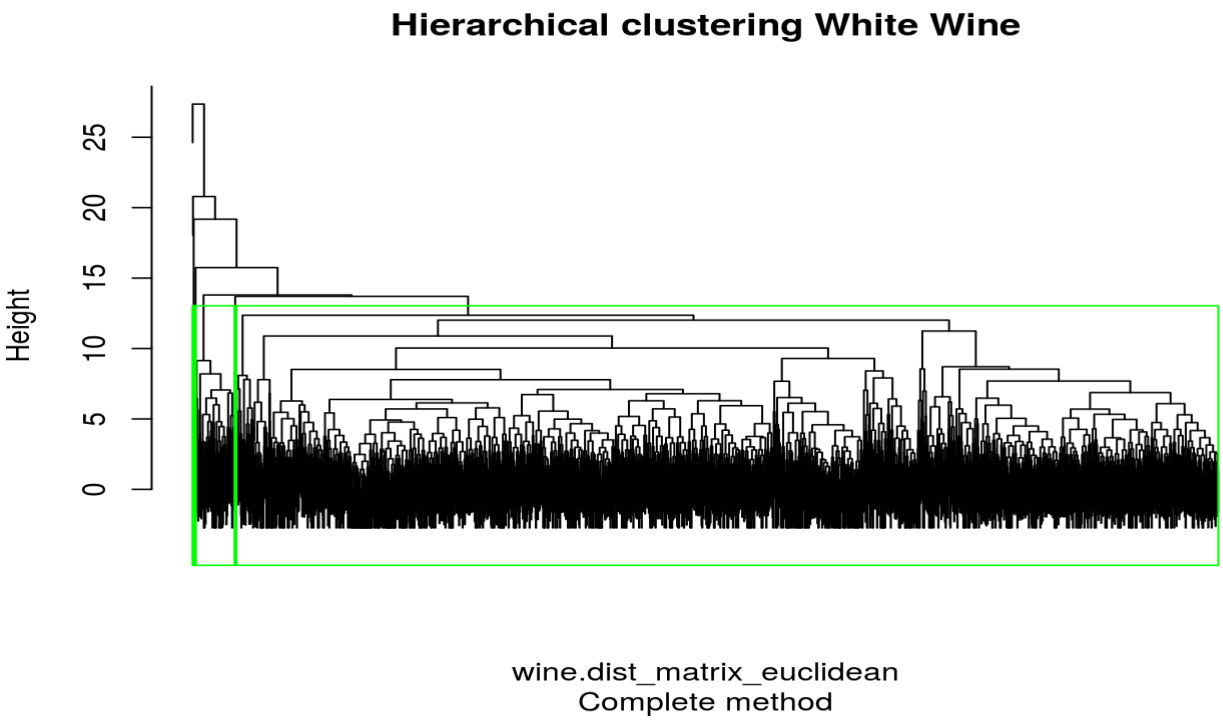
Now we can start clustering data using the hclust library. As per the requirements of this question we are calling hclust 3 times, to evaluate the “single”, “complete” and “average” methods for hierarchical clustering. Assuming this hierarchical clustering is meant to try and find relationships between the quantitative Wine properties and Wine quality, and knowing from the previous question that there are only 7 unique values for quality across the entire dataset, the next step is to cut out cluster trees to only have 7 branches.

```
wine.hclust_single <- hclust(wine.dist_matrix_euclidean, method
= "single")
wine.hclust_complete <- hclust(wine.dist_matrix_euclidean,
method = "complete")
wine.hclust_average <- hclust(wine.dist_matrix_euclidean,
method = "average")

#Display dendrograms
plot(wine.hclust_single, main="Hierarchical clustering White
Wine", sub = "Single method", labels=FALSE)
rect.hclust(wine.hclust_single, k=7, border="green")
```

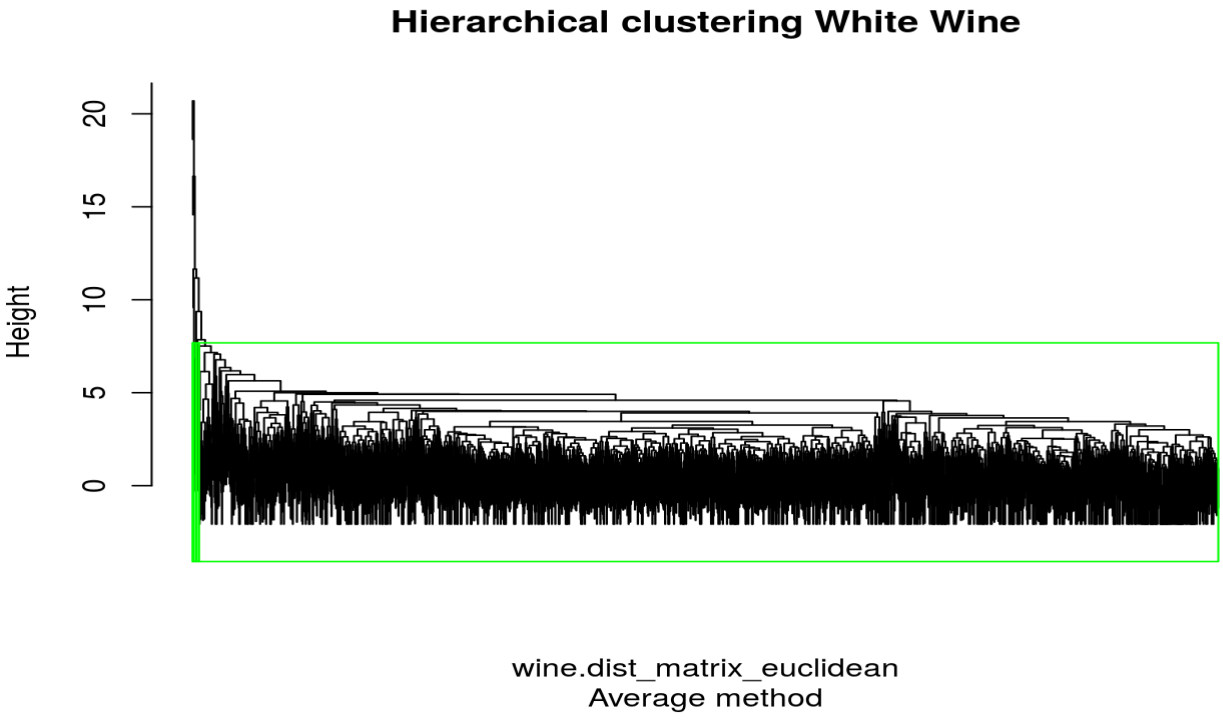



```
plot(wine.hclust_complete, main="Hierarchical clustering White
Wine", sub = "Complete method", labels=FALSE)
```

```
rect.hclust(wine.hclust_complete, k=7, border="green")
```




```
plot(wine.hclust_average, main="Hierarchical clustering White
Wine", sub = "Average method", labels=FALSE)
rect.hclust(wine.hclust_average, k=7, border="green")
```



```
wine.hclust_single_g7 <- cutree(wine.hclust_single, k=7)
table(wine.hclust_single_g7)
```

```
## wine.hclust_single_g7
## 1 2 3 4 5 6 7
## 3225 2 1 1 1 1 1
```

```
wine.hclust_complete_g7 <- cutree(wine.hclust_complete, k=7)
table(wine.hclust_complete_g7)
```

```
## wine.hclust_complete_g7
## 1 2 3 4 5 6 7
## 3094 123 7 1 5 1 1
```

```
wine.hclust_average_g7 <- cutree(wine.hclust_average, k=7)
table(wine.hclust_average_g7)
```

```
## wine.hclust_average_g7
## 1 2 3 4 5 6 7
## 3212 8 8 1 1 1 1
```

Interpreting the data

My observation from the cut trees is that unless the data is especially skewed, it seems like using the Complete method is the one method of the three tried that might have any hope of clustering in a way that could correlate, albeit weakly, with Quality. Having said that, this is not the main task of this question, which is to compare the dendrograms against one another to see how similar they are; To do this we convert the cluster models to the dendrogram data type before using `cor.dendlist` to cross-compare the trees.

```
d_list <- dendlist(
"Single" = wine.hclust_single %>% as.dendrogram,
"Complete" = wine.hclust_complete %>% as.dendrogram,
"Average" = wine.hclust_average %>% as.dendrogram
)
```

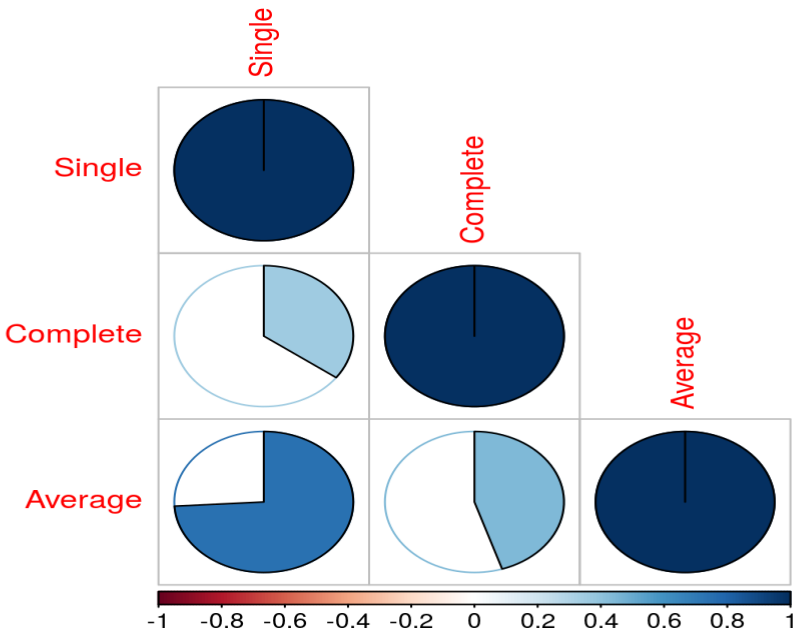
```
cophentic_coefficient <- cor.dendlist(d_list, "cophenetic")
```

Below is a matrix where the dendrograms are compared on a scale of 0 to 1 where 1 is 100% parity; below that is a digram expressing the same data but using pie charts to convey the information. By the share of the pies, we can see that the “single” (1) and “average” (3) methods deliver trees that are more similar to each other than they are to “complete” (**Comparing dendrograms: Essentials - articles - sthda, 2017**).

```
# Print correlation matrix
round(cophentic_coefficient, 2)
```

```
## Single Complete Average
## Single 1.00 0.35 0.74
## Complete 0.35 1.00 0.45
## Average 0.74 0.45 1.00
```

```
corrplot(cophentic_coefficient, "pie", "lower")
```



While we can see from coorplot, of even the corresponding table that the “Single” and “Average” dendrograms are 74% alike, this doesn’t mean they are better! We can run the hierarchical clustering through a confusion matrix as we did with k-means and find that

the results do not give us any meaningful relationship to Wine Quality. In fact all clustering methods used have ended with a massive bias towards one single cluster for which most nodes belong, across all qualities of wine!

```
hclust_single_g7_table <- table(wine.q, wine.hclust_single_g7)
names(dimnames(hclust_single_g7_table)) <- list("Quality",
"Clusters")
hclust_single_g7_table
```

```
## Clusters
## Quality 1 2 3 4 5 6 7
## 3 13 0 0 0 1 0 0
## 4 117 0 0 0 0 0 0
## 5 961 0 1 0 0 0 0
## 6 1437 2 0 1 0 1 1
## 7 584 0 0 0 0 0 0
## 8 112 0 0 0 0 0 0
## 9 1 0 0 0 0 0 0
```

```
randIndex(hclust_single_g7_table)
```

```
## ARI
## -0.0004977646
```

```
hclust_complete_g7_table <- table(wine.q,
wine.hclust_complete_g7)
names(dimnames(hclust_complete_g7_table)) <- list("Quality",
"Clusters")
hclust_complete_g7_table
```

```
## Clusters
## Quality 1 2 3 4 5 6 7
## 3 12 0 1 0 0 1 0
## 4 88 28 1 0 0 0 0
## 5 902 54 5 0 1 0 0
## 6 1406 30 0 1 4 0 1
## 7 576 8 0 0 0 0 0
```

```
## 8 109 3 0 0 0 0 0
## 9 1 0 0 0 0 0 0
```

```
randIndex(hclust_complete_g7_table)
```

```
## ARI
## 0.01511415
```

```
hclust_average_g7_table <- table(wine.q,
wine.hclust_average_g7)
names(dimnames(hclust_average_g7_table)) <- list("Quality",
"Clusters")
hclust_average_g7_table
```

```
## Clusters
## Quality 1 2 3 4 5 6 7
## 3 12 0 1 0 1 0 0
## 4 114 2 1 0 0 0 0
## 5 952 4 6 0 0 0 0
## 6 1438 1 0 1 0 1 1
## 7 583 1 0 0 0 0 0
## 8 112 0 0 0 0 0 0
## 9 1 0 0 0 0 0 0
```

```
randIndex(hclust_average_g7_table)
```

```
## ARI
## 0.002569654
```

Trying again with cleansed data

Just in case outliers have skewed the outcomes I’m going to run all cluster calls with data that has had outliers removed, using the boxplot method.

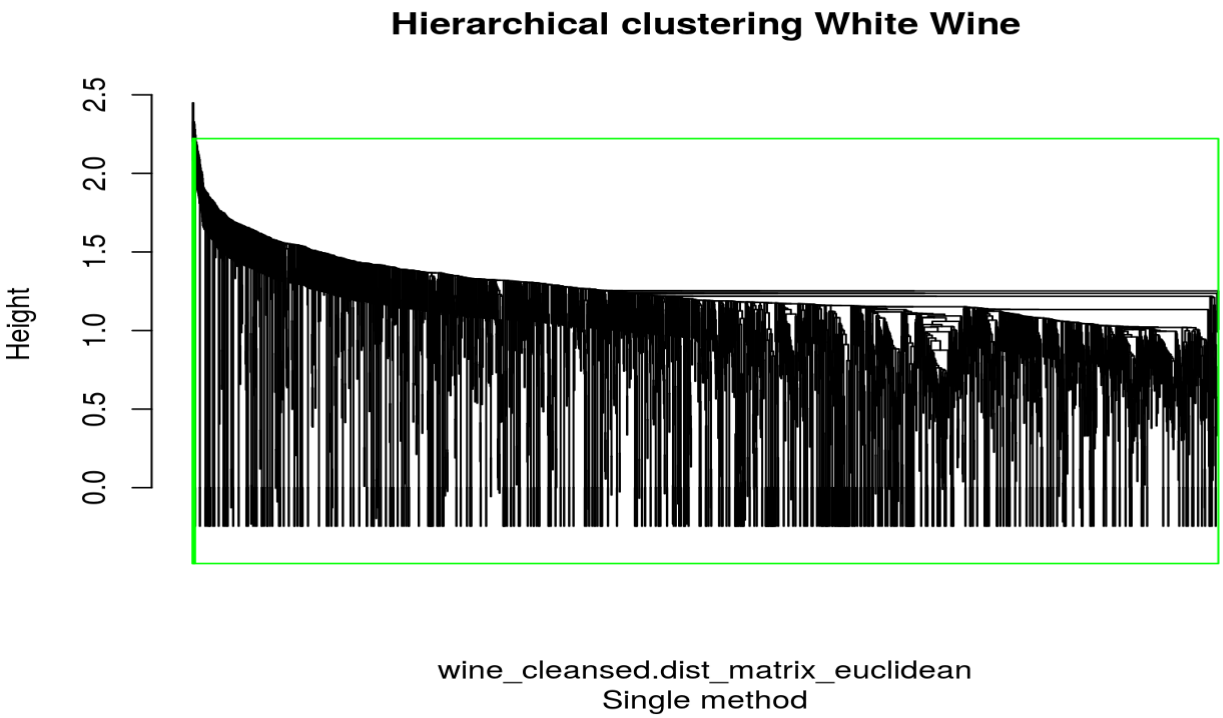
```
wine.scaled_cleansed_bp_all <- wine.scaled

wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$density %in%
boxplot(wine.scaled_cleansed_bp_all$density, plot=FALSE)$out),
]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`free sulfur dioxide` %in%
boxplot(wine.scaled_cleansed_bp_all$`free sulfur dioxide`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`residual sugar` %in%
boxplot(wine.scaled_cleansed_bp_all$`residual sugar`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`citric acid` %in%
boxplot(wine.scaled_cleansed_bp_all$`citric acid`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`fixed acidity` %in%
boxplot(wine.scaled_cleansed_bp_all$`fixed acidity`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`volatile acidity` %in%
boxplot(wine.scaled_cleansed_bp_all$`volatile acidity`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$chlorides %in%
boxplot(wine.scaled_cleansed_bp_all$chlorides,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$`total sulfur dioxide` %in%
boxplot(wine.scaled_cleansed_bp_all$`total sulfur dioxide`,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$pH %in%
boxplot(wine.scaled_cleansed_bp_all$pH, plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$sulphates %in%
boxplot(wine.scaled_cleansed_bp_all$sulphates,
plot=FALSE)$out), ]
wine.scaled_cleansed_bp_all <- wine.scaled_cleansed_bp_all[ !
(wine.scaled_cleansed_bp_all$alcohol %in%
boxplot(wine.scaled_cleansed_bp_all$alcohol, plot=FALSE)$out),
]

wine_cleansed.dist_matrix_euclidean <-
```

```
dist(wine.scaled_cleansed_bp_all) # Euclidean distance matrix.
summary(wine_cleansed.dist_matrix_euclidean)
```

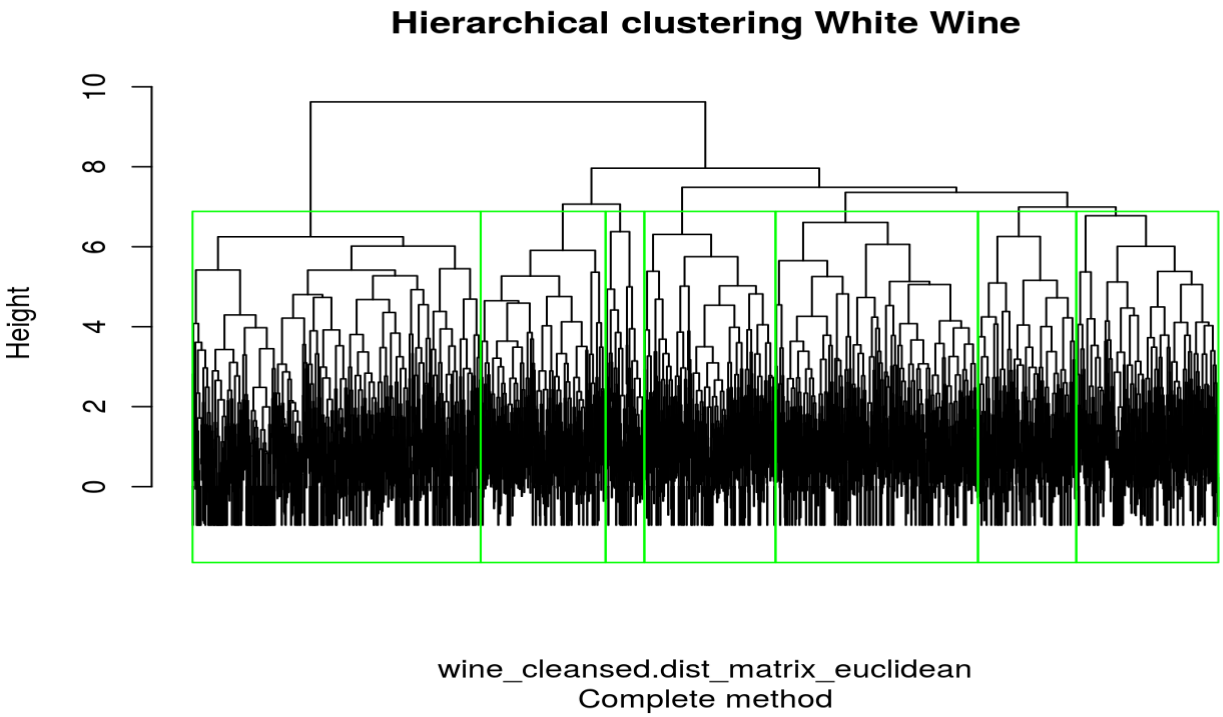
```
## Min. 1st Qu. Median Mean 3rd Qu. Max.
## 0.000 3.095 3.798 3.869 4.577 9.623
```

Now we can start clustering data using the hclust library. As per the requirements of this question we are calling hclust 3 times, to evaluate the “single”, “complete” and “average” methods for hierarchical clustering. Assuming this hierarchical clustering is meant to try and find relationships between the quantitative Wine properties and Wine quality, and knowing from the previous question that there are only 7 unique values for quality across the entire dataset, the next step is to cut out cluster trees to only have 7 branches (**Boateng, 2017**).

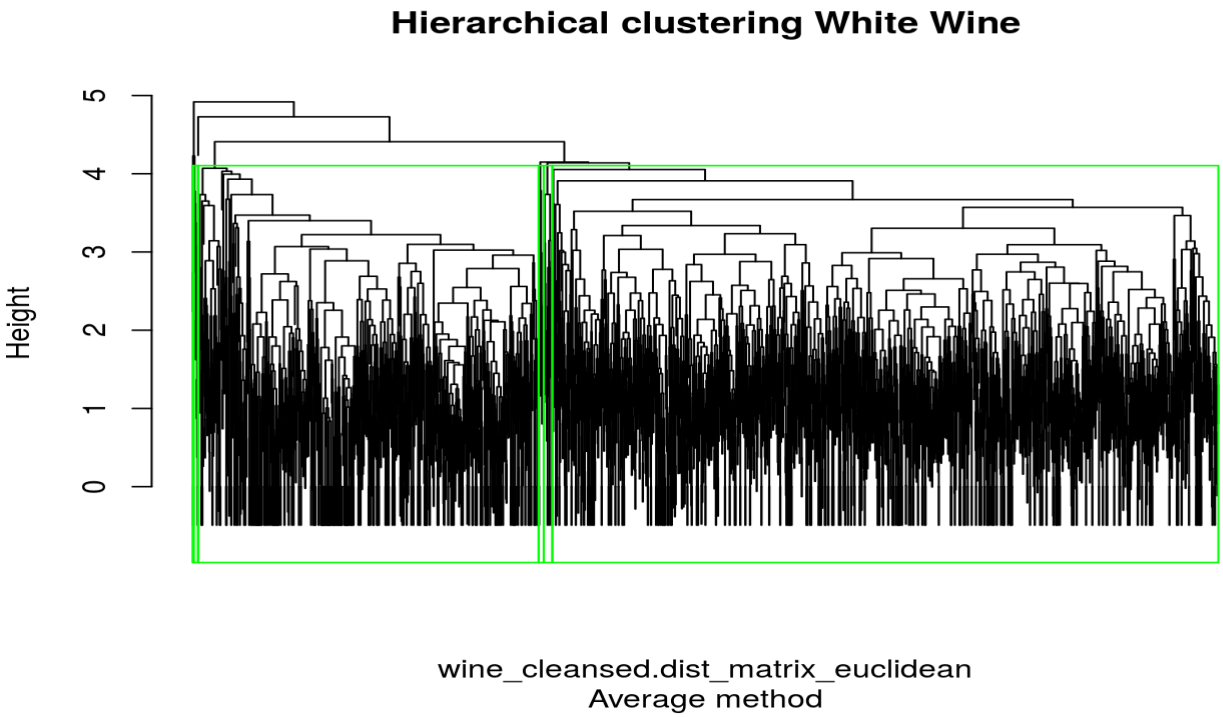
```
wine_cleansed.hclust_single <-
hclust(wine_cleansed.dist_matrix_euclidean, method = "single")
wine_cleansed.hclust_complete <-
hclust(wine_cleansed.dist_matrix_euclidean, method =
"complete")
wine_cleansed.hclust_average <-
hclust(wine_cleansed.dist_matrix_euclidean, method = "average")

#Display dendrograms
plot(wine_cleansed.hclust_single, main="Hierarchical clustering
White Wine", sub = "Single method", labels=FALSE)
rect.hclust(wine_cleansed.hclust_single, k=7, border="green")
```




```
plot(wine_cleansed.hclust_complete, main="Hierarchical
clustering White Wine", sub = "Complete method", labels=FALSE)
rect.hclust(wine_cleansed.hclust_complete, k=7, border="green")
```




```
plot(wine_cleansed.hclust_average, main="Hierarchical
clustering White Wine", sub = "Average method", labels=FALSE)
rect.hclust(wine_cleansed.hclust_average, k=7, border="green")
```




```
wine_cleansed.hclust_single_g7 <- cutree(wine.hclust_single,
k=7)
table(wine_cleansed.hclust_single_g7)
```

```
## wine_cleansed.hclust_single_g7
## 1 2 3 4 5 6 7
## 3225 2 1 1 1 1 1
```

```
wine_cleansed.hclust_complete_g7 <-
cutree(wine.hclust_complete, k=7)
table(wine_cleansed.hclust_complete_g7)
```

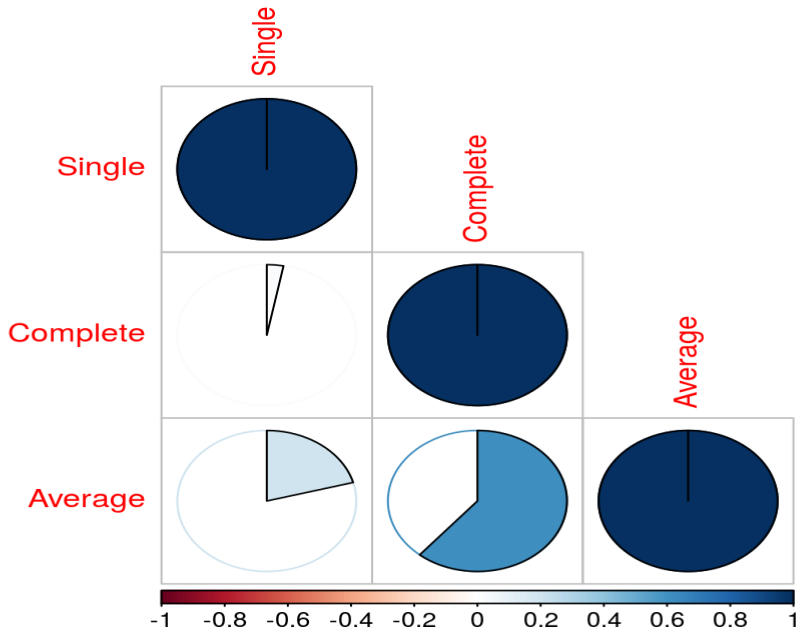
```
## wine_cleansed.hclust_complete_g7
## 1 2 3 4 5 6 7
## 3094 123 7 1 5 1 1
```

```
wine_cleansed.hclust_average_g7 <- cutree(wine.hclust_average,
k=7)
table(wine_cleansed.hclust_average_g7)
```

```
## wine_cleansed.hclust_average_g7
## 1 2 3 4 5 6 7
## 3212 8 8 1 1 1 1
```

Interestingly, it's already apparent from the dendrograms that the "single" and "average" trees are less similar with this set of data, and the "average" is now somewhere between the "single" and "complete" while the "single" outcome remains largely unchanged. As such, I think it merits the expense of plotting the diagrams to visually display the statistical comparison.

```
cleansed_d_list <- dendlist(
"Single" = wine_cleansed.hclust_single %>% as.dendrogram,
"Complete" = wine_cleansed.hclust_complete %>% as.dendrogram,
"Average" = wine_cleansed.hclust_average %>% as.dendrogram
)

cleansed_cophentic_coefficient <- cor.dendlist(cleansed_d_list,
"cophenetic")



# Print correlation matrix
round(cleansed_cophentic_coefficient, 2)



## Single Complete Average
## Single 1.00 0.03 0.21
## Complete 0.03 1.00 0.61
## Average 0.21 0.61 1.00



corrplot(cleansed_cophentic_coefficient, "pie", "lower")
```



The results above suggest that removing outlies makes a massive difference to how these methods operate; this must be considered in particular with my choice is keeping with the Euclidean type of distance matrix.

```
wine_cleansed.hclust_complete_g7_table <- table(wine.q,
wine_cleansed.hclust_complete_g7)
names(dimnames(wine_cleansed.hclust_complete_g7_table)) <-
list("Quality", "Clusters")
wine_cleansed.hclust_complete_g7_table
```

```
## Clusters
## Quality 1 2 3 4 5 6 7
## 3 12 0 1 0 0 1 0
## 4 88 28 1 0 0 0 0
## 5 902 54 5 0 1 0 0
## 6 1406 30 0 1 4 0 1
## 7 576 8 0 0 0 0 0
## 8 109 3 0 0 0 0 0
## 9 1 0 0 0 0 0 0
```

```
randIndex(wine_cleansed.hclust_complete_g7_table)
```

```
## ARI
## 0.01511415
```

Conclusion

Using hierarchical clustering over this dataset has not proved very insightful, in fact less correlation was found between quality and cluster than with the k-means method. Also, the size of the dataset and the complexity of agglomerative clustering, {}(n^{2}(n)) makes the method unsuitable for the original size of the dataset. Even though the hierarchical cluster results haven't shown obvious relationship to Wine Quality; seeing the dendrograms and the `coorplot` output, if I were to use this method of clustering in the future, assuming the Euclidean distance matrix, I think I would definitely have preferences and less favoured clustering methods: I would imagine that the "single" method which clusters just by finding the nearest point between groups of child nodes would be the bottom of my list; think I would pick "Complete" (grouping clusters based on maximum distance) over "Average" though I think more research on this would have to be done in order to get a more informed decision as to why, beyond my own findings.

Question 3: Forecasting (MLP)

Premise

> You need to construct an MLP neural network for this problem. You need to consider the appropriate input vector, as well as the internal network structure (hidden layers, nodes, learning rate). You may consider any de-trending scheme if you feel is necessary. Write a code in R Studio to address all these requirements. You need to show the performance of your network both graphically as well as in terms of usual statistical indices (MSE, RMSE and MAPE). Hint: Experiment with various network structures and show a comparison table of their performances. This will be a good justification for your final network choice. Show all your working steps. As everyone will have different forecasting result, emphasis in the marking scheme will be given to the adopted methodology and the explanation/justification of various decisions you have taken in order to provide an acceptable, in terms of performance, solution. The input selection problem is very important. Experiment with various options (i.e. how many past values you need to consider as potential network inputs).

Preparation of data

The exchange data needs to be loaded, partitioned (into training and testing datasets), and scaled (**Grogan, 2017**).

```
#going to import the Excel spreadsheet Currency Exchange
dataset
exchange.raw <- read_excel("../data/Exchange.xlsx")
```

Here's a glance at the dataset

```
head(exchange.raw)
```

```
## # 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
```

```
str(exchange.raw)
```

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

We want to scale the data to allow for faster training.

```
norm_func <- function(x){
min_x <- min(x)

return((x - min_x)/(max(x) - min_x))
}
```

```
exchange.scaled <- exchange.raw

exchange.scaled[3] <- scale(exchange.scaled[3])

#Summary of scaled wine data
summary(exchange.scaled)
```

```
## YYYY/MM/DD Wdy USD/EUR.USD/EUR
## Min. :2015-03-19 00:00:00 Length:390 Min. :-2.5760135
## 1st Qu.:2015-08-07 18:00:00 Class :character 1st
Qu.:-0.7059906
## Median :2015-12-29 12:00:00 Mode :character Median :
```

```
0.2038797
## Mean :2015-12-28 04:36:55 Mean : 0.0000000
## 3rd Qu.:2016-05-17 18:00:00 3rd Qu.: 0.7569851
## Max. :2016-10-06 00:00:00 Max. : 2.2941870



exchange.normalised <- exchange.raw

exchange.normalised[3] <- norm_func(exchange.normalised[3])

#Summary of scaled wine data
summary(exchange.normalised)



## YYYY/MM/DD Wdy USD/EUR
## Min. :2015-03-19 00:00:00 Length:390 Min. :0.0000
## 1st Qu.:2015-08-07 18:00:00 Class :character 1st Qu.:0.3840
## Median :2015-12-29 12:00:00 Mode :character Median :0.5708
## Mean :2015-12-28 04:36:55 Mean :0.5289
## 3rd Qu.:2016-05-17 18:00:00 3rd Qu.:0.6844
## Max. :2016-10-06 00:00:00 Max. :1.0000
```

And partition the data into a set for training and another for testing the neural network after.

```
exchange.scaled_train <- head(exchange.scaled, 320)
exchange.scaled_test <- tail(exchange.scaled, -320)
#exchange.scaled[-1:-320, 1:3] #also works!


#Summary of scaled data
summary(exchange.scaled_train)



## YYYY/MM/DD Wdy USD/EUR
## Min. :2015-03-19 00:00:00 Length:320 Min. :-2.57601
## 1st Qu.:2015-07-13 18:00:00 Class :character 1st
Qu.:-0.91550
## Median :2015-11-05 12:00:00 Mode :character Median : 0.1296

## Mean :2015-11-07 03:04:30 Mean :-0.06404
## 3rd Qu.:2016-03-03 06:00:00 3rd Qu.: 0.80487
## Max. :2016-06-27 00:00:00 Max. : 2.29419
```

```
summary(exchange.scaled_test)
```

```
## YYYY/MM/DD Wdy USD/EUR
## Min. :2016-06-28 00:00:00 Length:70 Min. :-0.62698
## 1st Qu.:2016-07-22 18:00:00 Class :character 1st
Qu.:-0.07986
## Median :2016-08-17 12:00:00 Mode :character Median : 0.3427

## Mean :2016-08-17 15:05:08 Mean : 0.29275
## 3rd Qu.:2016-09-12 18:00:00 3rd Qu.: 0.64924
## Max. :2016-10-06 00:00:00 Max. : 1.12093
```

```
exchange.normalised_train <- head(exchange.normalised, 320)
exchange.normalised_test <- tail(exchange.normalised, -320)
#exchange.normalised[-1:-320, 1:3] #also works!
```

```
#Summary of normalised data
paste("training:")
```

```
## [1] "training:"
```

```
summary(exchange.normalised_train)
```

```
## YYYY/MM/DD Wdy USD/EUR
## Min. :2015-03-19 00:00:00 Length:320 Min. :0.0000
## 1st Qu.:2015-07-13 18:00:00 Class :character 1st Qu.:0.3410
## Median :2015-11-05 12:00:00 Mode :character Median :0.5556
## Mean :2015-11-07 03:04:30 Mean :0.5158
## 3rd Qu.:2016-03-03 06:00:00 3rd Qu.:0.6942
## Max. :2016-06-27 00:00:00 Max. :1.0000
```

```
paste("test:")
```

```
## [1] "test:"
```

```
summary(exchange.normalised_test)
```

```
## YYYY/MM/DD Wdy USD/EUR
## Min. :2016-06-28 00:00:00 Length:70 Min. :0.4002
## 1st Qu.:2016-07-22 18:00:00 Class :character 1st Qu.:0.5125
## Median :2016-08-17 12:00:00 Mode :character Median :0.5993
## Mean :2016-08-17 15:05:08 Mean :0.5890
## 3rd Qu.:2016-09-12 18:00:00 3rd Qu.:0.6622
## Max. :2016-10-06 00:00:00 Max. :0.7591
```

The time series input problem

Because we're forecasting with time-series data, one of the main concerns when choosing the configuration of the neural network is how many input values are fed into the system in order to derive our output; too few inputs and we lack the data for forecasting but too many and the neural network. In the following process, the number of inputs will be on variable that is experimented with along side the number of hidden layers and back propagation techniques.

Setting up the training data

Because we're looking to train the neural network on time-series data, we have to transform the data to make it easy to pass in time based input, which means we need to provide more than 1 input value at a time to train against the desired output; to that end, below is a code chunk that takes the exchange rate values and creates rows where the last entry is the desired output, the third is considered day 0 and the 1st to entries are 2 previous days from day 0.

Before the data is ready, a function can be created in order to re-use the functionality over and over; this way the same code can be applied to generating out test data as the training data.

```
vector_to_time_series_data <- function(vec, colCount, step = 1)
{
row_count <- length(vec)

staggered_data_matrix <- matrix(vec, row_count, colCount)

for (i in 1:row_count){
new_row <- c(staggered_data_matrix[i])
```

```
for (j in 1:(colCount-1)){
new_row <- c(new_row, staggered_data_matrix[i + (step * j)])
}

staggered_data_matrix[i,] <- new_row
}

return (as.data.frame(head(staggered_data_matrix, row_count -
((colCount -1) * step))))
}
```

With this function we can transform the training currency values now:

```
staggered_data_frame <-
vector_to_time_series_data(exchange.scaled_train$`USD/EUR`, 4)

colnames(staggered_data_frame) <- c("Input_dneg2",
"Input_dneg1", "Input_d0", "Output")

#Summary of training data
head(staggered_data_frame)
```

```
## Input_dneg2 Input_dneg1 Input_d0 Output
## 1 -2.2886860 -1.4745916 -0.8233160 -0.9238806
## 2 -1.4745916 -0.8233160 -0.9238806 -0.5407773
## 3 -0.8233160 -0.9238806 -0.5407773 -0.8664151
## 4 -0.9238806 -0.5407773 -0.8664151 -0.9957125
## 5 -0.5407773 -0.8664151 -0.9957125 -1.3500830
## 6 -0.8664151 -0.9957125 -1.3500830 -1.7140311
```

Creating & using the neural net

Training the network

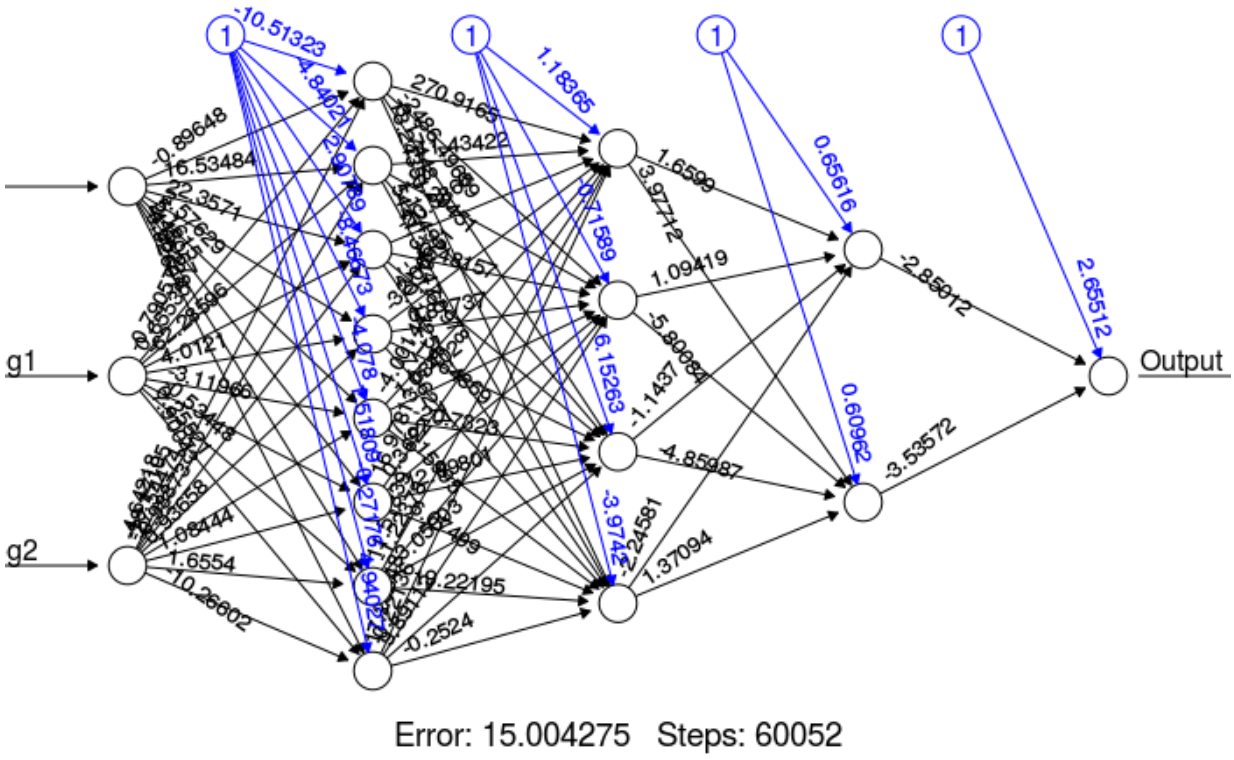
Now we have the data in a format such that we can provide 3 consecutive days as input and the following day as desired output, we are ready to train the neural network.

```
mlp.form1 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2")
```

```
mlp.nn1 <- neuralnet(mlp.form1, staggered_data_frame,
hidden=c(8,4,2), threshold=0.01)
```

Now the neural network has been trained we can view a representation of its structure.

```
plot(mlp.nn1)
```



Testing the network

Before we can test the neural network against test data we need to transform the test data we took from the original dataset and transform it so we can pass in three input variables similarly to how the network was trained. Picking the number of inputs and what those inputs might be is part of the challenge of creating a neural network for time-series data; it's pretty obvious that one input value will hardly help project a future value but what value to pick isn't clear. The lower the number or inputs the greater likelihood of error but the higher the number of inputs, the increased complexity and longer training period. I've picked 3 consecutive days to start with but will experiment with other configurations afterwards.

```
staggered_test_data_frame <-
vector_to_time_series_data(exchange.scaled_test$`USD/EUR`, 3)
```

Now that the test data is arranged in the same way that the training data is, the neural network can be tested.

```
expected_v_test_func <- function (expected, mlp_test) {
expected_v_test <- cbind(expected,
as.data.frame(head(mlp_test$net.result, -1)))
colnames(expected_v_test) <- c("Expected Output", "Neural Net
Output")
return(expected_v_test)
}
```

```
mlp.nn1_results <- compute(mlp.nn1, staggered_test_data_frame)

test_expected_data.nn1 <- tail(exchange.scaled_test$`USD/EUR`,
-3)

test_v_expected.nn1 <-
expected_v_test_func(test_expected_data.nn1, mlp.nn1_results)
head(test_v_expected.nn1)
```

```
## Expected Output Neural Net Output
## 1 0.21585168845 -0.30097304313
## 2 -0.09541973212 -0.33435905897
## 3 -0.10978610538 -0.11561401130
## 4 -0.16725159841 -0.03632470212
## 5 -0.30133774881 -0.20621919121
## 6 -0.26781621121 -0.29742400385
```

Evaluating the predictions

There are two ways we can look at the quality of the performance of the neural network: - Various single number values derived from the difference between the predicted and actual values - Visualising the data, for example, by plotting the estimates and actual values on the same graph to more easily draw comparisons by eye.

Numeric indicators

And then the Sum of Square Errors, and Mean Squared Error values can be derived in order to look at the performance of the trained neural network against test data. The closer to 0 the values are, the better (**Hyndman and Athanasopoulos, no date**).

```
output_delta_func <- function (expected, actual) {
return (c(expected-actual))
}
sse_func <- function(x) {
return (sum( (x - mean(x) )^2 ))
}
mse_func <- function (x) {
return(sse_func(x)/length(x))
}



test_delta.nn1 <-
output_delta_func(test_v_expected.nn1$`Expected Output`,
test_v_expected.nn1$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn1)
t_sse



## [1] 7.662076349



#MSE of this first nn
t_mse <- mse_func(test_delta.nn1)
t_mse



## [1] 0.1143593485



performance_scores <- list(nn1 = list(t_sse, t_mse))
```

Additionally we can look are the performance in terms of the Root Mean Square Error or the Mean Absolute Percentage Error. The Root Mean Square Error is the square root of the average of the squared errors, which allows for the greater individual errors to have a greater influence on the final error value. The Mean Absolute Percentage Error shows the error in terms of the the median of the errors as a fraction of the estimated value, which is displayed as a percentage; the biggest drawback to this measure is the risk of dividing by 0 is the estimated value is 0 but is doesn’t allow the biggest deltas to influence the final value more than by their proportions.

What follows are the RMSE and MAPE values for the neural net.

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn1$`Neural Net Output`,
test_v_expected.nn1$`Expected Output`)
t_rmse
```

```
## [1] 0.3418937515
```

```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn1$`Neural Net Output`,
test_v_expected.nn1$`Expected Output`))
t_mape
```

```
## [1] "23.2%"
```

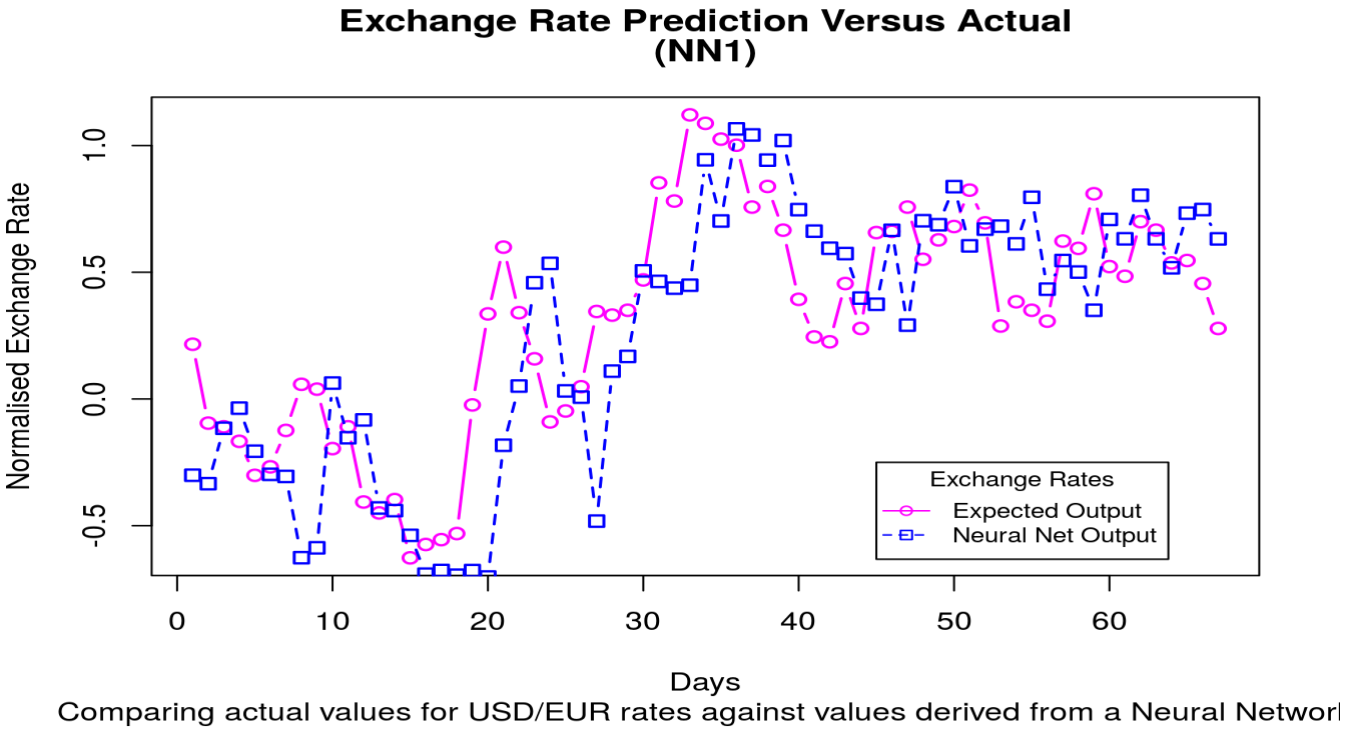
```
performance_scores$nn1 <- append(performance_scores$nn1,
c(t_rmse, t_mape))
```

Visual evaluation

That follows is a graph plotting neural net predicted values against actual (normalised/scaled) values.

```
line_compare_nn_to_actual <- function (compare_data, token_x,
suffix = "") {
count.col <- ncol(compare_data)

# get the range for the x and y axis
plot_range.y <- range(compare_data)
plot_range.x <- range(1:nrow(compare_data))

#print(c("plot_range.y:", plot_range.y, " - "))
#print(c("compare_data:", compare_data, " - "))

# set up the plot
plot(token_x, type="n", xlab="Days", ylab="Normalised Exchange
Rate")

colors <- c(6, 4)
```

```
linetype <- c(1:count.col)
plotchar <- seq(1, 0)

# add lines
for (i in 1:count.col) {
lines(compare_data[i], type="b", lwd=1.5,
lty=linetype[i], col=colors[i], pch=plotchar[i])
}

# add a title and subtitle
title(paste(c("Exchange Rate Prediction Versus Actual",
suffix)),
"Comparing actual values for USD/EUR rates against values
derived from a Neural Network")

return(list(linetype, plotchar, colors))
}



# Create Line Chart
linetypes <- line_compare_nn_to_actual(test_v_expected.nn1,
test_v_expected.nn1$`Expected Output`, suffix = "(NN1)")

# add a legend
legend(45, -.25, names(test_v_expected.nn1), cex=0.8,
col=linetypes[[3]], pch=linetypes[[2]], lty=linetypes[[1]],
title="Exchange Rates")
```



What's interesting about looking at it visually, is how it's more obvious from this perspective than it is from indicators values, that the neural net predictions are similar overall but with a latency of a few days; this isn't very useful for forecasting and might actually suggest a kind of over-fitting, where the network has been too tightly trained with the training data. I'm not certain that this is the case however because this graph is based on data that was not used for training purpose.

Experimenting with other Neural Network configurations

Now that we have one neural net with performance results, it's time to use this as a benchmark/baseline and create other neural networks to see if we can improve on performance.

Normalised data instead of scaled data

I'm just curious as to whether or not normalising the data between 0 and 1 would work better than between -1 and 1 around a mean of 0.

```
normalised_staggered_data_frame <-
vector_to_time_series_data(exchange.normalised_train$`USD/EUR`,
4)

colnames(normalised_staggered_data_frame) <- c("Input_dneg2",
"Input_dneg1", "Input_d0", "Output")

#Summary of training data
head(normalised_staggered_data_frame)
```

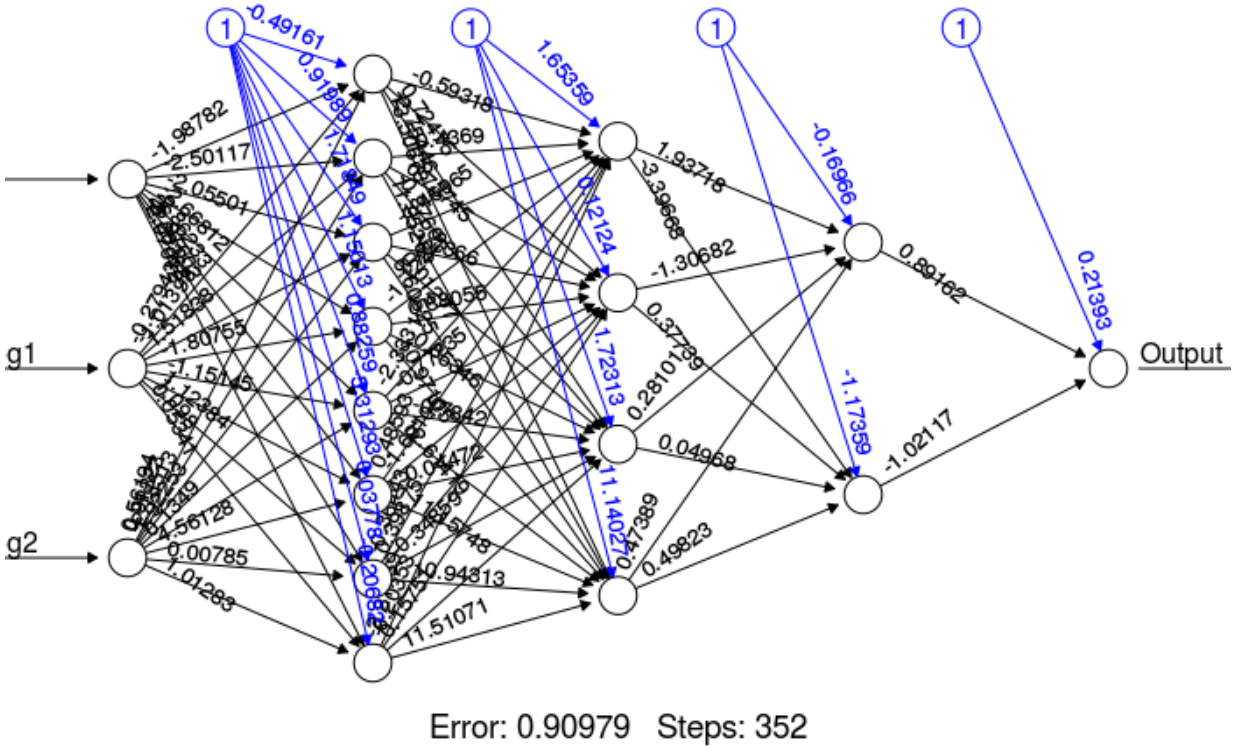
```
## Input_dneg2 Input_dneg1 Input_d0 Output
## 1 0.05899705015 0.2261553589 0.3598820059 0.3392330383
## 2 0.22615535890 0.3598820059 0.3392330383 0.4178957719
## 3 0.35988200590 0.3392330383 0.4178957719 0.3510324484
## 4 0.33923303835 0.4178957719 0.3510324484 0.3244837758
## 5 0.41789577188 0.3510324484 0.3244837758 0.2517207473
## 6 0.35103244838 0.3244837758 0.2517207473 0.1769911504
```

```
mlp.form1 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2")
mlp.nn1 <- neuralnet(mlp.form1, staggered_data_frame, hidden =
c(8,4,2), threshold=0.01)
```

```
## Warning: algorithm did not converge in 1 of 1 repetition(s)
within the
## stepmax
```

```
mlp.nn1_norm <- neuralnet(mlp.form1,
normalised_staggered_data_frame, hidden=c(8,4,2),
threshold=0.01)
```

```
plot(mlp.nn1_norm)
```



Error: 0.90979 Steps: 352

```
normalised_staggered_test_data_frame <-
vector_to_time_series_data(exchange.normalised_test$`USD/EUR`,
3)

colnames(normalised_staggered_test_data_frame) <-
c("Input_dneg2", "Input_dneg1", "Input_d0")
```

```
#Summary of training data
head(normalised_staggered_test_data_frame)
```

```
## Input_dneg2 Input_dneg1 Input_d0
## 1 0.4837758112 0.5467059980 0.4650934120
## 2 0.5467059980 0.4650934120 0.5732546706
## 3 0.4650934120 0.5732546706 0.5093411996
## 4 0.5732546706 0.5093411996 0.5063913471
## 5 0.5093411996 0.5063913471 0.4945919371
## 6 0.5063913471 0.4945919371 0.4670599803
```

```
mlp.nn1_norm_results <- compute(mlp.nn1_norm,
normalised_staggered_test_data_frame)

test_expected_data.nn1_norm <-
tail(exchange.normalised_test$`USD/EUR`, -3)

test_v_expected.nn1_norm <-
expected_v_test_func(test_expected_data.nn1_norm,
mlp.nn1_norm_results)
head(test_v_expected.nn1_norm)
```

```
## Expected Output Neural Net Output
## 1 0.5732546706 0.4945466833
## 2 0.5093411996 0.5464114087
## 3 0.5063913471 0.4739312502
## 4 0.4945919371 0.5814367810
## 5 0.4670599803 0.5156337810
## 6 0.4739429695 0.5135964219
```

```
test_delta.nn1_norm <-
output_delta_func(test_v_expected.nn1_norm$`Expected Output`,
test_v_expected.nn1_norm$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn1_norm)
t_sse
```

```
## [1] 0.3524394996
```

```
#MSE of this first nn
t_mse <- mse_func(test_delta.nn1_norm)
t_mse
```

```
## [1] 0.005260291039
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn1_norm$`Neural Net Output`,
test_v_expected.nn1_norm$`Expected Output`)
t_rmse
```

```
## [1] 0.07253368463
```

```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn1_norm$`Neural Net
Output`, test_v_expected.nn1_norm$`Expected Output`))
t_mape
```

```
## [1] "10.5%"
```

```
performance_scores <- append(performance_scores, list(nn1_norm
= list(t_sse, t_mse, t_rmse, t_mape)))
```

I'm not really sure what's going on with the line graph being plot for this but the figures show that the use of normalised data over scaled data is a marked improvement; the number of steps to train the network is faster, the accuracy is higher against test data; as such, I will make sure all other neural networks for this experiment will use the normalised data.

Alternative hidden layer structures

The hidden layers are the 'secret sauce' of this kind of neural network, the layers describe neurons consisting of nodes and vertices that connect to other neuron nodes; the vertices between nodes adjust the values as they are passed from node to node resulting in the answer at the end of the chain off layers. The art is to pick a good combination of hidden layers that doesn't over-fit to the training data and isn't so

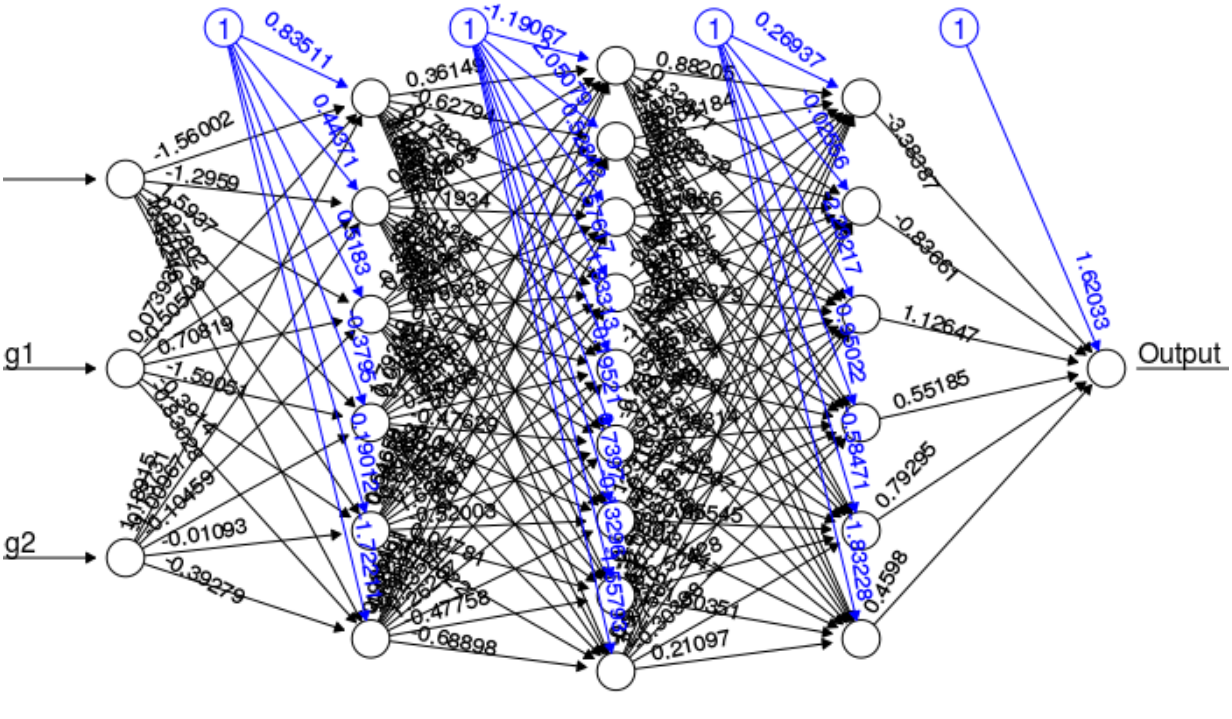
complicated it take a long time to train but also produces good accuracy results against test and validation data (**Heaton, 2017**). The first neural network had 3 hidden layers each powers of 2, starting with 8 and halving at each next layer.

The next step is to experiment with the same dataset and the same input values but experiment with the accuracy of some different hinder layer structures like:

1. 6, 9, 6
2. 6, 9, 6, 3
3. 12, 8, 4
4. 16, 9, 4

{6, 9, 6} hidden layer structure

```
#mlp.form1 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2")

mlp.nn_6_9_6 <- neuralnet(mlp.form1,
normalised_staggered_data_frame, hidden = c(6, 9, 6),
threshold=0.05, stepmax = 800000)


plot(mlp.nn_6_9_6)
```



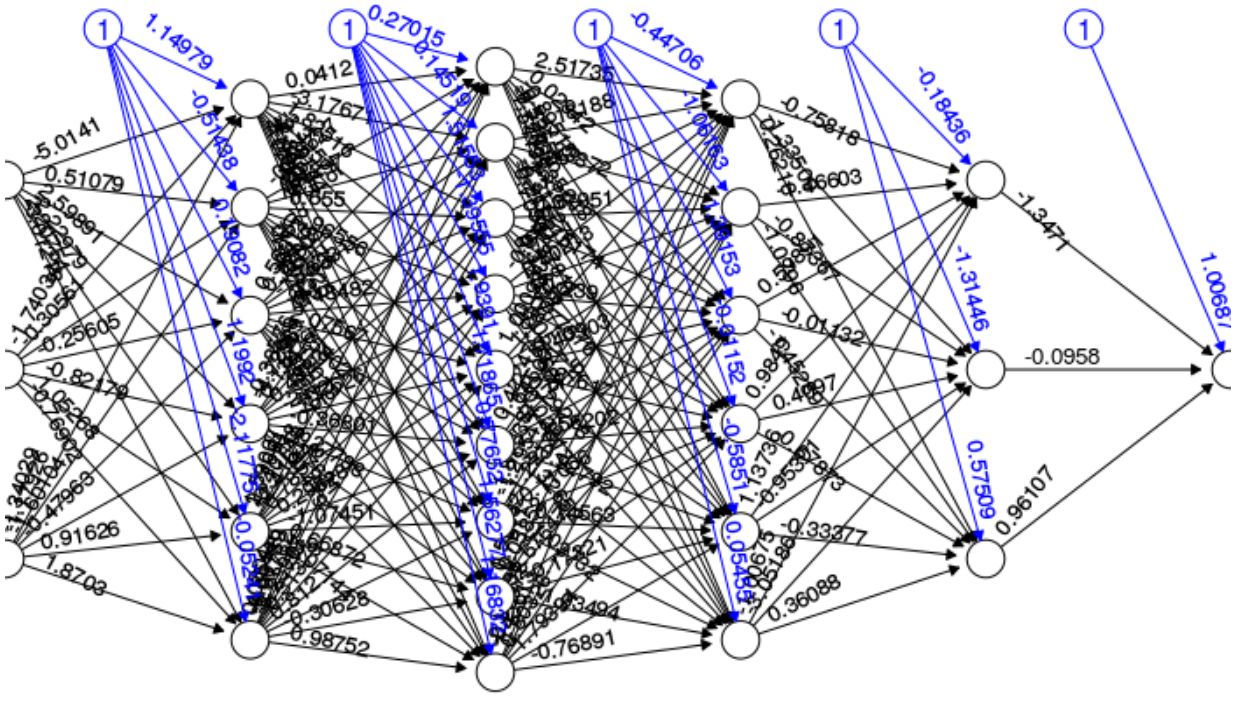
```
mlp.nn2_results <- compute(mlp.nn_6_9_6,
normalised_staggered_test_data_frame)

test_expected_data.nn2 <-
tail(exchange.normalised_test$`USD/EUR`, -3)

test_v_expected.nn2 <-
expected_v_test_func(test_expected_data.nn2, mlp.nn2_results)



test_delta.nn2 <-
output_delta_func(test_v_expected.nn2$`Expected Output`,
test_v_expected.nn2$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn2)
t_sse



## [1] 0.2993193903



#MSE of this first nn
t_mse <- mse_func(test_delta.nn2)
t_mse



## [1] 0.004467453587



#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn2$`Neural Net Output`,
test_v_expected.nn2$`Expected Output`)
t_rmse



## [1] 0.06683902229



#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn2$`Neural Net Output`,
test_v_expected.nn2$`Expected Output`))
t_mape
```

```
## [1] "9.54%"
```

```
performance_scores <- append(performance_scores, list(nn2 =
list(t_sse, t_mse, t_rmse, t_mape)))
```

{6, 9, 6, 3} hidden layer structure

```
#mlp.form1 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2")

mlp.nn_6_9_6_3 <- neuralnet(mlp.form1,
normalised_staggered_data_frame, hidden = c(6, 9, 6, 3),
threshold=0.05, stepmax = 1200000)


plot(mlp.nn_6_9_6_3)
```




```
mlp.nn3_results <- compute(mlp.nn_6_9_6_3,
normalised_staggered_test_data_frame)

test_expected_data.nn3 <-
tail(exchange.normalised_test$`USD/EUR`, -3)
```

```
test_v_expected.nn3 <-
expected_v_test_func(test_expected_data.nn3, mlp.nn3_results)



test_delta.nn3 <-
output_delta_func(test_v_expected.nn3$`Expected Output`,
test_v_expected.nn3$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn3)
t_sse
```

```
## [1] 0.3301664599
```

```
#MSE of this first nn
t_mse <- mse_func(test_delta.nn3)
t_mse
```

```
## [1] 0.00492785761
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn3$`Neural Net Output`,
test_v_expected.nn3$`Expected Output`)
t_rmse
```

```
## [1] 0.07021921885
```

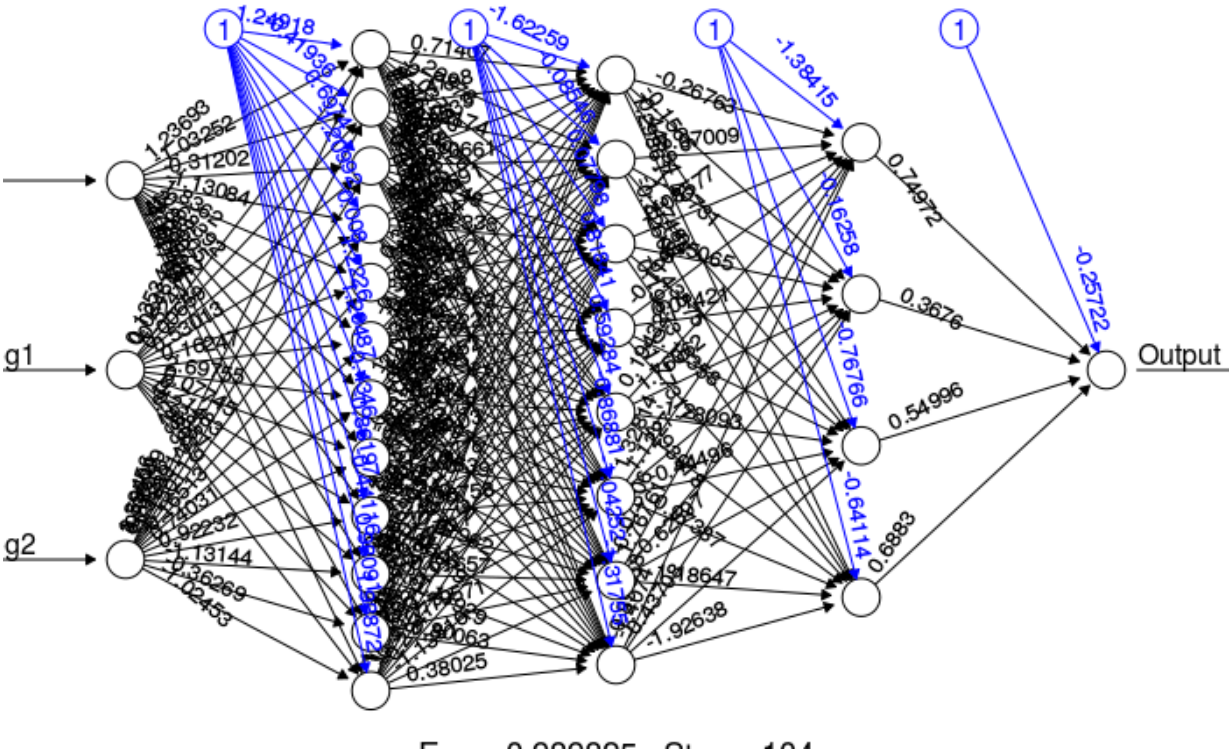
```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn3$`Neural Net Output`,
test_v_expected.nn3$`Expected Output`))
t_mape
```

```
## [1] "10.1%"
```

```
performance_scores <- append(performance_scores, list(nn3 =
list(t_sse, t_mse, t_rmse, t_mape)))
```

{12, 8, 4} hidden layer structure

```
#mlp.form1 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2")

mlp.nn_12_8_4 <- neuralnet(mlp.form1,
normalised_staggered_data_frame, hidden = c(12, 8, 4),
threshold=0.05, stepmax = 1200000)


plot(mlp.nn_12_8_4)
```




```
mlp.nn4_results <- compute(mlp.nn_12_8_4,
normalised_staggered_test_data_frame)

test_expected_data.nn4 <-
tail(exchange.normalised_test$`USD/EUR`, -3)

test_v_expected.nn4 <-
expected_v_test_func(test_expected_data.nn4, mlp.nn4_results)
```

```
test_delta.nn4 <-
output_delta_func(test_v_expected.nn4$`Expected Output`,
test_v_expected.nn4$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn4)
t_sse
```

```
## [1] 0.3275952559
```

```
#MSE of this first nn
t_mse <- mse_func(test_delta.nn4)
t_mse
```

```
## [1] 0.004889481432
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn4$`Neural Net Output`,
test_v_expected.nn4$`Expected Output`)
t_rmse
```

```
## [1] 0.06998631433
```

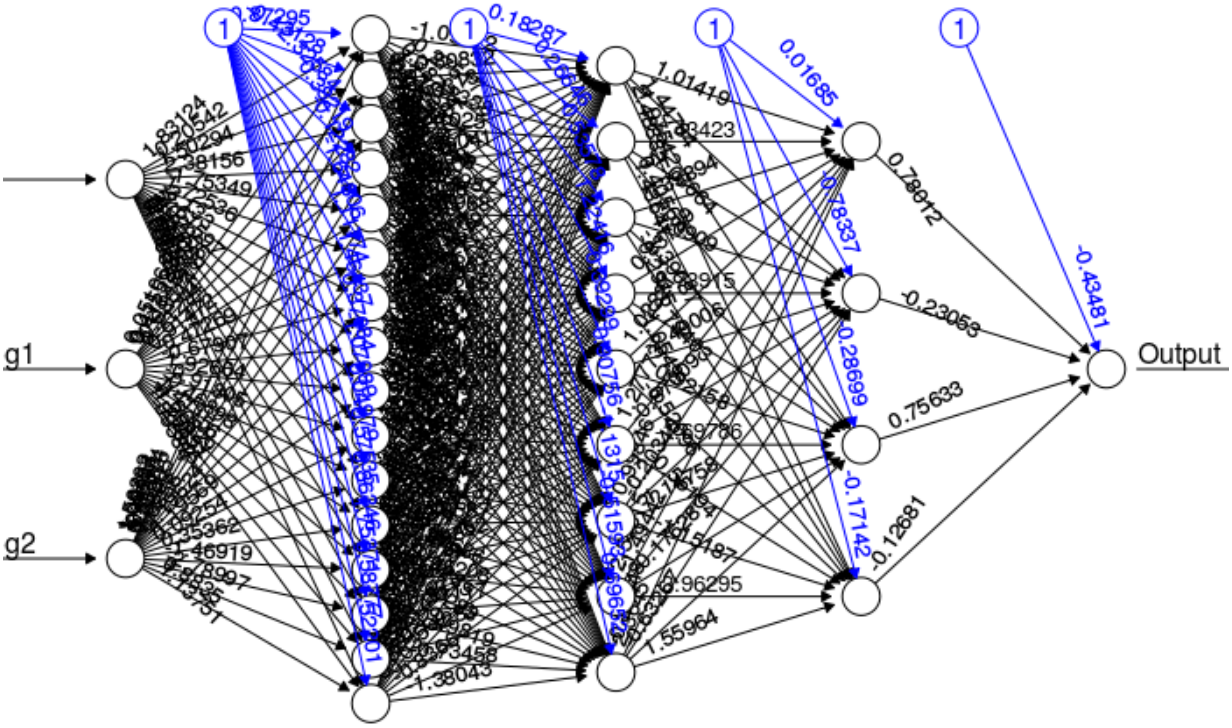
```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn4$`Neural Net Output`,
test_v_expected.nn4$`Expected Output`))
t_mape
```

```
## [1] "10%"
```

```
performance_scores <- append(performance_scores, list(nn4 =
list(t_sse, t_mse, t_rmse, t_mape)))
```

{16, 9, 4} hidden layer structure

```
#mlp.form1 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2")

mlp.nn_16_9_4 <- neuralnet(mlp.form1,
normalised_staggered_data_frame, hidden = c(16, 9, 4),
threshold=0.05, stepmax = 1200000)


plot(mlp.nn_16_9_4)
```



"/>

```
mlp.nn5_results <- compute(mlp.nn_16_9_4,
normalised_staggered_test_data_frame)

test_expected_data.nn5 <-
tail(exchange.normalised_test$`USD/EUR`, -3)

test_v_expected.nn5 <-
expected_v_test_func(test_expected_data.nn5, mlp.nn5_results)
```

```
test_delta.nn5 <-
output_delta_func(test_v_expected.nn5$`Expected Output`,
test_v_expected.nn5$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn5)
t_sse
```

```
## [1] 0.3515777048
```

```
#MSE of this first nn
t_mse <- mse_func(test_delta.nn5)
t_mse
```

```
## [1] 0.00524742843
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn5$`Neural Net Output`,
test_v_expected.nn5$`Expected Output`)
t_rmse
```

```
## [1] 0.07244232091
```

```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn5$`Neural Net Output`,
test_v_expected.nn5$`Expected Output`))
t_mape
```

```
## [1] "10.4%"
```

```
performance_scores <- append(performance_scores, list(nn5 =
list(t_sse, t_mse, t_rmse, t_mape)))
```

Performance Thus Far

```
# Display a table for performance so far
perf_dt_1 <- rbindlist(performance_scores)
perf_dt_1 <- cbind(names(performance_scores), perf_dt_1)
colnames(perf_dt_1) <- c("NN", "SSE", "MSE", "RMSE", "MAPE")
perf_dt_1
```

```
## NN SSE MSE RMSE MAPE
## 1: nn1 7.6620763490 0.114359348493 0.3418937515389 23.2%
## 2: nn1_norm 0.3524394996 0.005260291039 0.0725336846300639
10.5%
## 3: nn2 0.2993193903 0.004467453587 0.0668390222890017 9.54%
## 4: nn3 0.3301664599 0.004927857610 0.070219218845519 10.1%
## 5: nn4 0.3275952559 0.004889481432 0.0699863143262478 10%
## 6: nn5 0.3515777048 0.005247428430 0.0724423209098296 10.4%
```

Looking at the performance metrics so far, it would make sense to invest more time experimenting with neural networks with the following hidden layers: {6, 9, 6} (nn2), {12, 8, 4} (nn4); for these two structures, all the metrics outperform the other neural networks in terms of being closest to 0. What needs to be seen now is whether of not the accuracy of the networks can be improved by increasing or decreasing the number of inputs. It would make sense that accuracy should increase with greater number of inputs but for the sake of completeness, 2 inputs will be tried as well as 4.

Alternative Input Counts

In order to experiment with the inputs we need to first prepare the data.

Preparing the data for 2 inputs

```
normalised_staggered_2i_data_frame <-
vector_to_time_series_data(exchange.normalised_train$`USD/EUR`,
3)

colnames(normalised_staggered_2i_data_frame) <-
c("Input_dneg1", "Input_d0", "Output")

#Summary of training data
head(normalised_staggered_2i_data_frame)
```

```
## Input_dneg1 Input_d0 Output
## 1 0.05899705015 0.2261553589 0.3598820059
## 2 0.22615535890 0.3598820059 0.3392330383
## 3 0.35988200590 0.3392330383 0.4178957719
## 4 0.33923303835 0.4178957719 0.3510324484
## 5 0.41789577188 0.3510324484 0.3244837758
## 6 0.35103244838 0.3244837758 0.2517207473
```

```
normalised_staggered_2i_test_data_frame <-
vector_to_time_series_data(exchange.normalised_test$`USD/EUR`,
2)

colnames(normalised_staggered_2i_test_data_frame) <-
c("Input_dneg1", "Input_d0")

#Summary of training data
head(normalised_staggered_2i_test_data_frame)
```

```
## Input_dneg1 Input_d0
## 1 0.4837758112 0.5467059980
## 2 0.5467059980 0.4650934120
## 3 0.4650934120 0.5732546706
## 4 0.5732546706 0.5093411996
## 5 0.5093411996 0.5063913471
## 6 0.5063913471 0.4945919371
```

Preparing the data for 4 inputs

```
normalised_staggered_4i_data_frame <-
vector_to_time_series_data(exchange.normalised_train$`USD/EUR`,
5)

colnames(normalised_staggered_4i_data_frame) <-
c("Input_dneg3", "Input_dneg2", "Input_dneg1", "Input_d0",
"Output")

#Summary of training data
head(normalised_staggered_4i_data_frame)
```

```
## Input_dneg3 Input_dneg2 Input_dneg1 Input_d0 Output
## 1 0.05899705015 0.2261553589 0.3598820059 0.3392330383
0.4178957719
## 2 0.22615535890 0.3598820059 0.3392330383 0.4178957719
0.3510324484
## 3 0.35988200590 0.3392330383 0.4178957719 0.3510324484
0.3244837758
## 4 0.33923303835 0.4178957719 0.3510324484 0.3244837758
0.2517207473
## 5 0.41789577188 0.3510324484 0.3244837758 0.2517207473
0.1769911504
## 6 0.35103244838 0.3244837758 0.2517207473 0.1769911504
0.2025565388
```

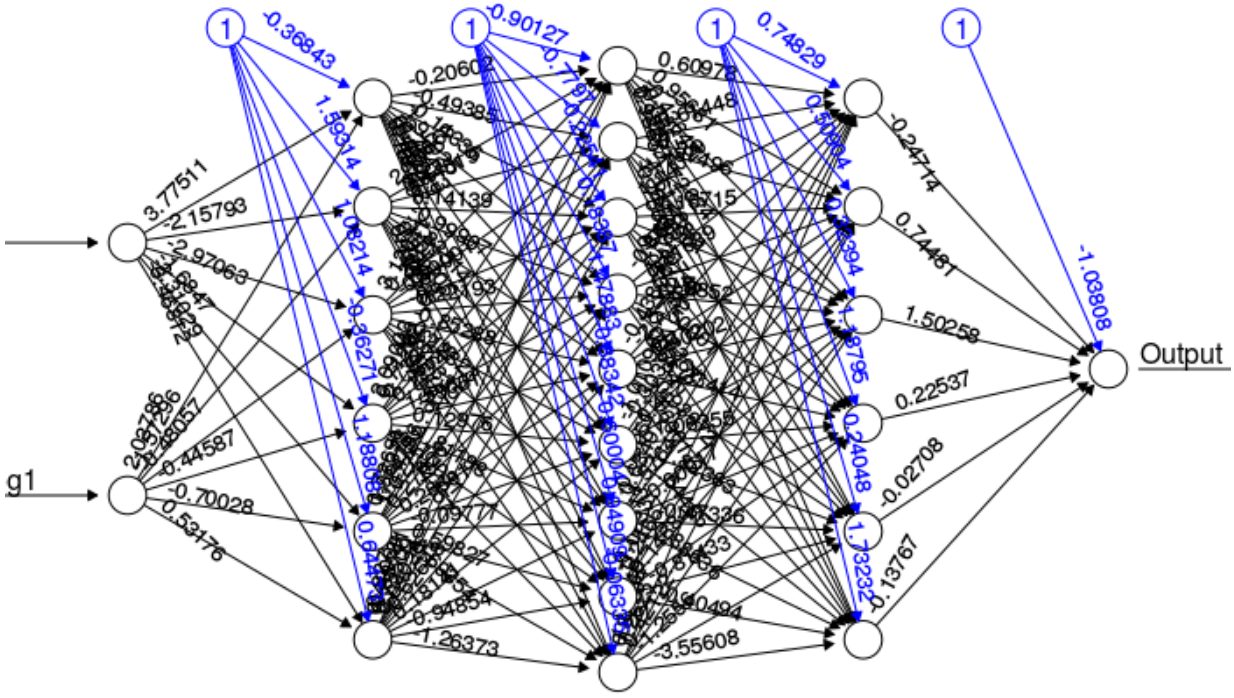
```
normalised_staggered_4i_test_data_frame <-
vector_to_time_series_data(exchange.normalised_test$`USD/EUR`,
4)

colnames(normalised_staggered_4i_test_data_frame) <-
c("Input_dneg3", "Input_dneg2", "Input_dneg1", "Input_d0")

#Summary of training data
head(normalised_staggered_4i_test_data_frame)
```

```
## Input_dneg3 Input_dneg2 Input_dneg1 Input_d0
## 1 0.4837758112 0.5467059980 0.4650934120 0.5732546706
## 2 0.5467059980 0.4650934120 0.5732546706 0.5093411996
## 3 0.4650934120 0.5732546706 0.5093411996 0.5063913471
## 4 0.5732546706 0.5093411996 0.5063913471 0.4945919371
## 5 0.5093411996 0.5063913471 0.4945919371 0.4670599803
## 6 0.5063913471 0.4945919371 0.4670599803 0.4739429695
```

{6, 9, 6} hidden layer structure with 2 inputs

```
mlp.form2 <- as.formula("Output ~ Input_d0 + Input_dneg1")

mlp.nn_6_9_6_2i <- neuralnet(mlp.form2,
normalised_staggered_2i_data_frame, hidden = c(6, 9, 6),
threshold=0.05, stepmax = 800000)
```

```
plot(mlp.nn_6_9_6_2i)
```




```
mlp.nn2_2i_results <- compute(mlp.nn_6_9_6_2i,
normalised_staggered_2i_test_data_frame)

test_expected_data.nn2_2i <-
tail(exchange.normalised_test$`USD/EUR`, -2)

test_v_expected.nn2_2i <-
expected_v_test_func(test_expected_data.nn2_2i,
mlp.nn2_2i_results)



test_delta.nn2_2i <-
output_delta_func(test_v_expected.nn2_2i$`Expected Output`,
test_v_expected.nn2_2i$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn2_2i)
t_sse



## [1] 0.1791759101
```

```
#MSE of this first nn
t_mse <- mse_func(test_delta.nn2_2i)
t_mse
```

```
## [1] 0.002634939854
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn2_2i$`Neural Net Output`,
test_v_expected.nn2_2i$`Expected Output`)
t_rmse
```

```
## [1] 0.05153763915
```

```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn2_2i$`Neural Net
Output`, test_v_expected.nn2_2i$`Expected Output`))
t_mape
```

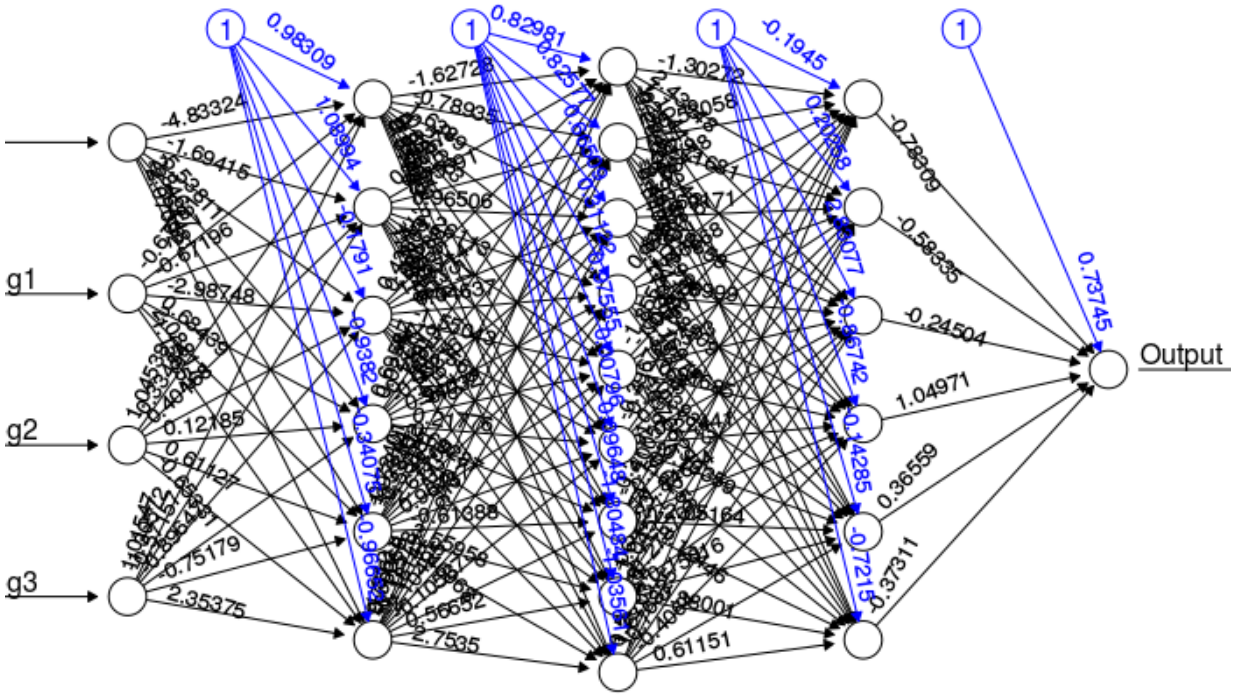
```
## [1] "7.26%"
```

```
performance_scores <- append(performance_scores, list(nn2_2i =
list(t_sse, t_mse, t_rmse, t_mape)))
```

{6, 9, 6} hidden layer structure with 4 inputs

```
mlp.form4 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2 + Input_dneg3")

mlp.nn_6_9_6_4i <- neuralnet(mlp.form4,
normalised_staggered_4i_data_frame, hidden = c(6, 9, 6),
threshold=0.05, stepmax = 800000)
```

```
plot(mlp.nn_6_9_6_4i)
```




```
mlp.nn2_4i_results <- compute(mlp.nn_6_9_6_4i,
normalised_staggered_4i_test_data_frame)

test_expected_data.nn2_4i <-
tail(exchange.normalised_test$`USD/EUR`, -4)

test_v_expected.nn2_4i <-
expected_v_test_func(test_expected_data.nn2_4i,
mlp.nn2_4i_results)



test_delta.nn2_4i <-
output_delta_func(test_v_expected.nn2_4i$`Expected Output`,
test_v_expected.nn2_4i$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn2_4i)
t_sse



## [1] 0.4307259632



#MSE of this first nn
t_mse <- mse_func(test_delta.nn2_4i)
t_mse
```

```
## [1] 0.006526150957
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn2_4i$`Neural Net Output`,
test_v_expected.nn2_4i$`Expected Output`)
t_rmse
```

```
## [1] 0.08079409715
```

```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn2_4i$`Neural Net
Output`, test_v_expected.nn2_4i$`Expected Output`))
t_mape
```

```
## [1] "11.6%"
```

```
performance_scores <- append(performance_scores, list(nn2_4i =
list(t_sse, t_mse, t_rmse, t_mape)))
```

Performance After Input Experimentation

```
# Display a table for performance so far
perf_dt_2 <- rbindlist(performance_scores)
perf_dt_2 <- cbind(names(performance_scores), perf_dt_2)
colnames(perf_dt_2) <- c("NN", "SSE", "MSE", "RMSE", "MAPE")
perf_dt_2
```

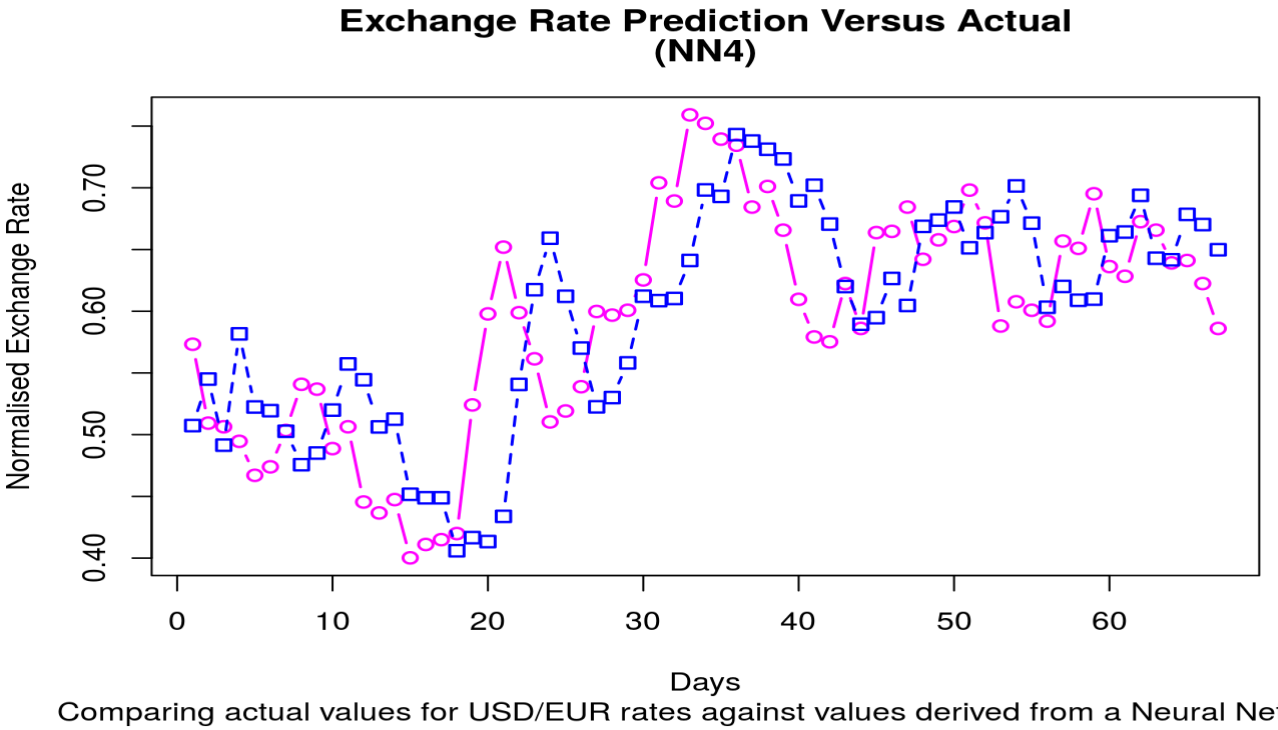
```
## NN SSE MSE RMSE MAPE
## 1: nn1 7.6620763490 0.114359348493 0.3418937515389 23.2%
## 2: nn1_norm 0.3524394996 0.005260291039 0.0725336846300639
10.5%
## 3: nn2 0.2993193903 0.004467453587 0.0668390222890017 9.54%
## 4: nn3 0.3301664599 0.004927857610 0.070219218845519 10.1%
## 5: nn4 0.3275952559 0.004889481432 0.0699863143262478 10%
## 6: nn5 0.3515777048 0.005247428430 0.0724423209098296 10.4%
```

```
## 7: nn2_2i 0.1791759101 0.002634939854 0.051537639147657
7.26%
## 8: nn2_4i 0.4307259632 0.006526150957 0.0807940971529594
11.6%
```

A closer look at the better performers

The following is a graph looking at the {12, 8, 4} hidden layer neural network, also labelled ‘nn4’; this was the second best outcome of the neural networks that used 3 inputs.

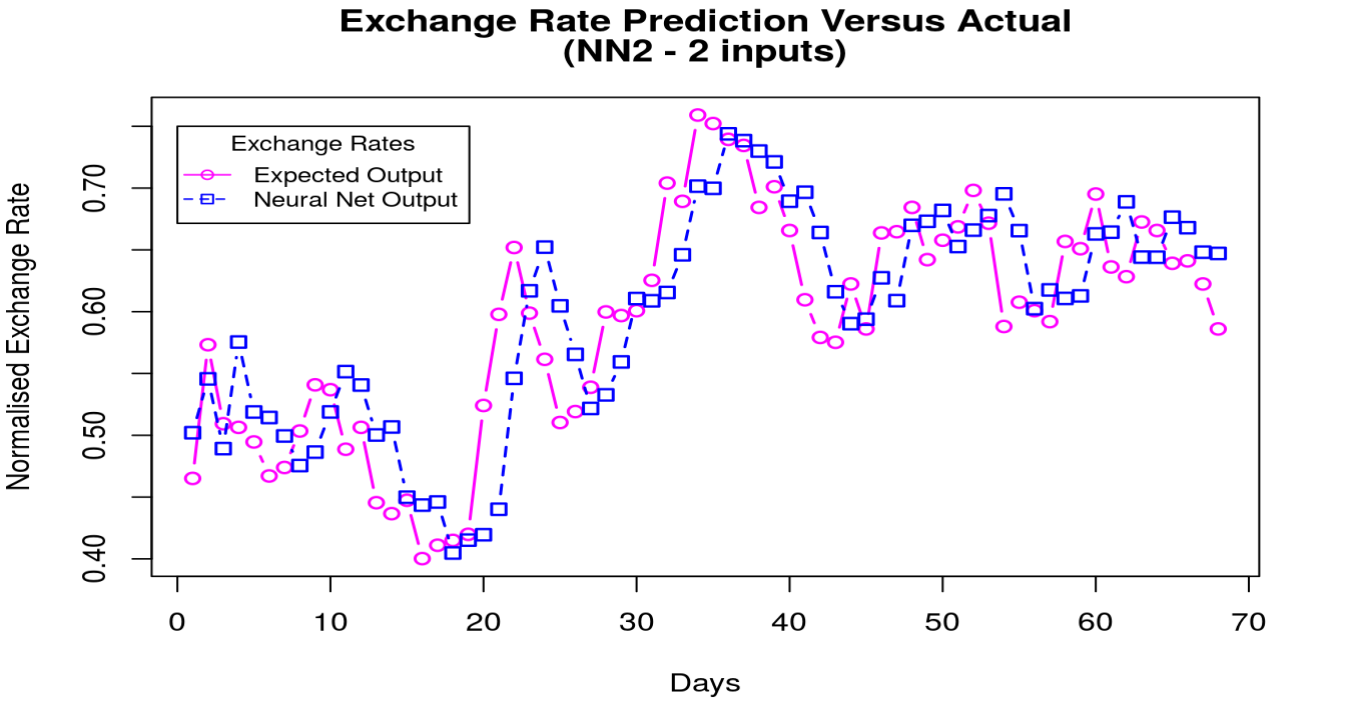
```
# Create Line Chart
linetypes <- line_compare_nn_to_actual(test_v_expected.nn4,
test_v_expected.nn4$`Expected Output`, suffix = "(NN4)")

# add a legend
legend(45, -.25, names(test_v_expected.nn4), cex=0.8,
col=linetypes[[3]], pch=linetypes[[2]], lty=linetypes[[1]],
title="Exchange Rates")
```



Comparing actual values for USD/EUR rates against values derived from a Neural Networ

The following is a graph looking at the {6, 9, 6} hidden layer neural network where the number of inputs was reduced to 2, also labelled ‘nn2_2i’; this was the second best outcome of the neural networks that used 3 inputs.

```
# Create Line Chart
linetypes <- line_compare_nn_to_actual(test_v_expected.nn2_2i,
test_v_expected.nn2_2i$`Expected Output`, suffix = "(NN2 - 2
inputs)")

# add a legend
legend(0, .75, names(test_v_expected.nn2_2i), cex=0.8,
col=linetypes[[3]], pch=linetypes[[2]], lty=linetypes[[1]],
title="Exchange Rates")
```



Comparing actual values for USD/EUR rates against values derived from a Neural Networ

The graphs tell a truth that the indicator values do not; there seems to be some serious over-fitting going on. The neural network results look like that they're mostly predicting based on the value of the previous day which is not really going to be much use in terms of real forecasting. There are are a few more courses of action we cane take to see if we can get better results:

- Look into tweaking other variables for the Neural Network behaviour
- Specifically the learning algorithm and learning rate, like using 'rprop-' instead of 'rprop+'
- Look into feeding in different data
- Like increasing the interval of days between input values
- Using a different Neural Network library to see if it's more suitable for time-series data

More experimentation

Adjusting threshold and algorithm

The next experiment is to drastically increase the threshold tolerance and choose a variation of the back propagation algorithm, ‘rprop-’.

```
#mlp.form1 <- as.formula("Output ~ Input_d0 + Input_dneg1 +
Input_dneg2")

mlp.nn_12_8_4_t0p5 <- neuralnet(mlp.form1,
normalised_staggered_data_frame, hidden = c(12, 8, 4),
threshold=0.5, algorithm = "rprop-")



mlp.nn4_t0p1_results <- compute(mlp.nn_12_8_4_t0p5,
normalised_staggered_test_data_frame)

test_expected_data.nn4_t0p1 <-
tail(exchange.normalised_test$`USD/EUR`, -3)

test_v_expected.nn4_t0p5 <-
expected_v_test_func(test_expected_data.nn4_t0p1,
mlp.nn4_t0p1_results)



test_delta.nn4_t0p5 <-
output_delta_func(test_v_expected.nn4_t0p5$`Expected Output`,
test_v_expected.nn4_t0p5$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn4_t0p5)
t_sse



## [1] 0.1732182494



#MSE of this first nn
t_mse <- mse_func(test_delta.nn4_t0p5)
t_mse
```

```
## [1] 0.002585347006
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn4_t0p5$`Neural Net Output`,
test_v_expected.nn4_t0p5$`Expected Output`)
t_rmse
```

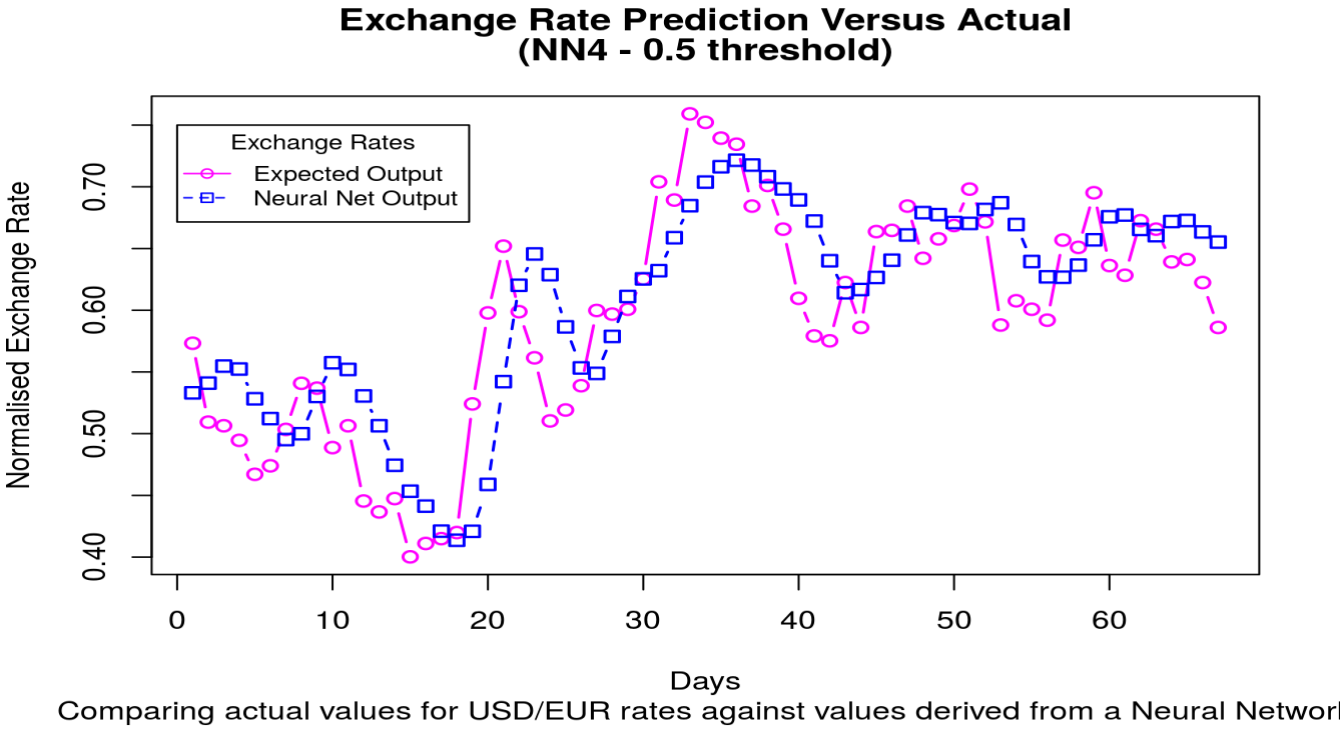
```
## [1] 0.05239438595
```

```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn4_t0p5$`Neural Net
Output`, test_v_expected.nn4_t0p5$`Expected Output`))
t_mape
```

```
## [1] "7.27%"
```

```
#performance_scores <- append(performance_scores, list(nn4 =
list(t_sse, t_mse, t_rmse, t_mape)))
```

```
# Create Line Chart
linetypes <-
line_compare_nn_to_actual(test_v_expected.nn4_t0p5,
test_v_expected.nn4_t0p5$`Expected Output`, suffix = "(NN4 -
0.5 threshold)")

# add a legend
legend(0, .75, names(test_v_expected.nn4_t0p5), cex=0.8,
col=linetypes[[3]], pch=linetypes[[2]], lty=linetypes[[1]],
title="Exchange Rates")
```



Comparing actual values for USD/EUR rates against values derived from a Neural Networ

Lessening the threshold for training actually allows the neural network to not fixate so much on the value but instead the trends to this graph demonstrates better performance but it's still not achieving the sort of results we'd hope for.

Adjusting input training data

The next experiment is to see if giving the Neural Network broader data it might better predict the expected value; in this case training with days that are a day apart.

Here follows the training data:

```
normalised_staggered_2day_data_frame <-
vector_to_time_series_data(exchange.normalised_train$`USD/EUR`,
4, step=2)

colnames(normalised_staggered_2day_data_frame) <-
c("Input_dneg6", "Input_dneg4", "Input_dneg2", "Output")

#Summary of training data
head(normalised_staggered_2day_data_frame)
```

```
## Input_dneg6 Input_dneg4 Input_dneg2 Output
## 1 0.05899705015 0.3598820059 0.4178957719 0.3244837758
## 2 0.22615535890 0.3392330383 0.3510324484 0.2517207473
```

```
## 3 0.35988200590 0.4178957719 0.3244837758 0.1769911504
## 4 0.33923303835 0.3510324484 0.2517207473 0.2025565388
## 5 0.41789577188 0.3244837758 0.1769911504 0.3097345133
## 6 0.35103244838 0.2517207473 0.2025565388 0.4375614553
```

Here follows the test data:

```
normalised_staggered_test_2day_data_frame <-
vector_to_time_series_data(exchange.normalised_test$`USD/EUR`,
3, step = 2)

colnames(normalised_staggered_test_2day_data_frame) <-
c("Input_dneg6", "Input_dneg4", "Input_dneg2")

#Summary of training data
head(normalised_staggered_test_2day_data_frame)
```

```
## Input_dneg6 Input_dneg4 Input_dneg2
## 1 0.4837758112 0.4650934120 0.5093411996
## 2 0.5467059980 0.5732546706 0.5063913471
## 3 0.4650934120 0.5093411996 0.4945919371
## 4 0.5732546706 0.5063913471 0.4670599803
## 5 0.5093411996 0.4945919371 0.4739429695
## 6 0.5063913471 0.4670599803 0.5034414946
```

```
mlp.form_2d <- as.formula("Output ~ Input_dneg2 + Input_dneg4 +
Input_dneg6")

mlp.nn_12_8_4_3id2 <- neuralnet(mlp.form_2d,
normalised_staggered_2day_data_frame, hidden = c(12, 8, 4),
threshold=0.5, algorithm = "rprop-")
```

```
mlp.nn4_3id2_results <- compute(mlp.nn_12_8_4_3id2,
normalised_staggered_test_2day_data_frame)

test_expected_data.nn4_3id2 <-
tail(exchange.normalised_test$`USD/EUR`, -5)

test_v_expected.nn4_3id2 <-
expected_v_test_func(test_expected_data.nn4_3id2,
mlp.nn4_3id2_results)
```

```
test_delta.nn4_3id2 <-
output_delta_func(test_v_expected.nn4_3id2$`Expected Output`,
test_v_expected.nn4_3id2$`Neural Net Output`)
#SSE of this first nn
t_sse <- sse_func(test_delta.nn4_3id2)
t_sse
```

```
## [1] 0.2013411963
```

```
#MSE of this first nn
t_mse <- mse_func(test_delta.nn4_3id2)
t_mse
```

```
## [1] 0.003097556867
```

```
#RMSE for the nn
t_rmse <- rmse(test_v_expected.nn4_3id2$`Neural Net Output`,
test_v_expected.nn4_3id2$`Expected Output`)
t_rmse
```

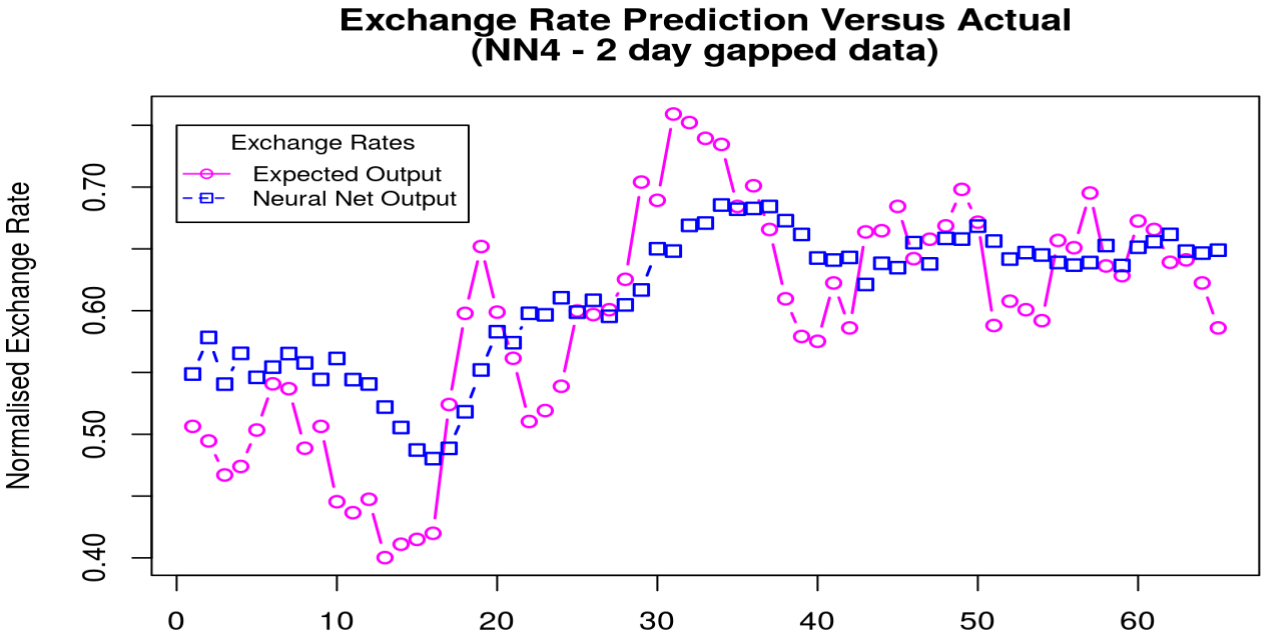
```
## [1] 0.05793260954
```

```
#MAPE for the nn
t_mape <- percent(mape(test_v_expected.nn4_3id2$`Neural Net
Output`, test_v_expected.nn4_3id2$`Expected Output`))
t_mape
```

```
## [1] "8.08%"
```

```
#performance_scores <- append(performance_scores, list(nn4 =
list(t_sse, t_mse, t_rmse, t_mape)))
```

```
# Create Line Chart
linetypes <-
line_compare_nn_to_actual(test_v_expected.nn4_3id2,
test_v_expected.nn4_3id2$`Expected Output`, suffix = "(NN4 - 2
day gapped data)")

# add a legend
legend(0, .75, names(test_v_expected.nn4_3id2), cex=0.8,
col=linetypes[[3]], pch=linetypes[[2]], lty=linetypes[[1]],
title="Exchange Rates")
```



Comparing actual values for USD/EUR rates against values derived from a Neural Networ

Adjusting the input data to be over a spread of 6 days rather than 3 has not create the results I had been hoping for; my idea was that the neural network my find a more general trend of a longer time period but instead it seems to just lag behind even further; it's as though whatever is the most recent input value passed into the neural network is what then is the main determinant of the predicted value, without any kind of trajectory of extrapolation.

Conclusion

I think that further research into other Neural Network libraries would be the first course of action I would take next, for example Caret and Nnet. The Neural network with the performance that pleased me the most was labelled `nn4_t0p5`, having a {12, 8, 4} hidden layer structure and a 0.5 accuracy threshold; this happens to have delivered best

results (MAPE: 7.72%) when you take into account it wasn’t so tightly over-fit to the data; I’d like to see how it would perform with more test data but I think another type of neural network might perform better.

One thing I didn’t do, was to use any smoothing methods on the data to take into account noise and seasonality; if I had time, I would research how to do this and apply it to the training data to see if that provided any benefit, as I have a feeling it may well have! Another tool I would look into would be the auto-regressive moving average, which looks like another way of looking at recent trends as the time series moves along; perhaps even adding extra inputs that are my own derived values based on input data might help nudge the performance in the right direction; I won’t know without trying.

Question 4: Forecasting (SVR)

Premise

> You need to construct a SVM model to address this forecasting problem. You need to consider the appropriate input vector. Write a code in R Studio to implement this SVR scheme. You need to show the performance of your model both graphically as well as in terms of usual statistical indices (MSE, RMSE and MAPE). Hind: The input selection problem is very important. Experiment with various options. Show all your working steps. As everyone will have different forecasting result, emphasis in the marking scheme will be given to the adopted methodology and the explanation/justification of various decisions you have taken in order to provide an acceptable, in terms of performance, solution.

Unfortunately this segment could not be completed.

References

Alice, M. (2015). *Fitting a neural network in r; neuralnet package* [].Available from https://datascienceplus.com/fitting-neural-network-in-r/ (https://datascienceplus.com/fitting-neural-network-in-r/).

Boateng, N. (2017). *Some clustering algorithms * [].Available from https://rstudio-pubs-static.s3.amazonaws.com/320180_c39d043794c349938dc4f76d91e70d2f.html (https://rstudio-pubs-static.s3.amazonaws.com/320180_c39d043794c349938dc4f76d91e70d2f.html).

Comparing dendrograms: Essentials - articles - sthda (2017).Available from http://www.sthda.com/english/articles/28-hierarchical-clustering-essentials/91-comparing-dendrograms-essentials/ (http://www.sthda.com/english/articles/28-hierarchical-clustering-essentials/91-comparing-dendrograms-essentials/).

Galili, T. (2017). *Hierarchical cluster analysis on famous data sets - enhanced with the dendextend package* [].Available from https://cran.r-project.org/web/packages/dendextend/vignettes/Cluster_Analysis.html (https://cran.r-project.org/web/packages/dendextend/vignettes/Cluster_Analysis.html).

Grogan, M. (2017). *Neuralnet: Train and test neural networks using r* [].Available from http://www.michaeljgrogan.com/neural-network-modelling-neuralnet-r/ (http://www.michaeljgrogan.com/neural-network-modelling-neuralnet-r/).

Heaton, J. (2017). *The number of hidden layers* [].Available from http://www.heatonresearch.com/2017/06/01/hidden-layers.html (http://www.heatonresearch.com/2017/06/01/hidden-layers.html).

Hyndman, R. and Athanasopoulos, G. (no date). *2.5 evaluating forecast accuracy* [].Available from https://www.otexts.org/fpp/2/5 (https://www.otexts.org/fpp/2/5).

Kaastra, I. and Boyd, M. (1996). *Designing a neural network for forecasting financial and economic time series*.

Kourentzes, N. (2017). *Forecasting time series with neural networks in r – nikolaos kourentzes* [].Available from http://kourentzes.com/forecasting/2017/02/10/forecasting-time-series-with-neural-networks-in-r/

(http://kourentzes.com/forecasting/2017/02/10/forecasting-time-series-with-neural-networks-in-r/).

Kulma, K. (2017). *Determining optimal number of clusters in your data* [].Available from http://kkulma.github.io/2017-04-24-determining-optimal-number-of-clusters-in-your-data/ (http://kkulma.github.io/2017-04-24-determining-optimal-number-of-clusters-in-your-data/).

Neural networks in r tutorial – learn by marketing (no date).Available from http://www.learnbymarketing.com/tutorials/neural-networks-in-r-tutorial/ (http://www.learnbymarketing.com/tutorials/neural-networks-in-r-tutorial/).

Prabhakaran, S. (2016). *Outlier detection and treatment with r* [].Available from https://datascienceplus.com/outlier-detection-and-treatment-with-r/ (https://datascienceplus.com/outlier-detection-and-treatment-with-r/).

Rand, W.M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66 (336), 846–850.Available from http://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356 (http://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356).

Using k-means to cluster wine dataset (2015).Available from https://datayo.wordpress.com/2015/05/06/using-k-means-to-cluster-wine-dataset/ (https://datayo.wordpress.com/2015/05/06/using-k-means-to-cluster-wine-dataset/).

Zhang, Z. (2016). Neural networks: Further insights into error function, generalized weights and others. *Annals of translational medicine*, 4 (16), 300.Available from http://www.ncbi.nlm.nih.gov/pubmed/27668220 (http://www.ncbi.nlm.nih.gov/pubmed/27668220).