INFORMATICS
INSTITUTE OF
TECHNOLOGY

UNIVERSITY OF
WESTMINSTER

INFORMATION INSTITUTE OF TECHNOLOGY

Department of computing (B.Sc.) in Computer Science

Module: 5DATA001C.2

Machine Learning and Data Mining

Module Leader: Mr. Achala Aponso

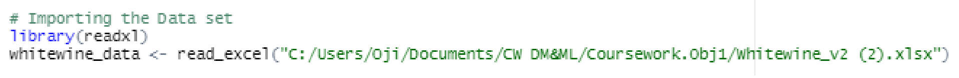
Course Work

1st Objective

Dataset

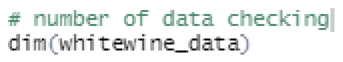
In this study, we examine a set of observations on different white wine types, including chemical attributes and taster ranks. There is only one dataset (whitewine v2.xls) with 4710 white wine varietals. Each wine is produced in a distinct area of Portugal. Data is collected on 12 distinct wine attributes, one of which is Quality (i.e., the very last column), which is based on sensory data, while the others are chemical properties including density, acidity, and alcohol concentration. The chemical characteristics of wine are all continuous variables. The ordinal variable quality has a range of 1 to 10 (worst to greatest) (best). Three separate tasters evaluate each wine variety, with the final ranking decided by the median score of the tasters.

The dataset consists of 4710 wine samples and includes 11 input variables and one outcome variable (quality). There are four diverse levels of quality.

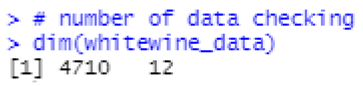


```
# Importing the Data set
library(readxl)
whitewine_data <- read_excel("C:/Users/Oji/Documents/CW DM&ML/Coursework.Obj1/Whitewine_v2 (2).xlsx")
```

As shown above we must first load the 'readxl' package into our R project in order to read xml files. The diagram below illustrates how we may do this. We can check how much data is in our dataset by executing the code. Shown as Below

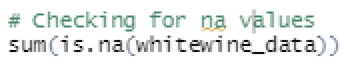


```
# number of data checking
dim(whitewine_data)
```

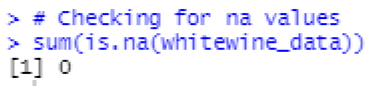


```
> # number of data checking
> dim(whitewine_data)
[1] 4710   12
```

Removing Outlier



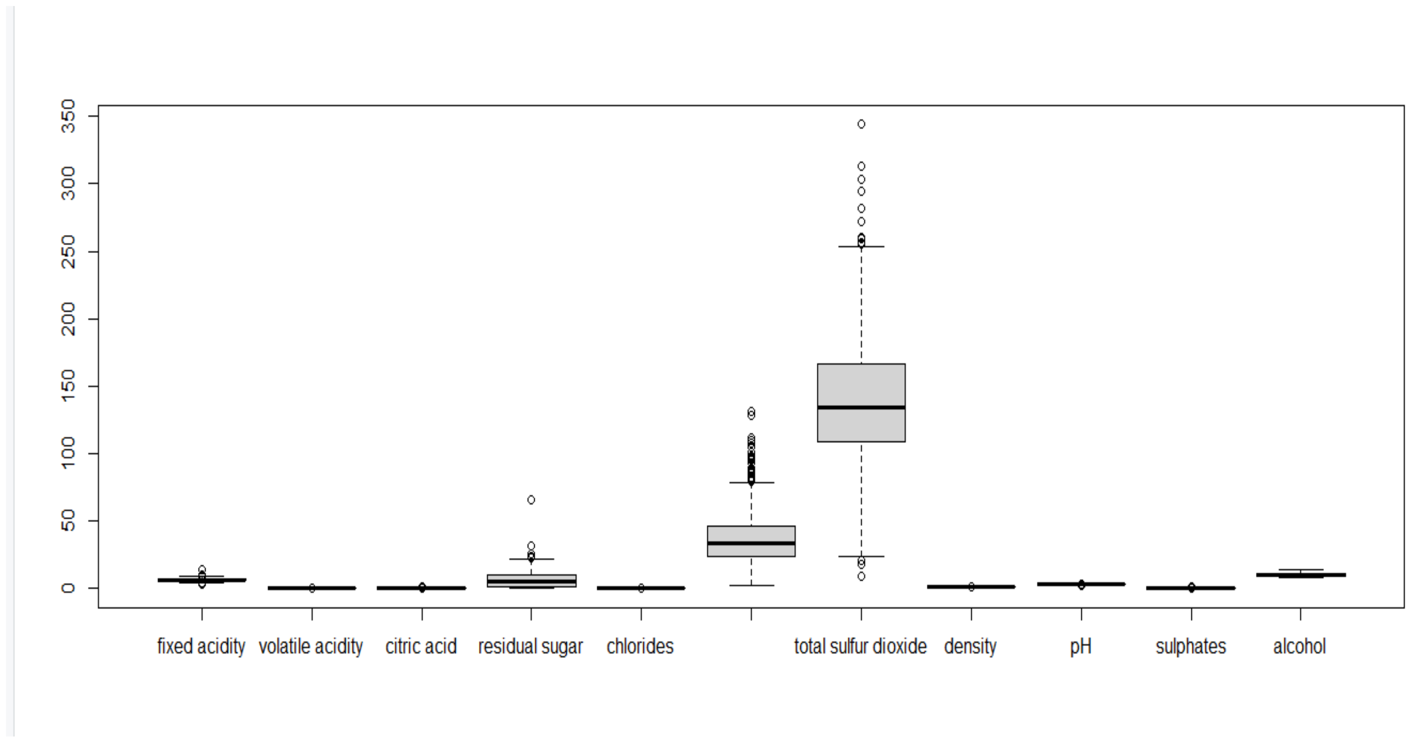
```
# Checking for na values
sum(is.na(whitewine_data))
```



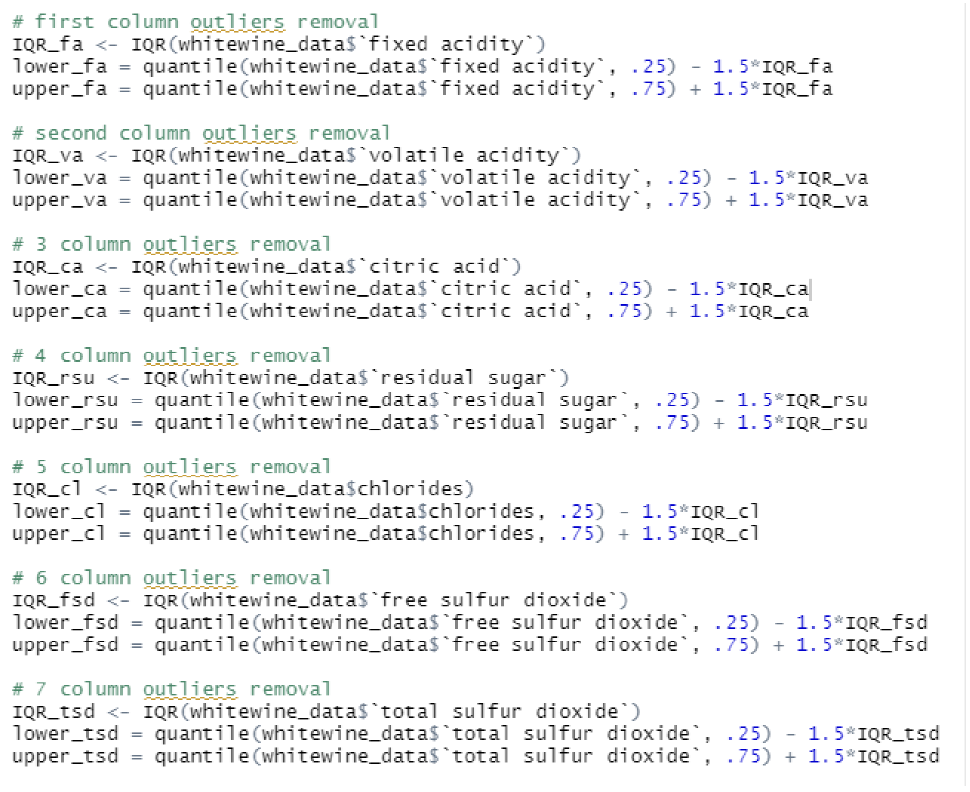
```
> # Checking for na values
> sum(is.na(whitewine_data))
[1] 0
```

Getting rid of NA values is crucial; else, our R project's output would suffer. We may see if our data collection has any NA values by executing the following code. There are no NA values in this project, as displayed.

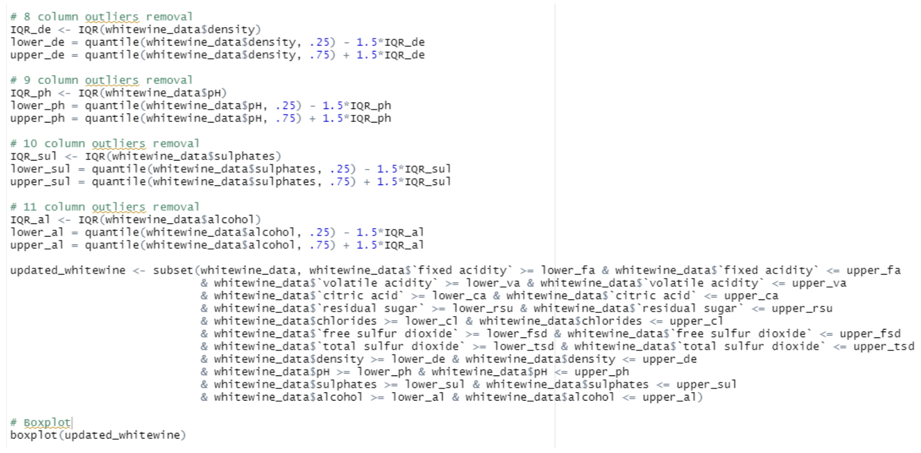
Then you must see whether there are any outlier values. If there are any outliers, we must eliminate them before using Kmean clustering. Outliers raise the variability of our data, lowering statistical power. As a result, eliminating outliers might make our findings statistically significant. The outliers may be identified using the following set of codes and the results of those codes. There are outliers in our data set, as shown in the graph below.



By running the following set codes it’s possible to remove the outliers of our dataset.

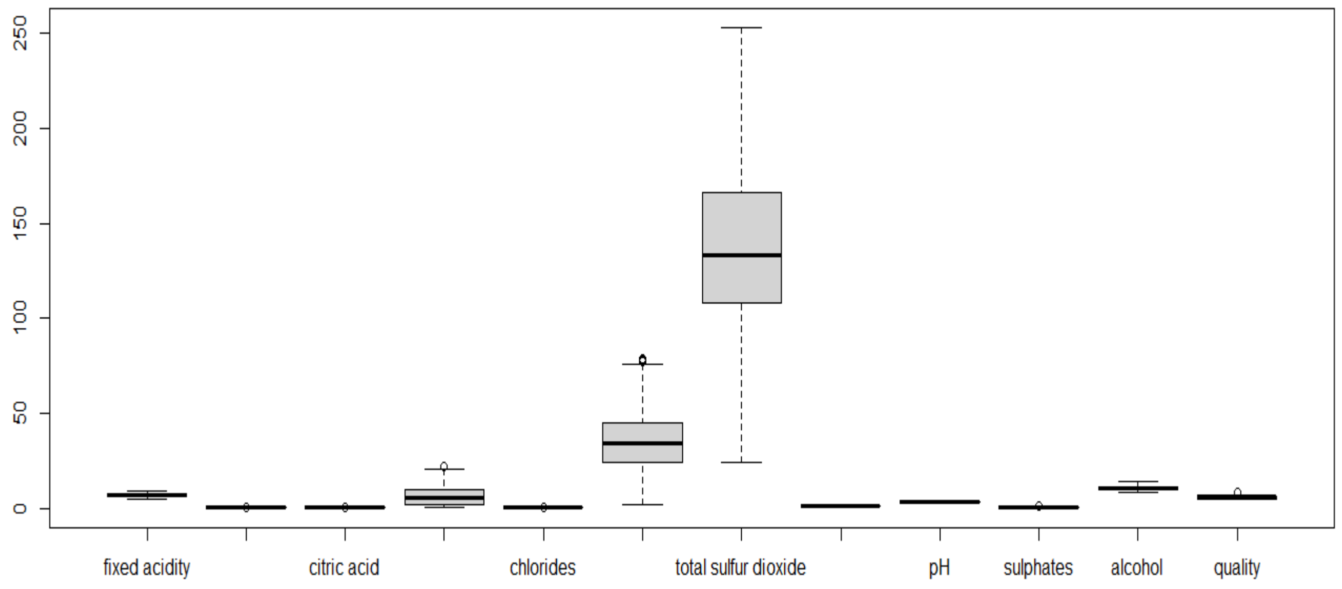


```r
# first column outliers removal
IQR_fa <- IQR(whitewine_data$`fixed acidity`)
lower_fa = quantile(whitewine_data$`fixed acidity`, .25) - 1.5*IQR_fa
upper_fa = quantile(whitewine_data$`fixed acidity`, .75) + 1.5*IQR_fa

# second column outliers removal
IQR_va <- IQR(whitewine_data$`volatile acidity`)
lower_va = quantile(whitewine_data$`volatile acidity`, .25) - 1.5*IQR_va
upper_va = quantile(whitewine_data$`volatile acidity`, .75) + 1.5*IQR_va

# 3 column outliers removal
IQR_ca <- IQR(whitewine_data$`citric acid`)
lower_ca = quantile(whitewine_data$`citric acid`, .25) - 1.5*IQR_ca
upper_ca = quantile(whitewine_data$`citric acid`, .75) + 1.5*IQR_ca

# 4 column outliers removal
IQR_rsu <- IQR(whitewine_data$`residual sugar`)
lower_rsu = quantile(whitewine_data$`residual sugar`, .25) - 1.5*IQR_rsu
upper_rsu = quantile(whitewine_data$`residual sugar`, .75) + 1.5*IQR_rsu

# 5 column outliers removal
IQR_cl <- IQR(whitewine_data$chlorides)
lower_cl = quantile(whitewine_data$chlorides, .25) - 1.5*IQR_cl
upper_cl = quantile(whitewine_data$chlorides, .75) + 1.5*IQR_cl

# 6 column outliers removal
IQR_fsd <- IQR(whitewine_data$`free sulfur dioxide`)
lower_fsd = quantile(whitewine_data$`free sulfur dioxide`, .25) - 1.5*IQR_fsd
upper_fsd = quantile(whitewine_data$`free sulfur dioxide`, .75) + 1.5*IQR_fsd

# 7 column outliers removal
IQR_tsd <- IQR(whitewine_data$`total sulfur dioxide`)
lower_tsd = quantile(whitewine_data$`total sulfur dioxide`, .25) - 1.5*IQR_tsd
upper_tsd = quantile(whitewine_data$`total sulfur dioxide`, .75) + 1.5*IQR_tsd
```



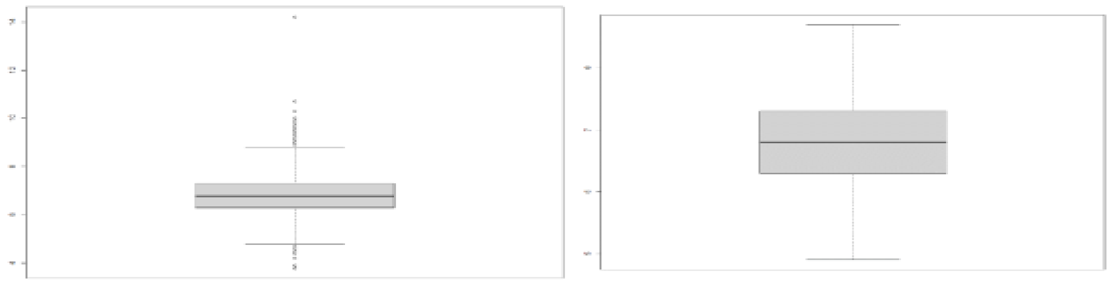
```r
# 8 column outliers removal
IQR_de <- IQR(whitewine_data$density)
lower_de = quantile(whitewine_data$density, .25) - 1.5*IQR_de
upper_de = quantile(whitewine_data$density, .75) + 1.5*IQR_de

# 9 column outliers removal
IQR_ph <- IQR(whitewine_data$pH)
lower_ph = quantile(whitewine_data$pH, .25) - 1.5*IQR_ph
upper_ph = quantile(whitewine_data$pH, .75) + 1.5*IQR_ph

# 10 column outliers removal
IQR_sul <- IQR(whitewine_data$sulphates)
lower_sul = quantile(whitewine_data$sulphates, .25) - 1.5*IQR_sul
upper_sul = quantile(whitewine_data$sulphates, .75) + 1.5*IQR_sul

# 11 column outliers removal
IQR_al <- IQR(whitewine_data$alcohol)
lower_al = quantile(whitewine_data$alcohol, .25) - 1.5*IQR_al
upper_al = quantile(whitewine_data$alcohol, .75) + 1.5*IQR_al

updated_whitewine <- subset(whitewine_data, whitewine_data$`fixed acidity` >= lower_fa & whitewine_data$`fixed acidity` <= upper_fa
                            & whitewine_data$`volatile acidity` >= lower_va & whitewine_data$`volatile acidity` <= upper_va
                            & whitewine_data$`citric acid` >= lower_ca & whitewine_data$`citric acid` <= upper_ca
                            & whitewine_data$`residual sugar` >= lower_rsu & whitewine_data$`residual sugar` <= upper_rsu
                            & whitewine_data$chlorides >= lower_cl & whitewine_data$chlorides <= upper_cl
                            & whitewine_data$`free sulfur dioxide` >= lower_fsd & whitewine_data$`free sulfur dioxide` <= upper_fsd
                            & whitewine_data$`total sulfur dioxide` >= lower_tsd & whitewine_data$`total sulfur dioxide` <= upper_tsd
                            & whitewine_data$density >= lower_de & whitewine_data$density <= upper_de
                            & whitewine_data$pH >= lower_ph & whitewine_data$pH <= upper_ph
                            & whitewine_data$sulphates >= lower_sul & whitewine_data$sulphates <= upper_sul
                            & whitewine_data$alcohol >= lower_al & whitewine_data$alcohol <= upper_al)

# Boxplot
boxplot(updated_whitewine)
```

By eliminating the outliers from the initial dataset, a new dataset is formed. The data has been renamed "Whitewine." After deleting the outliers, the boxplot is presented below.

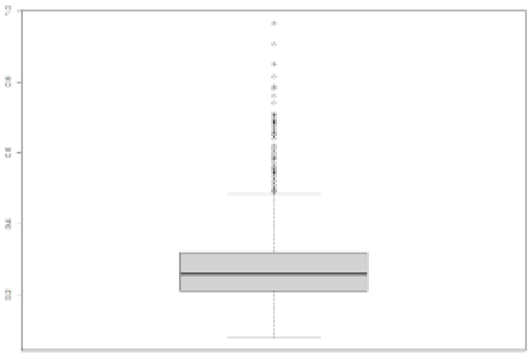


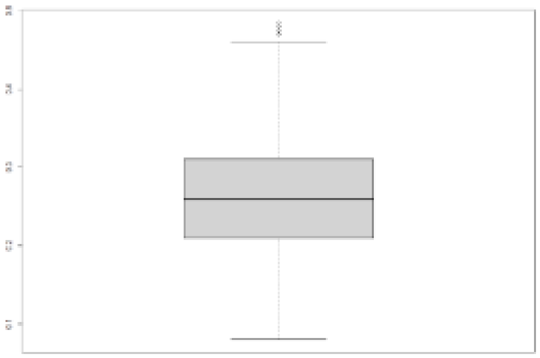
The following figures show how the data changes in each column after removing outliers.

- Fixed Acidity

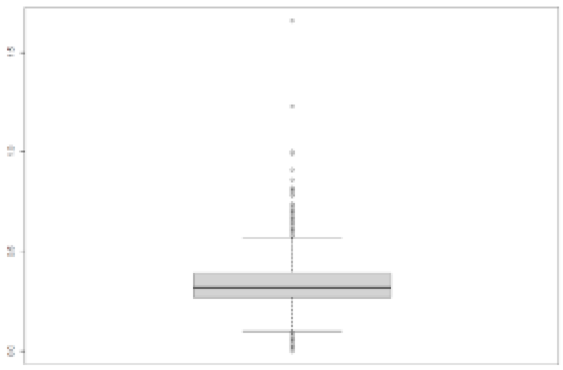


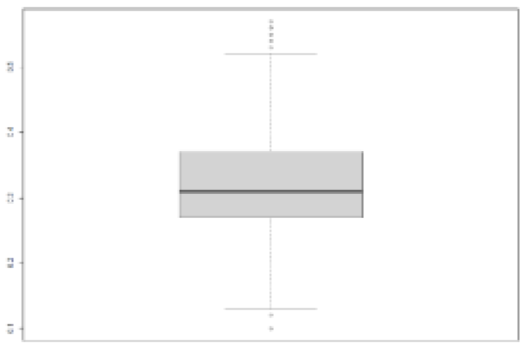
- Volatile Acidity



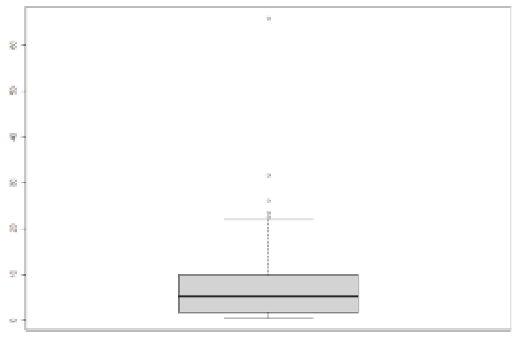
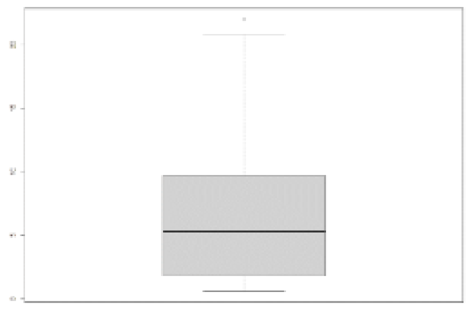


- Citric Acid

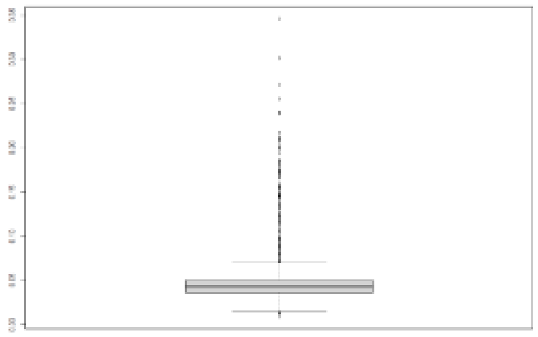
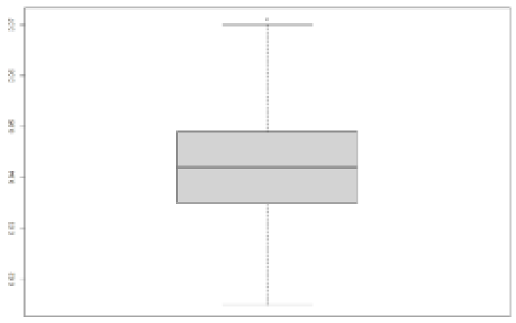




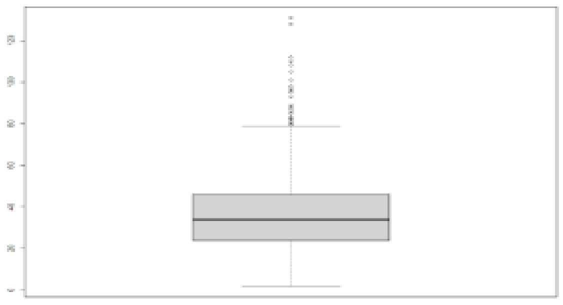
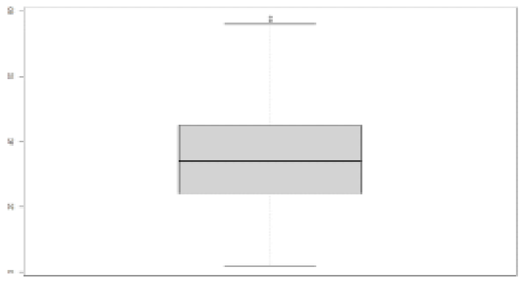
- Residual sugar

Chlorides

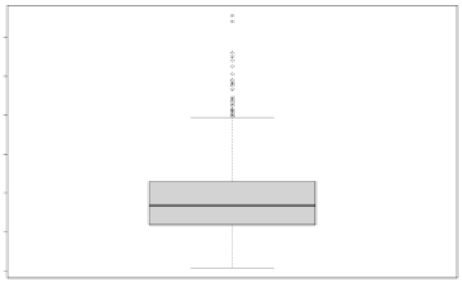
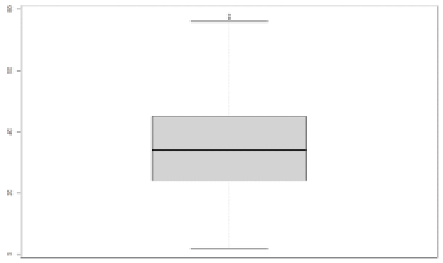



Free Sulphur dioxide

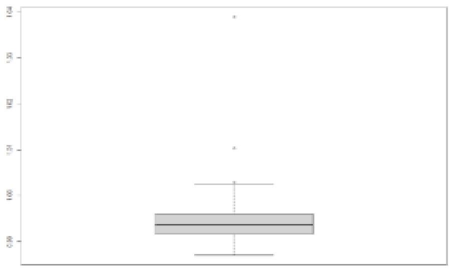
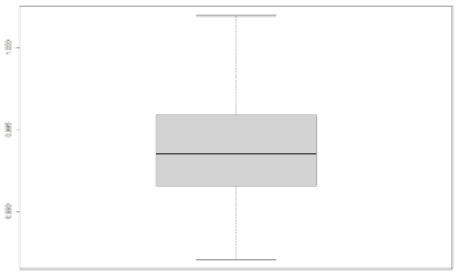



		7

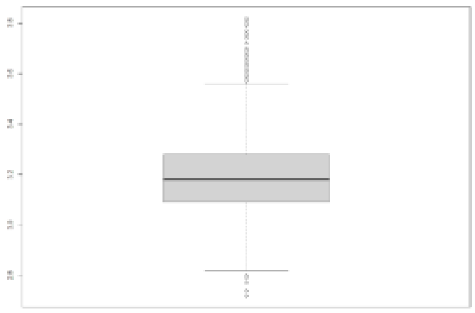
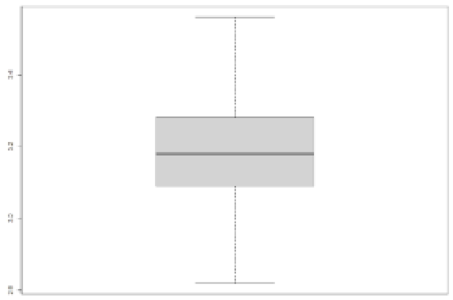
- Total Sulphur dioxide

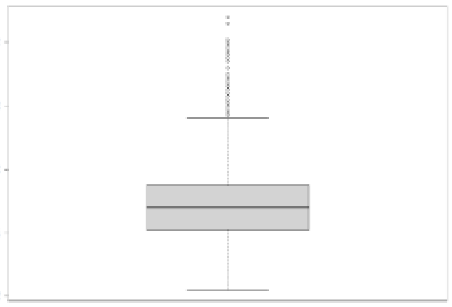
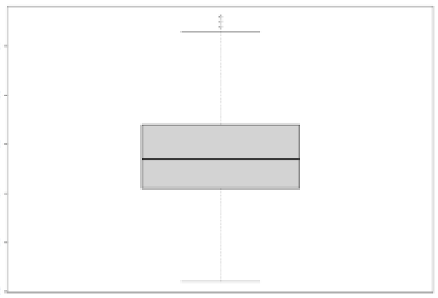
- Density

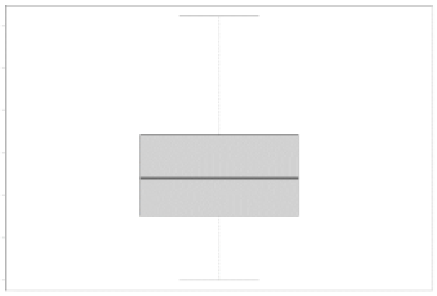
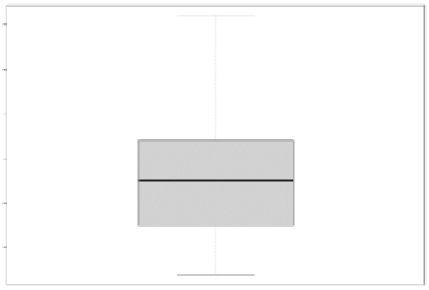
- pH

- Sulphate

- Alcohol

Data Normalization (Scaling)

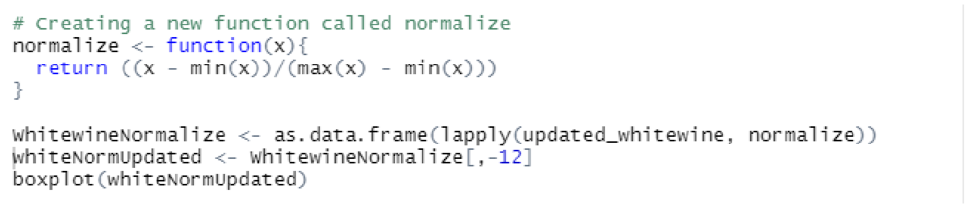
Normalization can take several forms; in the most basic situation, it is integrating all values measured on various scales into a single scale. Normalization is a rescaling strategy in statistics that aims to fit all data points into a range of 0 to 1 to bring them closer together. It is a widely used data scaling approach. In this technique of data scaling, the least value of each characteristic is transformed to 0, while the greatest value is turned to 1. In the normalization procedure, the difference between any value and the minimum value is divided by the difference between the maximum and minimum values. The following is a representation of normalization:

$$X_{norm} = \frac{x-(x)}{(x)-(x)}$$

Where x is any value from the feature x, min(X) is the feature's minimum value, and max(x) is the feature's maximum value.

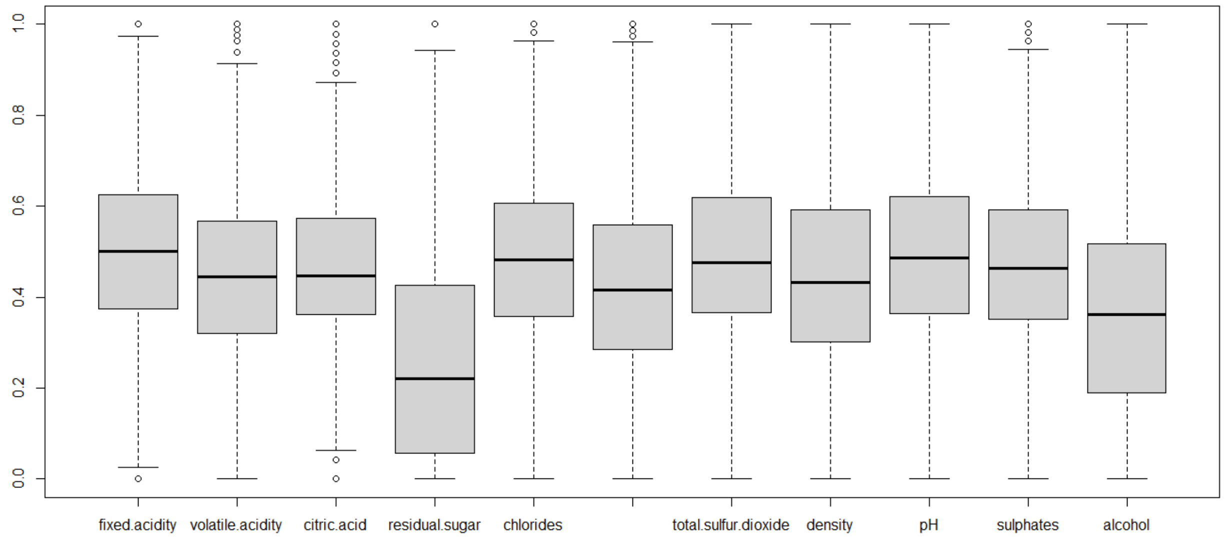
Using the whitewine2 dataset to apply the min-max normalization.

To normalize the dataset, I created a function named normalize. Using that method, I produced a new data set named WhitewineNormalize from the Whitewine dataset. WhitewineNormalize scales the data.



```
# Creating a new function called normalize
normalize <- function(x){
  return ((x - min(x))/(max(x) - min(x)))
}

whitewineNormalize <- as.data.frame(lapply(updated_whitewine, normalize))
whiteNormUpdated <- whitewineNormalize[,-12]
boxplot(whiteNormUpdated)
```

The boxplot result,



Cluster Centers Number

The "cluster center" is the arithmetic mean of all of the cluster's points.

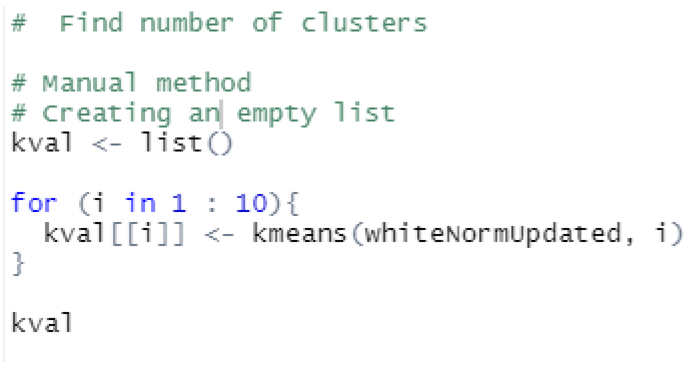
To determine the number of cluster centers in this project, one human approach and three automated methods are employed.

Manual Method

Fitting different k mean clustering models to our data across a variety of K values and judging how well they match is what is done here.

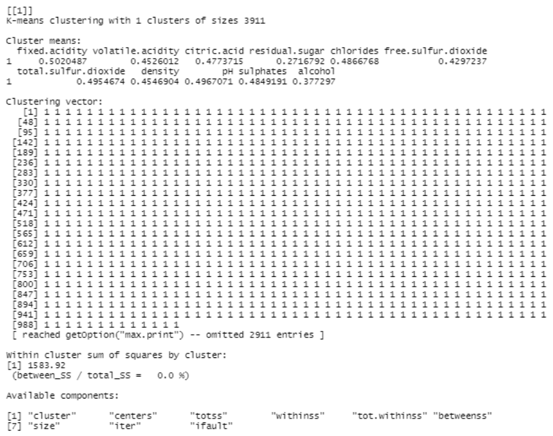
I created an empty list called kval and put I in the range of 1 to 10. The I value in the kval list is the outcome of whitewineNormupdated's k-means algorithm.

Essentially, loop over the values of 1 to 10 and fit a k-means model for each of these values.

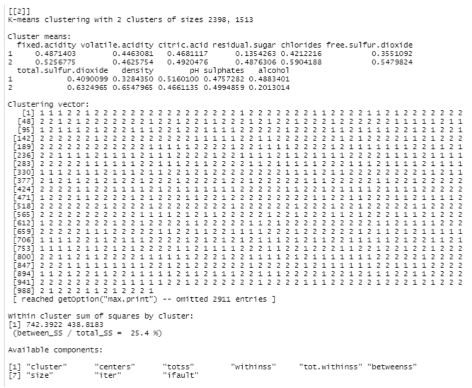


```
#  Find number of clusters

# Manual method
# Creating an empty list
kval <- list()

for (i in 1 : 10){
  kval[[i]] <- kmeans(whiteNormUpdated, i)
}

kval
```

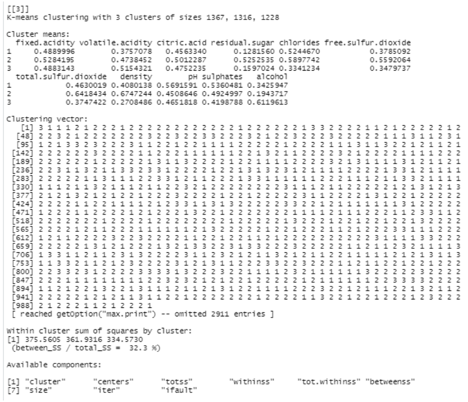
Below have the results ,



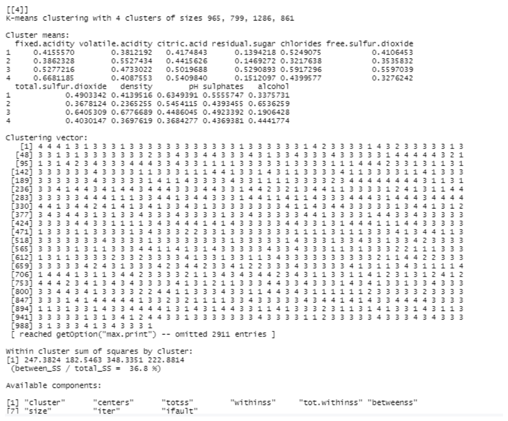
```
[[1]]
K-means clustering with 1 clusters of sizes 3911

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.5020487        0.4526012   0.4773715      0.2716792 0.4866768           0.4297237
  total.sulfur.dioxide   density        pH sulphates  alcohol
1            0.4954674 0.4546904 0.4967071 0.4849191 0.377297

Clustering vector:
   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  [48] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  [95] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [142] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [189] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [236] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [283] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [330] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [377] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [424] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [471] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [518] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [565] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [612] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [659] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [706] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [753] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [800] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [847] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [894] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [941] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [988] 1 1 1 1 1 1 1 1 1 1 1 1 1
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 1583.92
 (between_SS / total_SS =   0.0 %)

Available components:

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



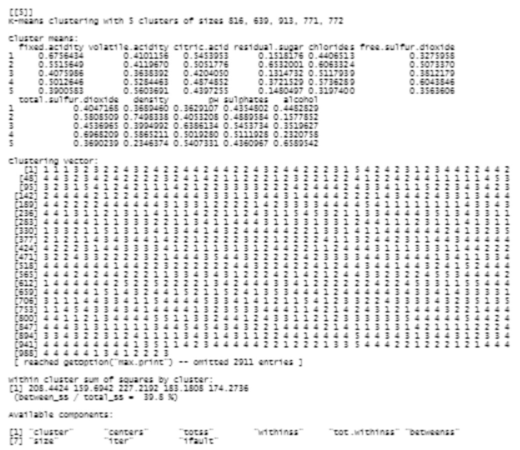
```
[[2]]
K-means clustering with 2 clusters of sizes 2398, 1513

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4871403        0.4463081   0.4681117      0.1354263 0.4212216           0.3551092
2     0.5256775        0.4625754   0.4920476      0.4876306 0.5904188           0.5479824
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.4090099 0.3284350 0.5160100 0.4757282 0.4883401
2            0.6324965 0.6547965 0.4661135 0.4994859 0.2013014

Clustering vector:
   [1] 1 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2
  [48] 2 2 1 2 1 2 2 2 2 2 2 2 1 2 2 1 2 2 1 1 2 2 2 1 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 1 1 1 1 1 1 2 1 1
  [95] 1 2 1 1 1 2 1 2 2 2 1 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 2 1 2 2 2 2 1 1 2 1 1 1 1 1 2 2 1 2 1 2 2 2 1
 [142] 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 2 2 2 2 2 2 2 1 2 1 2 2 2 2 1 1 2 1 2 2 2
 [189] 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2 2 2 1 2 2 1 1 1 1 2 2 2 2 1 2 1 1 1 1 1 1 1 2 2 1 1 2 1
 [236] 2 2 1 1 1 1 2 1 1 1 1 2 1 1 1 2 2 2 2 1 2 2 1 1 1 1 2 1 1 2 1 1 1 1 2 2 2 2 1 1 1 1 2 1 1 1 1
 [283] 2 2 2 2 2 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 2 2 1 1 2 1 1 1 1 1 2 2 2 1 1 1 2 1 1 1 1 1 2 1 1 1 1
 [330] 1 1 1 2 1 1 1 2 1 1 1 2 1 1 2 2 1 2 1 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 1 2 2 2 2 2 2 2 2 1 1 2 1 1
 [377] 2 1 2 1 1 2 1 2 1 2 2 1 2 2 2 1 2 2 2 2 1 2 2 1 2 2 2 2 2 1 1 1 1 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2
 [424] 2 2 2 2 1 1 2 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 2 1 2 1 1 1 1 2 1 1 1 1 1 2 2 2 2 2
 [471] 1 2 2 2 1 1 2 2 2 2 1 2 1 2 2 2 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 1 2 2 2 1 1 2 1 1 1 1 2
 [518] 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 2 2 2 1 1 2 2 2 2
 [565] 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 2 1 2 1 1 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 1 2 2 2 1 1 1 1 2 1 2 2 2
 [612] 1 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 2 2 2
 [659] 2 2 2 2 2 1 1 1 2 2 2 2 2 1 1 2 2 1 1 2 2 1 1 1 1 1 2 2 2 1 2 2 2 2 2 2 2 1 2 1 1 2 1 2 1 1 1 1 1
 [706] 1 1 1 1 2 2 1 1 2 1 1 1 2 2 2 2 1 2 1 2 1 2 1 2 1 1 1 2 1 2 1 1 2 2 1 1 1 1 1 1 2 1 2 2 1 1 2 1
 [753] 1 1 1 1 2 1 1 2 1 2 1 2 2 2 1 1 2 1 1 1 1 2 2 2 1 1 2 2 1 2 2 2 1 1 2 2 1 2 2 1 2 2 1 2 2 2
 [800] 2 2 1 1 2 1 1 2 2 2 2 1 1 1 1 1 1 2 2 2 1 2 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 2
 [847] 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 2 2 2 2 2 2 1 2 2 2 2 2 1 2 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 2
 [894] 1 1 2 1 2 2 1 1 2 2 1 1 1 1 1 1 2 1 1 2 1 1 1 2 2 1 1 1 2 2 2 2 1 2 2 1 2 1 1 1 2 2 1 1 2 2 2 2
 [941] 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 1 2 2 2 2 2 1 2 2 2 1 2 1 2 2 2
 [988] 2 1 2 2 2 1 1 1 2 1 2 2 2 1
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 742.3922 438.8183
 (between_SS / total_SS =  25.4 %)

Available components:

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



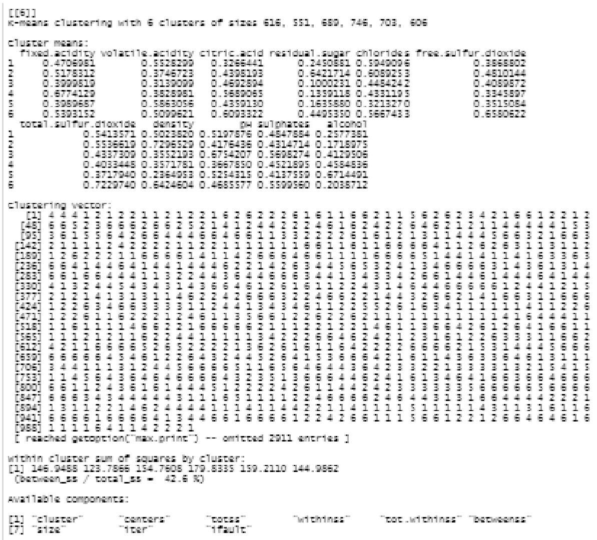
```
[[3]]
K-means clustering with 3 clusters of sizes 1367, 1316, 1228

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4889996        0.3757078   0.4563340      0.1281560 0.5244670           0.3785092
2     0.5284195        0.4738452   0.5012287      0.5252535 0.5897742           0.5592064
3     0.4883143        0.5154321   0.4752235      0.1597024 0.3341234           0.3479737
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.4630019 0.4080138 0.5691591 0.5360481 0.3425947
2            0.6418434 0.6747244 0.4508646 0.4924997 0.1943717
3            0.3747422 0.2708486 0.4651818 0.4198788 0.6119613

Clustering vector:
   [1] 3 1 1 1 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 3 3 2 2 2 2 1 1 2 1 2 2 2 2 2 1 2
  [48] 2 2 3 2 1 2 2 2 2 2 2 3 2 2 3 2 2 3 3 2 2 2 3 2 1 2 2 3 2 2 2 3 2 2 2 2 2 1 1 1 3 1 1 2 3 1
  [95] 1 2 1 3 3 2 3 2 2 2 3 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 2 1 2 2 2 2 1 1 1 3 1 1 3 2 2 1 2 1 1 2 1
 [142] 2 2 2 2 2 2 3 2 2 2 2 1 1 2 2 2 1 1 1 1 1 1 2 2 1 3 2 1 1 2 2 2 2 3 2 1 2 2 2 2 1 1 1 1 2 2 2
 [189] 2 2 2 2 2 2 1 2 2 2 2 2 1 3 1 1 3 2 2 2 2 1 2 2 1 1 1 1 2 2 2 2 3 2 1 3 1 1 1 1 3 1 2 1 1 2 1
 [236] 2 2 3 1 1 3 2 1 1 3 3 2 3 3 1 2 2 2 2 1 2 2 1 3 1 3 2 3 1 2 1 1 1 1 2 2 2 2 1 3 3 1 2 1 1 1 1
 [283] 2 2 2 2 2 1 1 3 1 1 1 2 2 3 3 1 2 1 1 2 2 2 1 3 3 1 1 1 1 1 1 2 2 2 1 1 1 2 1 1 3 3 2 1 1 1 1
 [330] 1 1 1 2 1 1 3 2 1 1 1 2 1 1 2 2 3 2 1 2 2 2 2 2 2 2 2 3 1 1 1 2 1 1 2 2 2 2 2 1 2 1 3 1 2 1 3
 [377] 2 1 2 1 3 2 1 2 1 2 2 1 2 2 2 3 2 2 2 2 1 2 2 1 2 2 2 2 2 3 1 1 2 2 2 2 2 1 3 1 2 2 1 2 2 2 2 2
 [424] 2 2 2 2 1 1 2 2 1 1 1 1 2 1 2 3 3 1 1 3 1 3 2 2 2 2 2 3 3 2 2 1 2 1 1 1 1 1 1 1 1 3 1 2 2 2 2 2
 [471] 1 2 2 2 1 1 2 2 2 2 1 2 1 2 2 2 1 3 2 2 1 2 2 2 2 2 2 2 2 1 1 1 1 2 1 1 1 2 2 2 1 1 2 3 3 1 1 2
 [518] 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 3 2 2 2 1 2 2 1 2 2 1 2 2 1 3 2 2 2 2
 [565] 2 2 2 2 1 2 1 1 2 2 2 1 1 1 1 1 1 2 1 3 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 1 2 2 2 3 3 1 1 1 2 2 2
 [612] 1 2 1 1 2 2 2 2 3 2 2 3 2 2 2 2 1 1 2 2 1 2 2 1 1 2 1 2 2 2 2 2 2 2 2 2 2 3 1 1 1 1 3 3 2 2 2
 [659] 2 2 2 2 2 1 3 1 2 1 2 2 2 1 3 2 1 3 3 2 2 3 1 3 3 2 2 2 3 2 2 2 2 2 1 1 2 1 1 2 3 2 1 1 1 1 3
 [706] 1 3 3 1 1 2 1 1 2 3 1 3 2 2 2 2 3 1 1 2 3 2 1 2 1 1 3 2 1 2 1 1 2 2 1 1 1 1 1 1 2 1 2 1 3 1 1 3
 [753] 1 1 3 3 2 1 1 2 1 2 3 2 2 2 2 3 1 2 1 3 1 1 2 2 2 3 3 2 2 3 2 2 2 1 1 2 1 1 2 2 1 2 2 1 2 2 2
 [800] 2 2 3 3 2 3 1 2 2 2 2 3 3 3 3 1 3 2 2 2 3 2 2 1 1 1 1 2 3 2 1 1 1 1 1 1 3 2 2 2 2 2 3 2 2 2 2
 [847] 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 3 2 3 2 1 2 1 2 2 1 2 2 2 2 2 1 2 1 1 1 1 1 1 1 2 2 3 3 3 3 2 2 2 2
 [894] 1 1 2 1 2 2 1 3 2 2 1 3 1 1 1 2 1 1 2 1 3 1 2 2 1 1 1 1 2 2 2 3 2 2 1 2 1 1 1 2 2 1 3 2 1 1 2
 [941] 2 2 2 2 2 1 2 1 2 1 1 3 1 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 3 2 2 1 2 2 1 2 2 2 1 2 3 2 2 2
 [988] 2 1 2 2 2 1 1 2 1 2 2 2 1
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 375.5605 361.9316 334.5730
 (between_SS / total_SS =  32.3 %)

Available components:

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



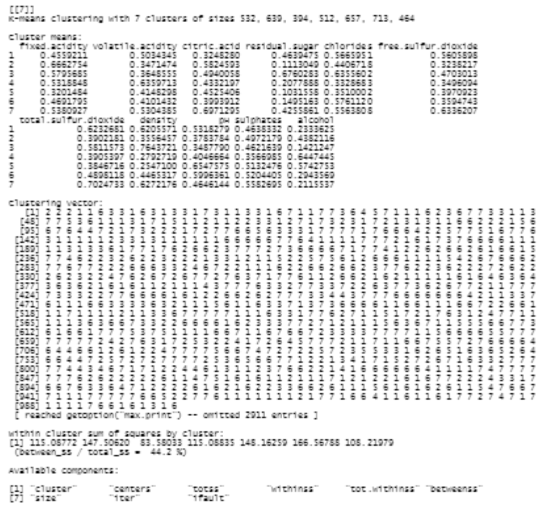
```
[[4]]
K-means clustering with 4 clusters of sizes 965, 799, 1286, 861

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4155570        0.3812192   0.4174843      0.1394218 0.5249075           0.4106453
2     0.3862328        0.5527434   0.4415626      0.1469272 0.3217638           0.3535832
3     0.5277216        0.4733022   0.5019688      0.5290893 0.5917296           0.5597039
4     0.6681185        0.4087553   0.5409840      0.1512097 0.4399577           0.3276242
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.4903342 0.4139516 0.6349391 0.5555747 0.3375731
2            0.3678124 0.2365255 0.5454115 0.4393455 0.6536259
3            0.6405309 0.6776689 0.4486045 0.4923392 0.1906428
4            0.4030147 0.3697619 0.3684277 0.4369381 0.4441774

Clustering vector:
   [1] 4 4 4 1 3 1 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 1 4 2 3 3 3 3 1 4 3 2 3 3 3 3 3 1 3
  [48] 3 3 1 3 1 3 3 3 3 3 3 3 2 3 3 4 3 3 4 4 3 3 3 4 3 1 3 3 4 3 3 3 4 3 3 3 3 3 1 4 4 4 4 4 3 2 1
  [95] 1 3 1 4 2 3 4 3 3 3 4 4 4 3 3 4 3 3 1 1 1 1 3 3 3 3 3 1 3 3 3 3 1 1 1 4 4 4 2 3 3 1 3 1 1 3 1
 [142] 3 3 3 3 3 3 4 3 3 3 3 1 1 3 3 3 1 1 1 4 4 1 3 3 1 4 3 1 1 3 3 3 3 4 1 1 3 3 3 3 1 1 4 1 3 3 3
 [189] 3 3 3 3 3 3 4 3 3 3 3 3 1 4 1 1 4 3 3 3 3 4 3 3 1 1 1 1 3 3 3 3 2 3 4 4 4 4 4 4 4 4 3 1 1 3 1
 [236] 3 3 4 1 4 4 3 4 1 4 4 3 4 4 4 3 3 3 4 4 3 3 1 4 4 2 3 2 1 3 4 4 1 1 3 3 3 3 1 2 4 1 3 1 1 4 4
 [283] 3 3 3 3 3 4 4 4 1 1 1 3 3 4 4 1 3 4 4 3 3 3 1 4 4 1 1 4 1 1 4 3 3 3 4 4 4 3 1 4 4 4 3 4 4 4 4
 [330] 4 4 1 3 4 4 2 4 1 4 1 3 4 1 3 3 4 3 1 3 3 3 3 3 3 3 3 4 1 1 4 3 4 4 3 3 3 3 3 1 3 4 4 1 3 1 2
 [377] 3 4 3 4 4 3 1 3 1 3 3 4 3 3 3 4 3 3 3 3 1 3 3 4 3 3 3 3 3 4 4 1 3 3 3 3 1 4 4 3 3 4 3 3 3 3 3
 [424] 3 3 3 3 4 4 3 3 1 1 1 1 3 4 3 4 4 4 1 4 1 4 3 3 3 3 3 4 4 3 3 1 3 1 4 4 4 1 1 1 4 4 3 3 3 3 3
 [471] 1 3 3 3 1 1 3 3 3 3 1 3 4 3 3 3 2 2 3 3 1 3 3 3 3 3 3 3 3 1 1 1 1 3 1 1 1 3 3 3 4 1 3 4 4 1 1 3
 [518] 3 3 3 3 3 3 3 4 3 3 3 3 1 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 1 4 3 3 3 1 3 3 4 3 3 1 3 3 4 2 3 3 3 3
 [565] 3 3 3 3 1 3 1 1 3 3 3 4 4 1 1 4 1 3 1 4 3 3 3 3 3 4 3 3 4 3 3 3 1 1 3 1 3 3 3 2 2 1 1 1 3 3 3
 [612] 1 3 1 1 3 3 3 3 2 3 3 2 3 3 3 3 4 1 3 3 1 3 3 1 1 3 4 3 3 3 3 3 3 3 3 3 3 2 1 1 4 4 2 2 3 3 3
 [659] 3 3 3 3 3 4 2 4 3 1 3 3 3 4 2 3 4 4 2 3 3 4 1 2 2 3 3 3 4 3 3 3 3 3 4 1 3 1 1 3 4 3 1 1 1 1 4
 [706] 1 4 4 4 1 3 1 1 3 4 4 2 3 3 3 3 2 1 1 3 4 3 4 3 4 4 2 3 4 3 1 1 3 3 1 1 4 1 2 3 1 3 1 2 4 1 2
 [753] 4 4 4 2 3 4 1 3 4 3 4 3 3 3 3 4 1 3 1 2 1 1 3 3 3 4 4 3 3 4 3 3 3 1 4 3 4 1 3 3 1 3 3 4 3 3 3
 [800] 3 3 4 4 3 4 1 3 3 3 3 2 2 4 4 1 1 3 3 3 4 3 3 1 1 4 4 3 4 3 1 1 1 1 1 1 2 3 3 3 3 3 2 3 3 3 3
 [847] 3 3 3 1 4 1 4 4 4 4 4 1 3 3 2 3 2 1 1 1 1 3 3 4 3 3 3 3 3 4 3 4 4 1 4 1 4 3 3 4 4 4 4 3 3 3 3
 [894] 1 1 3 1 3 3 1 4 3 3 4 4 4 4 1 3 1 4 3 1 4 4 3 3 1 1 4 1 3 3 3 2 3 1 1 3 1 4 1 3 3 1 4 3 1 1 3
 [941] 3 3 3 3 3 1 3 1 3 4 1 2 4 4 3 3 1 3 3 3 3 3 3 3 4 3 3 3 3 1 1 2 3 3 3 3 3 4 3 3 3 4 3 4 3 3 3
 [988] 3 1 3 3 3 4 1 3 4 3 3 3 1
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 247.3824 182.5463 348.3351 222.8814
 (between_SS / total_SS =  36.8 %)

Available components:

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



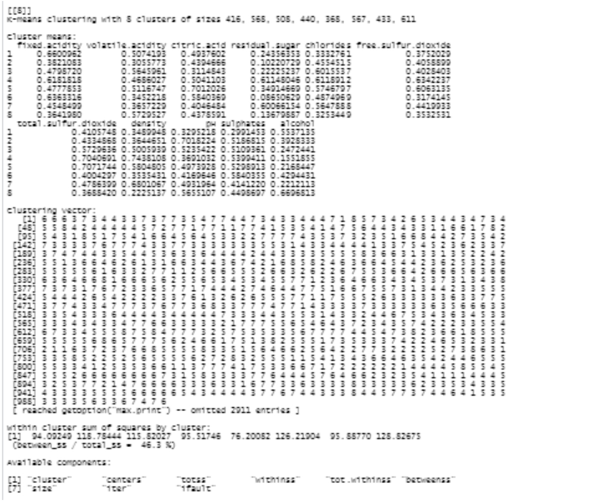
```
[[5]]
K-means clustering with 5 clusters of sizes 816, 639, 913, 771, 772

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.6756434        0.4101156   0.5453953      0.1518176 0.4406513           0.3275958
2     0.5515649        0.4119670   0.5051776      0.6532001 0.6063324           0.5073370
3     0.4075986        0.3638392   0.4204050      0.1314732 0.5117939           0.3812179
4     0.5012646        0.5284463   0.4874852      0.3721529 0.5736289           0.6043846
5     0.3900583        0.5603691   0.4397255      0.1480497 0.3197400           0.3563606
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.4047168 0.3689460 0.3629107 0.4354802 0.4482829
2            0.5808509 0.7498338 0.4053208 0.4889584 0.1577852
3            0.4536965 0.3994992 0.6386134 0.5453734 0.3519627
4            0.6968209 0.5865211 0.5019280 0.5111928 0.2320758
5            0.3690239 0.2346374 0.5407331 0.4360967 0.6589542

Clustering vector:
   [1] 1 1 1 3 2 3 2 2 4 3 2 4 2 2 4 4 2 4 4 2 2 4 3 2 4 4 2 2 2 3 1 5 4 2 4 2 3 1 2 3 4 4 2 2 4 4 2
  [48] 4 4 3 2 3 4 2 4 2 2 2 2 3 2 4 1 4 2 1 1 2 2 2 1 2 3 2 2 1 2 2 4 1 2 2 4 2 4 4 1 1 1 1 1 1 4 5 3
  [95] 3 2 3 1 5 4 1 2 4 2 1 1 1 4 2 1 2 2 3 3 3 3 2 2 2 4 2 4 4 4 2 4 3 3 4 1 1 1 5 2 2 3 4 3 4 2 3
 [142] 2 4 4 4 4 2 1 2 2 4 2 4 4 4 4 4 3 3 3 1 1 3 4 4 3 1 4 4 4 4 2 2 2 1 4 3 4 4 2 4 3 3 1 3 4 4 4
 [189] 4 4 2 2 2 2 4 4 4 4 4 4 3 1 3 3 1 2 2 2 2 1 4 2 3 3 3 3 4 4 4 4 5 4 1 1 1 1 1 1 1 1 1 4 3 3 4 3
 [236] 4 4 1 3 1 1 2 1 3 1 1 4 1 1 1 4 2 2 1 1 2 4 3 1 1 5 4 5 3 2 1 1 3 3 4 4 4 4 3 5 1 3 4 3 3 1 1
 [283] 4 4 4 4 4 1 1 1 3 3 3 2 2 1 1 3 4 1 1 4 4 4 3 1 1 3 3 1 3 3 1 2 4 4 1 1 1 2 3 1 1 1 4 1 3 1 1
 [330] 1 3 3 2 1 1 5 1 3 1 3 4 1 3 4 4 1 4 3 2 4 4 4 4 4 2 2 1 3 3 1 4 1 1 4 4 4 4 4 4 2 4 1 3 2 3 5
 [377] 2 1 2 1 1 4 3 4 3 4 4 1 4 2 2 1 2 2 2 2 3 2 2 1 2 2 2 2 4 1 1 3 2 4 4 2 3 1 1 4 4 3 4 4 4 4 4
 [424] 4 2 2 2 3 1 4 4 3 3 3 3 4 1 2 1 1 1 3 1 3 1 2 4 4 2 2 1 1 2 4 4 4 3 1 1 1 3 3 3 1 1 4 4 2 2 2
 [471] 3 2 2 4 3 3 2 2 2 2 3 2 1 4 4 4 3 5 4 4 3 2 2 2 2 2 4 2 3 3 3 3 4 4 3 3 4 4 4 1 3 4 1 1 3 3 4
 [518] 4 4 4 4 4 4 4 1 4 4 2 2 3 2 2 2 2 2 2 4 4 3 2 2 4 2 2 4 1 2 4 4 3 4 4 1 4 4 3 2 4 1 5 2 4 4 4
 [565] 4 4 4 2 4 2 4 4 2 2 2 1 1 3 3 4 3 4 3 1 2 2 2 4 2 1 4 2 1 2 4 4 3 3 2 3 2 2 4 5 5 3 4 4 4 4 2
 [612] 1 4 4 4 4 4 4 2 5 2 2 5 2 2 2 2 1 3 4 2 4 4 4 3 3 4 1 2 2 2 2 2 2 4 2 2 4 5 3 3 1 1 5 5 4 4 4
 [659] 4 4 4 4 4 1 5 1 4 3 2 4 4 1 5 2 1 1 5 2 4 1 3 5 3 4 4 4 1 2 4 4 4 4 4 3 4 3 3 4 1 4 3 3 3 3 1
 [706] 3 1 1 1 4 4 3 3 4 1 1 5 4 4 4 4 5 3 3 4 1 4 1 2 1 1 5 4 1 2 3 3 2 2 4 3 3 3 3 4 3 2 3 5 1 3 5
 [753] 1 1 4 5 4 3 3 4 3 4 1 4 4 4 4 1 3 2 3 5 3 3 4 4 4 1 1 4 2 1 4 2 4 3 1 2 1 4 4 4 3 2 2 1 4 4 4
 [800] 4 4 1 1 2 1 3 4 4 4 4 5 5 1 1 3 3 2 4 4 1 2 4 3 3 1 1 2 1 2 3 3 3 3 3 3 5 4 4 4 4 4 5 4 4 2 4
 [847] 4 4 4 3 1 3 1 1 1 1 1 1 3 4 4 5 4 5 4 3 4 3 2 2 1 4 4 4 4 2 1 4 1 1 3 1 3 1 4 2 1 1 1 1 2 2 2 4
 [894] 3 3 4 3 2 2 3 1 2 4 1 1 1 1 3 4 3 1 4 3 1 1 2 2 4 4 1 4 4 4 4 5 4 4 3 4 3 1 3 4 4 3 1 2 3 3 4
 [941] 4 4 4 4 4 4 4 4 4 4 1 3 5 1 1 4 2 3 4 4 4 4 4 2 2 1 2 2 2 1 3 3 5 4 4 4 2 2 1 2 2 2 1 2 1 4 4 4
 [988] 4 4 4 4 4 1 3 4 1 2 2 2 3
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 208.4424 159.6942 227.2192 183.1808 174.2736
 (between_SS / total_SS =  39.8 %)

Available components:

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



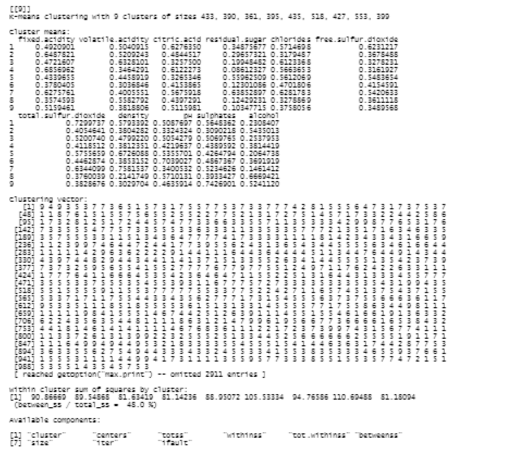
```
[[6]]
K-means clustering with 6 clusters of sizes 616, 551, 689, 746, 703, 606

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4706981        0.5528299   0.3266441      0.2450881 0.3949096           0.3868802
2     0.5178312        0.3746723   0.4398193      0.6421714 0.6089233           0.4810144
3     0.3999819        0.3139099   0.4692594      0.1000231 0.4484242           0.4089872
4     0.6774129        0.3828981   0.3689065      0.1359118 0.4331193           0.3345897
5     0.3989687        0.5863056   0.4359130      0.1635880 0.3213270           0.3515084
6     0.5393152        0.5099621   0.6093322      0.4495350 0.5667433           0.6580622
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.5413571 0.5023820 0.5197876 0.4847884 0.2577381
2            0.5536619 0.7296529 0.4176436 0.4314714 0.1718975
3            0.4337309 0.3552193 0.6754207 0.5698274 0.4129506
4            0.4033448 0.3571781 0.3667850 0.4521895 0.4584836
5            0.3717940 0.2364953 0.5254315 0.4137559 0.6714491
6            0.7229740 0.6424604 0.4685377 0.5599580 0.2038712

Clustering vector:
   [1] 4 4 4 1 2 1 2 2 1 1 2 1 2 2 1 6 2 6 2 2 2 6 1 6 1 1 6 6 2 1 1 5 6 2 6 2 3 4 2 1 6 6 1 2 2 1 2
  [48] 6 6 5 2 3 6 6 6 2 6 6 2 5 2 1 4 1 2 4 4 2 2 2 4 6 1 6 2 4 2 2 6 4 6 2 1 2 1 1 4 4 4 4 4 1 5 3
  [95] 3 6 1 5 5 6 4 2 6 6 4 4 4 6 6 4 6 6 1 1 3 3 2 2 2 2 6 1 6 1 2 1 3 1 1 4 4 4 5 6 6 3 2 1 6 6 3
 [142] 2 1 1 1 1 2 4 2 2 2 2 1 1 2 2 1 1 1 1 1 1 1 6 6 1 1 6 1 1 6 6 6 6 4 1 1 2 6 2 6 3 1 1 3 1 1 2
 [189] 1 2 6 2 2 2 1 1 6 6 6 6 1 4 1 1 4 2 6 6 6 4 6 6 1 1 1 1 6 6 6 6 5 1 4 4 1 4 1 1 4 1 6 3 3 6 3
 [236] 6 6 4 1 4 4 6 4 1 4 4 1 4 4 4 6 2 2 1 4 2 6 3 4 4 5 6 5 3 2 4 1 3 4 6 6 6 6 3 1 4 3 6 1 3 1 4
 [283] 6 6 1 6 6 4 4 4 1 1 3 2 2 4 4 3 6 4 4 6 6 6 3 4 4 1 3 4 3 3 4 2 6 6 1 4 4 6 1 4 4 4 6 4 1 4 4
 [330] 4 1 3 2 4 4 5 4 3 4 3 1 4 3 6 6 4 6 1 2 6 1 6 1 1 2 2 4 3 1 4 6 4 4 6 6 6 6 6 6 1 2 4 4 1 2 1 5
 [377] 2 1 2 1 4 1 3 1 3 1 1 4 6 2 2 4 2 6 6 6 3 2 2 4 6 6 2 2 1 4 4 3 2 6 6 2 1 4 1 6 6 3 1 1 6 6 6
 [424] 1 2 2 6 3 4 6 6 3 3 3 3 1 1 2 4 4 1 3 4 3 4 6 1 1 2 2 5 5 2 6 1 6 3 4 1 1 1 1 1 1 1 4 1 1 2 2 6
 [471] 1 2 2 6 1 1 6 2 2 2 1 2 4 6 1 1 3 5 6 6 1 2 2 6 2 2 6 2 1 1 1 1 1 1 1 1 1 1 1 4 1 6 4 4 4 1 1
 [518] 1 1 6 1 1 1 1 4 6 6 2 2 1 6 6 6 6 6 2 1 1 1 2 2 1 2 2 1 4 6 1 1 3 6 6 4 2 6 1 2 6 4 1 6 6 1 1
 [565] 1 1 1 2 1 2 1 1 6 2 2 4 4 1 1 1 1 1 1 3 4 2 2 2 6 6 4 6 2 4 2 1 2 3 1 6 1 2 2 6 3 3 3 1 1 6 6 2
 [612] 4 2 1 1 6 6 6 6 5 2 6 5 2 2 2 2 1 3 6 2 6 1 6 1 1 6 4 2 2 2 2 2 6 6 6 6 2 1 5 3 1 4 4 4 5 6 6 6
 [659] 6 6 6 6 6 4 5 4 6 1 2 2 6 4 3 2 4 4 5 2 6 4 1 5 3 6 6 6 4 2 1 6 1 1 4 3 6 3 3 6 4 6 1 3 1 1 1
 [706] 3 4 4 1 1 1 3 1 2 4 4 5 6 6 6 6 5 1 1 6 5 6 4 6 4 4 3 6 4 2 3 3 2 2 1 3 3 3 3 1 3 2 1 5 4 1 5
 [753] 1 1 4 5 6 4 3 6 4 6 4 6 6 6 6 4 3 2 3 5 1 3 6 6 6 4 4 6 2 4 1 6 1 3 4 6 4 1 6 6 3 6 6 4 6 6 6
 [800] 6 6 1 1 2 4 3 6 1 6 1 4 4 4 5 1 2 2 2 2 4 2 6 1 1 4 4 2 4 2 3 3 3 3 3 3 3 5 6 6 6 6 6 5 6 6 6 6
 [847] 6 6 6 3 4 3 4 4 4 4 4 3 1 1 1 6 5 1 1 1 1 2 2 4 6 6 6 6 2 4 6 4 4 3 1 3 1 6 6 4 4 4 4 2 2 2 1
 [894] 1 3 1 1 2 2 1 4 6 2 4 4 4 4 1 1 1 4 1 1 4 4 2 2 1 1 4 1 1 1 1 5 1 1 1 1 1 4 3 1 1 3 1 6 1 1 6
 [941] 6 6 6 6 1 6 6 6 6 4 1 3 4 4 6 6 1 6 6 6 6 1 2 2 4 2 6 6 1 1 1 5 6 6 1 2 2 1 2 6 6 4 6 4 6 1 6
 [988] 1 1 1 1 6 4 1 1 4 2 2 2 1
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 146.9488 123.7866 154.7608 179.8335 159.2110 144.9862
 (between_SS / total_SS =  42.6 %)

Available components:

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



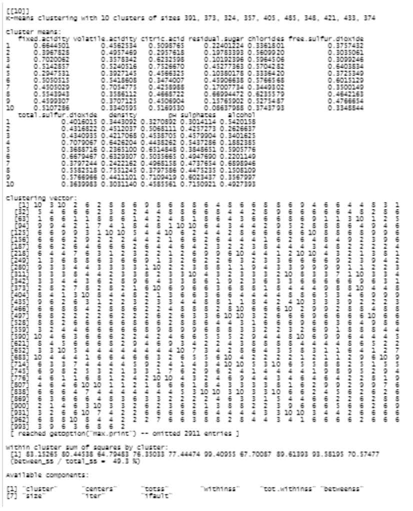
```
[[7]]
K-means clustering with 7 clusters of sizes 532, 639, 394, 512, 657, 713, 464

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4559211        0.5034345   0.3248280      0.4639475 0.5665951           0.5605898
2     0.6662754        0.3471474   0.5824593      0.1113049 0.4406718           0.3238217
3     0.5795685        0.3648555   0.4940058      0.6760283 0.6355602           0.4703013
4     0.5318848        0.6359713   0.4332197      0.2077888 0.3328683           0.3496094
5     0.3201484        0.4148298   0.4525406      0.1031558 0.3510002           0.3970923
6     0.4691795        0.4101432   0.3993912      0.1495163 0.5761120           0.3594743
7     0.5380927        0.5304385   0.6971295      0.4255861 0.5563808           0.6336207
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.6232681 0.6205571 0.5318279 0.4638332 0.2333625
2            0.3902181 0.3556457 0.3783784 0.4972179 0.4382116
3            0.5811573 0.7643721 0.3487790 0.4621639 0.1421247
4            0.3905397 0.2792719 0.4046664 0.3566985 0.6447445
5            0.3846716 0.2547100 0.6547575 0.5132476 0.5742753
6            0.4898118 0.4465317 0.5996361 0.5204405 0.2943569
7            0.7024733 0.6272176 0.4646144 0.5582695 0.2115537

Clustering vector:
   [1] 2 2 2 1 1 6 3 3 1 6 3 1 3 3 1 7 3 1 1 3 3 1 6 7 1 1 7 7 3 6 4 5 7 1 1 1 6 2 3 6 7 7 3 3 1 1 3
  [48] 7 7 5 3 6 1 1 1 3 1 7 1 5 1 1 2 1 1 2 2 3 3 1 2 7 1 7 3 2 3 1 7 2 1 3 1 3 1 1 6 6 2 2 2 1 5 6
  [95] 6 7 6 4 4 7 2 1 7 3 2 2 2 1 7 2 7 7 6 6 5 6 3 3 3 1 7 7 7 7 1 7 6 6 6 4 2 2 5 7 7 5 1 6 7 7 6
 [142] 3 1 1 1 1 1 2 3 3 1 3 1 1 1 1 1 1 1 6 6 6 6 6 7 7 6 4 1 1 1 1 7 7 7 2 1 6 1 7 3 7 6 6 6 6 1 1 1
 [189] 1 1 3 1 3 3 6 1 7 7 1 7 6 2 6 6 2 3 7 7 7 2 7 3 6 6 6 6 7 1 7 7 4 1 2 2 6 2 6 6 2 6 1 6 6 1 5
 [236] 7 7 4 6 2 2 3 2 6 2 2 3 2 2 2 1 3 3 1 2 1 1 5 2 2 5 7 5 6 1 2 6 6 6 1 1 1 1 1 5 4 2 6 7 6 6 6 2
 [283] 7 7 6 7 7 2 2 2 6 6 6 3 3 2 4 6 7 2 2 1 1 1 6 2 2 6 6 2 6 6 2 3 7 7 6 2 2 3 6 2 2 2 7 2 6 2 2
 [330] 2 6 2 3 2 2 4 7 6 2 6 7 2 6 7 7 2 7 6 3 7 1 7 7 6 1 1 2 6 6 2 1 6 2 1 1 1 1 1 6 1 6 4 6 3 6 4
 [377] 3 6 3 6 2 1 6 1 6 1 1 2 1 1 1 4 3 7 7 7 6 3 3 2 7 7 3 3 7 2 2 6 3 7 7 3 6 2 6 7 7 2 1 1 7 7 7
 [424] 7 3 3 3 2 2 7 7 6 6 6 6 1 6 1 2 2 6 6 2 6 2 7 7 7 7 3 3 4 4 3 7 7 7 6 6 6 6 6 6 6 4 2 1 1 3 3 7
 [471] 6 1 1 1 6 6 3 3 3 3 6 3 2 1 1 1 5 6 1 1 6 3 3 7 1 3 7 3 6 6 6 6 1 1 6 6 1 1 1 6 6 7 2 2 6 6 1
 [518] 1 1 7 1 1 1 1 2 1 1 3 3 6 7 7 7 7 7 7 1 1 1 6 3 3 1 7 7 6 2 7 1 1 5 1 7 2 1 7 6 3 1 2 4 7 7 1 1
 [565] 1 1 1 3 6 3 6 6 7 3 3 2 2 6 6 6 6 1 6 2 3 3 3 7 7 2 7 1 2 1 1 1 5 6 3 6 7 1 1 5 5 5 6 6 7 7 3
 [612] 6 1 6 6 1 7 7 7 5 3 7 5 1 1 1 1 6 6 7 1 1 6 7 6 6 7 2 3 3 3 3 7 7 7 7 1 1 5 6 6 6 6 5 5 7 7 7
 [659] 7 7 7 7 7 2 4 2 7 6 3 1 7 2 5 3 2 2 4 1 7 2 6 4 5 7 7 7 2 1 1 7 1 1 6 6 7 5 5 7 2 7 6 6 6 6 4
 [706] 6 4 4 6 6 1 2 6 1 2 2 4 7 7 7 7 5 6 6 7 4 7 2 7 2 2 5 7 2 3 5 5 3 3 1 6 2 6 5 1 6 3 6 5 2 6 4
 [753] 6 6 4 4 7 2 5 7 2 7 2 7 7 7 7 7 2 5 3 6 5 6 6 7 7 1 2 2 1 3 4 1 7 1 5 2 7 2 6 7 1 6 3 3 2 7 7 7
 [800] 7 7 4 4 3 4 6 7 1 7 1 2 2 4 4 6 1 3 1 1 2 3 7 6 6 2 2 1 4 1 6 6 6 6 6 6 4 1 1 1 1 1 7 4 7 7 7 7
 [847] 7 7 7 6 2 6 2 2 2 2 2 2 6 1 1 4 7 5 1 6 1 6 1 1 2 1 1 1 7 1 2 1 2 2 1 6 1 6 7 7 2 2 2 2 4 3 3 1 7
 [894] 6 6 7 6 3 3 6 4 7 1 2 2 2 2 6 1 6 2 1 6 2 2 3 3 6 6 2 6 1 1 1 5 6 1 6 1 6 2 6 1 1 5 4 7 6 6 7
 [941] 7 1 1 1 7 7 7 7 7 7 6 6 5 2 2 7 7 6 1 1 1 1 1 1 1 1 2 1 7 7 7 1 6 6 4 1 1 6 1 1 6 1 7 7 2 7 4 7 1 7
 [988] 1 1 1 1 7 6 6 1 6 1 3 1 6
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 115.08772 147.50620  83.58033 115.08835 148.16259 166.56788 108.21979
 (between_SS / total_SS =  44.2 %)

Available components:

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



```
[[8]]
K-means clustering with 8 clusters of sizes 416, 568, 508, 440, 368, 567, 433, 611

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.6600962        0.5074193   0.4937602     0.24356353 0.3332761           0.3752029
2     0.3821083        0.3055773   0.4394666     0.10220729 0.4554515           0.4058899
3     0.4798720        0.5645961   0.3114843     0.22225237 0.6015537           0.4028403
4     0.6181818        0.4686027   0.5041103     0.61148046 0.6118912           0.6342237
5     0.4777853        0.5116747   0.7012026     0.34914669 0.5746797           0.6063135
6     0.6363316        0.3452218   0.5840369     0.08650629 0.4874969           0.3174145
7     0.4548499        0.3657229   0.4046484     0.60066154 0.5647888           0.4419933
8     0.3641980        0.5729527   0.4378591     0.13679887 0.3253449           0.3532531
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.4105748 0.3489948 0.3295218 0.2991453 0.5537135
2            0.4334868 0.3644651 0.7018224 0.5186815 0.3928333
3            0.5729636 0.5005939 0.5235422 0.5109361 0.2472441
4            0.7040691 0.7438108 0.3691032 0.5399411 0.1551855
5            0.7071744 0.5804805 0.4973928 0.5298913 0.2168447
6            0.4004297 0.3535431 0.4169646 0.5840355 0.4294431
7            0.4786399 0.6801067 0.4931964 0.4141220 0.2212113
8            0.3688420 0.2225137 0.5655107 0.4498697 0.6696813

Clustering vector:
   [1] 6 6 6 3 7 3 4 4 3 3 7 3 7 7 3 5 4 7 7 4 4 7 3 4 3 3 4 4 4 7 1 8 5 7 3 4 2 6 5 3 4 4 3 4 7 3 4
  [48] 5 5 8 4 2 4 4 4 4 4 5 7 2 7 7 1 7 7 1 1 7 7 4 1 5 3 5 4 1 4 7 5 6 4 4 3 4 3 3 1 1 6 6 1 7 8 2
  [95] 5 4 3 1 8 5 1 7 5 4 1 6 6 4 5 6 4 5 3 3 2 2 7 7 7 7 4 3 5 3 7 3 2 3 5 1 6 6 8 4 4 2 7 3 5 4 2
 [142] 7 3 3 3 3 7 6 7 7 7 4 3 3 7 7 3 3 3 3 3 3 3 5 5 3 1 4 3 3 4 4 4 4 1 3 3 7 5 4 5 2 3 6 2 3 3 7
 [189] 3 7 4 7 4 4 3 3 5 4 4 5 3 6 3 3 6 4 4 4 4 2 4 4 3 3 3 3 3 5 5 5 5 8 3 6 6 3 1 3 3 1 3 5 2 2 4 2
 [236] 5 5 1 3 6 6 4 6 2 6 1 3 1 6 6 3 4 4 3 6 7 4 2 1 6 8 5 8 2 4 6 3 6 6 4 5 4 4 2 3 6 2 5 3 2 3 6
 [283] 5 5 5 5 5 6 1 6 3 3 2 7 7 1 1 2 5 6 6 5 5 5 2 6 6 3 2 6 2 2 6 7 5 5 3 6 6 4 2 6 6 6 5 6 3 6 6
 [330] 6 3 6 4 6 6 8 1 6 6 6 5 6 2 5 5 6 5 3 4 5 3 4 5 5 7 7 1 2 3 6 4 6 6 3 4 3 4 5 3 7 3 1 3 4 3 8
 [377] 7 3 7 3 1 7 6 7 2 3 3 6 5 7 7 1 7 4 4 4 2 7 4 6 4 4 7 7 5 1 6 6 7 5 5 7 3 1 3 4 4 2 3 3 5 5 5
 [424] 5 4 4 4 2 6 5 4 2 2 2 2 3 3 7 6 1 3 2 6 2 6 5 5 5 7 7 1 1 7 5 5 5 2 6 3 3 3 3 3 3 6 3 3 7 7 5
 [471] 3 7 7 4 3 3 4 7 7 7 3 7 6 7 7 3 6 8 3 3 7 7 7 4 7 7 1 4 3 3 3 3 7 3 3 3 3 3 3 3 6 3 5 6 6 6 3 3
 [518] 3 3 5 4 3 3 3 6 4 4 4 4 3 4 4 4 4 4 7 3 3 3 4 4 3 5 5 3 1 4 3 3 2 4 4 6 7 5 3 4 3 6 3 4 5 3 3
 [565] 3 3 3 4 3 4 3 3 4 7 7 6 6 3 3 3 3 3 2 1 7 7 7 5 5 6 5 4 6 4 3 7 2 3 4 3 5 7 4 2 2 2 3 3 5 5 4
 [612] 6 7 3 3 4 5 5 5 8 7 5 8 4 7 7 7 3 2 5 7 5 3 5 3 3 5 6 7 7 7 7 4 4 5 4 7 3 8 2 3 6 6 1 8 5 5 5
 [659] 5 5 5 5 5 6 8 6 5 7 7 7 5 6 2 4 6 6 1 7 5 1 3 8 2 5 5 5 1 7 3 5 3 3 3 7 4 2 2 4 6 5 3 2 3 3 1
 [706] 2 1 1 6 3 7 2 3 7 6 6 8 5 5 5 5 8 3 3 5 1 5 6 4 6 6 2 5 6 4 2 2 7 7 3 2 2 2 2 2 5 2 7 3 8 6 3 1
 [753] 3 3 5 8 5 2 2 5 2 5 6 5 5 5 5 6 2 7 2 8 3 2 5 5 5 1 1 5 4 1 3 4 3 6 6 4 6 3 5 4 2 4 4 6 5 5 5
 [800] 5 5 3 3 4 1 2 5 3 5 3 6 6 1 1 3 7 7 7 4 1 7 5 3 3 6 6 7 1 7 2 2 2 2 2 2 2 1 4 4 4 4 5 8 5 5 4 5
 [847] 5 5 5 2 6 6 6 6 6 6 6 6 6 7 3 1 5 8 3 3 3 3 7 7 6 4 4 4 5 7 6 4 6 6 2 3 2 3 5 4 1 1 1 1 4 4 4 5
 [894] 3 2 5 3 7 7 2 1 4 7 6 6 6 6 3 3 3 6 3 3 1 6 7 7 3 3 6 3 3 3 3 8 3 3 3 3 3 6 2 3 3 5 3 4 3 3 5
 [941] 4 3 3 3 3 5 5 5 5 6 6 6 6 6 5 4 3 4 4 4 4 4 3 7 7 6 7 4 4 3 3 3 8 4 4 5 7 7 3 7 4 4 6 4 1 5 3 5
 [988] 3 3 3 3 5 6 3 3 6 7 4 7 6
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1]  94.09249 118.78444 115.82027  95.51746  76.20082 126.21904  95.88770 128.82675
 (between_SS / total_SS =  46.3 %)

Available components:

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



```
[[9]]
K-means clustering with 9 clusters of sizes 433, 390, 361, 395, 435, 518, 427, 553, 399

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4920901        0.5040915   0.6276350     0.34875677 0.5714698           0.6231217
2     0.6487821        0.5209243   0.4844517     0.29657321 0.3179487           0.3678488
3     0.4721607        0.6328101   0.3257500     0.19948482 0.6123368           0.3278231
4     0.6856962        0.3464291   0.6122273     0.08612327 0.5663653           0.3161927
5     0.4339655        0.4458919   0.3265346     0.55962509 0.5612069           0.5483654
6     0.3780405        0.3036846   0.4153865     0.12301086 0.4701806           0.4154591
7     0.6275761        0.4005551   0.5675918     0.63852897 0.6281783           0.5420633
8     0.3574593        0.5582792   0.4397291     0.12429231 0.3278869           0.3611118
9     0.5159461        0.3818806   0.5115981     0.10347715 0.3758056           0.3489568
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.7299737 0.5793392 0.5087697 0.5648362 0.2308407
2            0.4054641 0.3804282 0.3324324 0.3090218 0.5435013
3            0.5200740 0.4799220 0.5054279 0.5069765 0.2537953
4            0.4118512 0.3812351 0.4219637 0.4389592 0.3814419
5            0.5755659 0.6726088 0.5355701 0.4264794 0.2064738
6            0.4462874 0.3853152 0.7039027 0.4867367 0.3691919
7            0.6344099 0.7581537 0.3400532 0.5234626 0.1461412
8            0.3760039 0.2141749 0.5710131 0.3933427 0.6669421
9            0.3828676 0.3029704 0.4635914 0.7426901 0.5241120

Clustering vector:
   [1] 9 4 9 3 5 3 7 7 3 6 5 1 5 7 3 1 7 5 5 7 7 5 3 7 3 3 7 7 7 4 2 8 1 5 5 5 6 4 7 3 1 7 3 7 5 3 7
  [48] 1 1 8 7 6 1 5 1 5 5 7 5 6 5 5 2 5 5 2 2 7 7 5 2 1 3 1 7 2 7 5 1 4 5 7 1 7 3 3 2 2 4 4 2 5 8 6
  [95] 1 7 3 2 8 1 2 5 1 7 2 4 4 5 7 4 7 7 3 3 6 6 3 3 5 5 7 1 1 3 5 3 3 3 4 2 9 9 8 7 7 6 5 3 1 7 6
 [142] 7 3 3 5 5 5 4 7 7 5 7 3 3 5 5 5 3 3 6 3 3 3 1 1 3 3 5 3 3 3 5 7 7 7 2 1 3 5 1 7 1 6 3 4 6 3 3 5
 [189] 3 5 7 5 5 5 3 3 1 1 5 1 3 4 3 6 4 7 7 7 7 4 1 7 3 3 3 3 1 1 1 1 8 1 4 2 3 4 3 3 4 3 1 6 6 5 9
 [236] 1 1 2 3 9 9 7 4 6 4 4 7 2 4 4 1 7 7 3 9 5 5 6 2 4 3 1 3 6 5 4 3 4 4 5 5 5 5 6 3 4 6 1 6 6 4 4
 [283] 1 1 1 1 1 4 2 9 6 3 6 2 2 2 2 9 1 4 4 1 1 1 6 4 4 3 6 4 4 4 4 5 1 1 3 4 4 7 6 4 4 9 1 4 3 4 4
 [330] 4 3 9 7 4 4 8 2 9 4 4 3 4 4 1 1 4 1 3 7 1 5 1 3 3 5 5 2 6 3 4 1 4 4 5 5 5 5 1 3 5 4 2 3 7 3 9
 [377] 7 3 7 3 2 5 9 5 6 5 5 4 1 5 5 2 7 7 7 7 7 6 7 7 9 7 7 5 5 1 2 4 9 7 1 1 7 6 2 4 2 2 6 3 3 1 1 1
 [424] 1 7 7 7 6 4 1 1 6 6 6 4 5 3 5 4 2 3 6 4 6 4 7 1 1 5 5 2 2 5 1 3 1 6 4 4 3 1 3 3 3 4 5 5 7 7 7
 [471] 3 5 5 5 3 3 7 5 5 5 3 5 4 5 5 3 9 3 1 1 6 7 7 7 5 7 2 7 3 3 3 3 5 3 3 3 3 5 5 5 4 3 1 9 9 4 3 5
 [518] 5 5 1 5 5 5 5 9 1 1 5 5 3 7 7 7 7 7 5 5 3 3 7 7 5 2 2 4 2 7 1 1 6 5 7 4 5 1 3 7 5 4 3 7 1 5 5
 [565] 5 3 3 7 1 7 1 1 7 5 5 4 4 3 3 4 3 5 6 2 7 7 7 1 7 3 1 5 4 5 5 5 6 3 7 3 7 5 5 6 6 6 3 6 1 1 7
 [612] 9 5 3 3 5 1 1 1 8 5 1 8 5 5 5 5 3 6 1 5 1 3 1 3 3 1 4 5 5 5 5 7 7 1 7 5 5 8 6 3 4 4 9 8 1 1 1
 [659] 1 1 1 1 1 9 8 4 1 5 5 5 1 4 6 7 4 4 2 5 1 2 6 3 9 1 1 1 4 5 5 1 5 5 4 6 1 6 6 1 9 1 3 6 3 3 2
 [706] 6 2 2 4 3 5 6 3 5 4 4 8 1 1 1 1 8 6 3 1 2 1 4 7 9 9 6 1 9 5 6 6 7 7 3 6 6 6 6 1 6 5 3 8 4 3 2
 [753] 4 4 1 8 1 4 6 1 4 1 4 1 1 1 1 4 6 7 6 8 3 6 1 1 1 2 2 1 7 2 3 7 3 9 9 7 4 3 1 5 6 7 7 4 1 1 1
 [800] 1 1 3 3 7 2 6 1 3 1 3 9 9 2 2 3 2 5 5 5 2 5 1 3 3 4 4 5 2 5 6 6 6 6 6 6 2 5 5 5 5 1 9 1 1 7 1
 [847] 1 1 1 6 4 9 9 4 9 4 4 9 5 3 2 1 8 3 3 3 3 5 5 4 5 5 5 1 5 4 1 4 4 6 3 6 3 1 7 4 4 2 8 7 7 5 3
 [894] 3 6 3 3 5 5 6 2 7 5 4 9 4 4 3 3 3 4 3 3 2 4 5 5 3 3 4 1 5 5 3 8 3 3 3 3 3 4 6 5 5 9 3 7 6 6 1
 [941] 1 5 5 5 3 1 1 1 1 1 4 4 9 9 4 1 7 3 1 1 1 1 1 3 5 5 9 5 7 7 3 3 3 8 5 5 1 5 5 3 5 7 7 4 7 2 1 5 1
 [988] 5 3 5 5 1 4 3 5 4 5 7 5 3
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1]  90.86669  89.54868  81.63419  81.14236  88.95072 105.53334  94.76586 110.69488  81.18094
 (between_SS / total_SS =  48.0 %)

Available components:

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



```
[[10]]
K-means clustering with 10 clusters of sizes 391, 373, 324, 357, 405, 485, 348, 421, 433, 374

Cluster means:
   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1      0.6644501        0.4562534   0.5098765     0.22401224 0.3361801           0.3757432
2      0.3967828        0.4957469   0.2957618     0.19783393 0.5609920           0.3035061
3      0.7020062        0.3578342   0.6232598     0.10192396 0.5964506           0.3099246
4      0.5142857        0.5240516   0.7526670     0.45277363 0.5704282           0.6403834
5      0.2947531        0.3927145   0.4566325     0.10380178 0.3336420           0.3725349
6      0.5050515        0.5418608   0.3474007     0.45906638 0.5766568           0.6015129
7      0.4505029        0.7054775   0.4258988     0.17007734 0.3449302           0.3500149
8      0.5543943        0.3586112   0.4668722     0.66994472 0.6235579           0.4642163
9      0.4599307        0.3707125   0.4506904     0.15765902 0.5275487           0.4766654
10     0.5107286        0.3340595   0.5169530     0.08637988 0.3743793           0.3348844
   total.sulfur.dioxide   density        pH sulphates   alcohol
1             0.4016015 0.3443092 0.3270892 0.3014114 0.5420158
2             0.4316822 0.4512037 0.5068111 0.4257273 0.2626637
3             0.4340935 0.4217068 0.4538705 0.4579904 0.3401625
4             0.7079067 0.6426204 0.4438262 0.5437286 0.1882385
5             0.3688716 0.2365100 0.6514848 0.3848651 0.5905776
6             0.6679467 0.6329307 0.5035665 0.4947690 0.2201149
7             0.3797244 0.2422162 0.4968158 0.4737654 0.6898946
8             0.5582518 0.7551245 0.3797586 0.4475235 0.1508109
9             0.5766966 0.4411101 0.7109419 0.6023437 0.3567997
10            0.3639983 0.3031140 0.4585561 0.7150921 0.4927393

Clustering vector:
   [1] 10  3 10  2  6  2  8  8  6  9  8  6  8  8  6  4  8  6  6  8  8  6  9  4  6  6  4  4  8  3  1
  [32]  5  4  6  6  6  2  3  8  2  4  4  8  8  6  6  8  4  4  2  8  9  6  6  6  6  6  4  8  2  8  6
  [63]  1  6  6  1  1  8  8  6  1  4  2  4  8  1  8  6  4  3  6  6  6  8  6  9  1  1  3 10  1  2  5
  [94]  9  9  4  2  1  7  4  1  8  4  8  1 10 10  6  4  3  4  4  2  9  5  2  8  8  8  6  4  9  4  6
 [125]  6  6  9  9  3  1 10 10  5  4  4 10  6  2  4  4  2  8  6  6  6  6  6 10  8  8  6  8  9  9  6
 [156]  6  6  6  2  9  2  2  2  4  4  2  1  6  6  2  6  4  4  4  1  6  2  6  4  8  4  9  2  3  9  6
 [187]  6  6  2  6  4  6  6  6  2  6  4  4  6  4  2  3  2  2  3  8  4  4  4  3  4  8  2  2  2  2  4
 [218]  6  4  4  7  6  3  1  2  3  2  2  1  2  6  9  9  6 10  4  4  1  2 10 10  4  3  2  1  3  8  1
 [249]  1  3  6  8  8  6  3  6  6  9  1  3  2  6  2  2  6  3  2  3  3  6  6  6  6  9  2  3  9  4  2
 [280]  9  3  3  4  4  3  3  3  3  1 10  2  2  3  8  8  1  1  9  4  3  3  9  9  9  9  1  1  2  2  3
 [311]  3  3  3  8  4  4  2  3  3  8  2  3  3 10  4  3  2  3  3  3  2 10  8  3  3  7  1 10  3  3  4
 [342]  3  3  4  4  3  6  2  8  9  6  4  3  3  8  6  1  9  2  3  6  3  3  6  6  6  6  6  9  6  3  1
 [373]  2  8  2  7  8  2  8  2  1  6 10  6  9  6  6  3  6  8  8  1  8  4  4  4  9  8  8 10  4  4  8
 [404]  8  4  1  3 10  8  4  4  8  2  1  3  6  6  3  6  6  4  4  4  4  8  8  6  5  3  4  6  9  9  9
 [435]  9  6  2  6  3  1  2  2  3  2  3  3  4  4  8  8  1  1  8  4  2  4 10  3  3  3  9  2  2  2  3
 [466]  6  6  8  8  4  2  8  8  6  2  2  4  8  8  8  2  8 10  6  6  6 10  2  9  9  2  8  8  4  8  8
 [497]  6  8  2  2  2  2  6  2  2  2  6  2  6  3  3  6 10 10  3  2  6  6  6  9  6  6  6  6 10  6  6
 [528]  8  8  2  6  6  6  6  6  8  6  6  2  8  8  6  4  4  3  1  6  6  6  9  6  6  3  6  4  9  8  6
 [559]  3  2  4  4  6  6  6  6  6  8  9  8  9  9  6  8  8  3  3  2  2  3  2  6  9  1  8  8  8  4  4
 [590] 10  4  6  3  6  6  6  2  2  4  2  4  6  6  5  5  5  2  9  4  4  8 10  6  9  9  6  4  4  4  5
 [621]  8  4  5  6  6  6  8  2  9  4  6  9  3  4  2  2  4  3  8  8  8  8  4  4  4  4  8  6  5  2  2
 [652]  3  3 10  5  4  4  4  4  4  4  4  4 10  7  3  4  2  8  6  4  3  5  8  3  1  1  6  4  1  2  7
 [683] 10  4  4  4  1  6  6  4  6  6  3  2  6  5  5  6 10  4  2  2  2  2  1  2  1  1  2  9  6 10  9
 [714]  6  3  3  7  4  4  4  4  5  9  2  4  1  4  3  4 10 10  5  3 10  6  5  5  8  8  2  5  1  5  5
 [745]  6  9  8  2  5  3  2  1  3  3  1  7  6  2  9  6  2  4  1  4  4  4  4  1  9  8  9  5  2  9  4
 [776]  4  6  1  1  6  8  1  2  4  2 10 10  4  3  2  4  6  9  4  4  3  4  6  4  6  4  2  2  8  1  9
 [807]  4  6  4  6 10 10  1  1  2  1  8  6  6  1  8  4  3  3  3  3  8  1  6  2  9  9  2  9  9  7  6
 [838]  6  6  6  4  7  4  4  4  4  4  4  4  5  3 10 10  3 10  3  3 10  6  6  2  4  5  6  2  6  2  6
 [869]  6  3  6  6  6  4  8  3  6  3  3  2  2  2  2  4  6  3  3  1  5  4  4  6  6  2  2  6  2  8  8
 [900]  2  1  4  6  3 10  3  3  2  6  2  3  6  2  1  3  8  8  2  2  3  9  6  6  6  5  6  6  2  6  9
 [931]  3  2  6  6 10  7  4  2  2  6  6  6  6  6  6  4  4  4  4  3  3 10 10  3  4  4  2  6  6  6  6
 [962]  6  8  8 10  8  4  4  2  2  2  7  6  6  3  8  8  2  8  4  4  3  4  1  6  6  6  6  2  6  6  6
 [993]  3  9  6  3  6  8  6  2
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
 [1] 83.15265 80.44538 64.79483 76.35033 77.44474 99.40955 67.70087 89.61393 93.38195 70.57477
 (between_SS / total_SS =  49.3 %)

Available components:

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

Looking at the ratio between the sum of squares and the overall sum of squares is helpful. So, when there is just one cluster, this value equals zero, which isn't very good, but when clustering with two separate clusters is done, this value rises to 25.4 percent, and when three clusters are assigned, this value rises to 32.3 percent, indicating that as we add clusters, this value rises.

But what we're doing here is plotting this percentage against the amount of clusters we have, and if we see a sort of shoulder in the plot, that's generally a really good indicator of how many clusters we should be looking for.

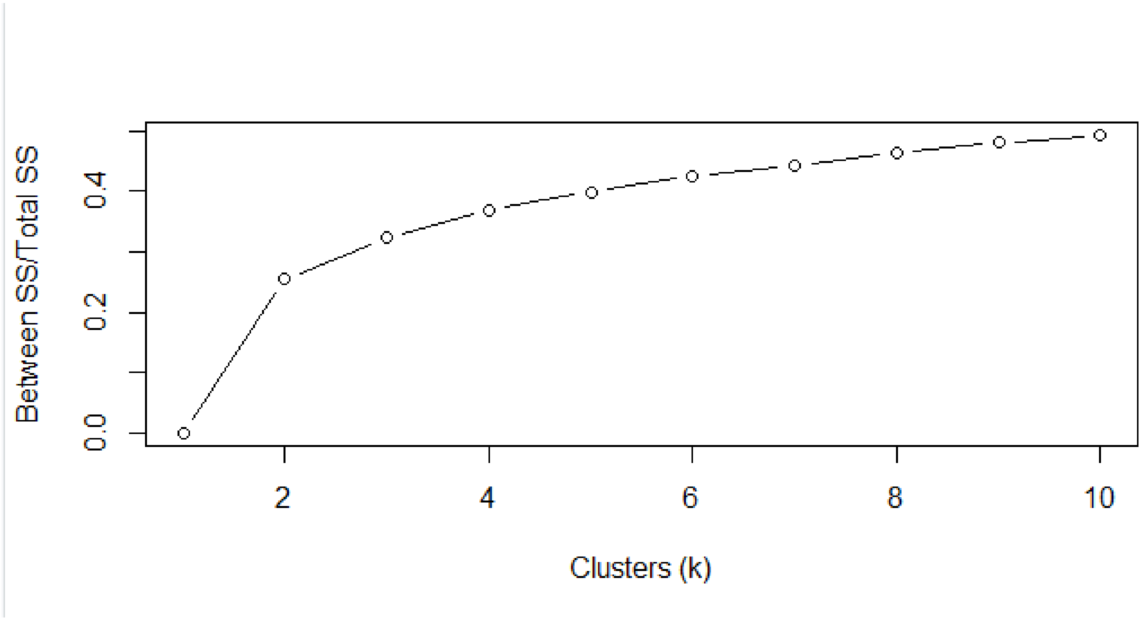
```
#  bss/tss Calculation
# list to store the val
betweenSS_totss <- list()
for (i in 1:10) {
  betweenSS_totss[[i]] <- kval[[i]]$betweenss/kval[[i]]$totss
}
```

For each k-means model, the above code is used to calculate the bss divide by the tss.

The ratio of the sum of squares is then shown against the number of clusters in our data.

```
# Plotting the sum of squares ratio vs the number of clusters
plot(1:10, betweenSS_totss, type = "b",
     ylab = "Between SS/Total SS", xlab = "Clusters (k)")
```

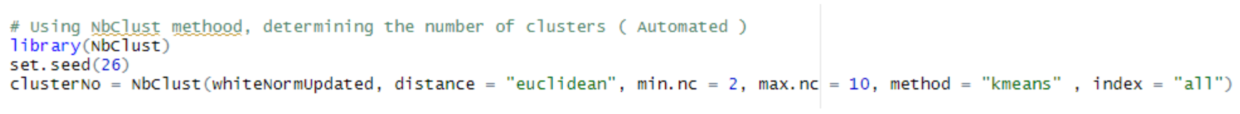
Graph results are below,



The best method to interpret this is to determine the optimal number of clusters when the plot has a shoulder. Using this strategy, the best number of clusters is selected as 2 based on the preceding graph and theory.

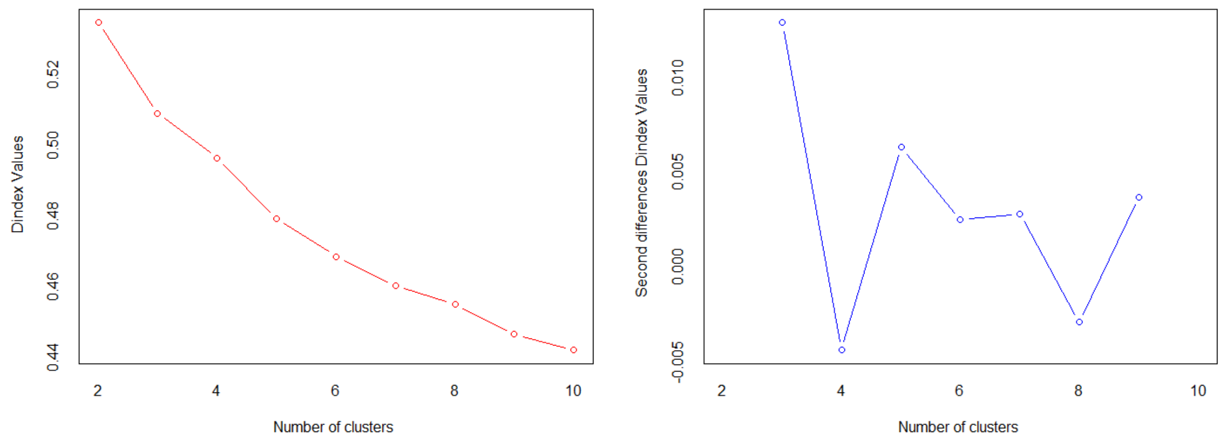
NbClust is a method to calculate the number of clusters.

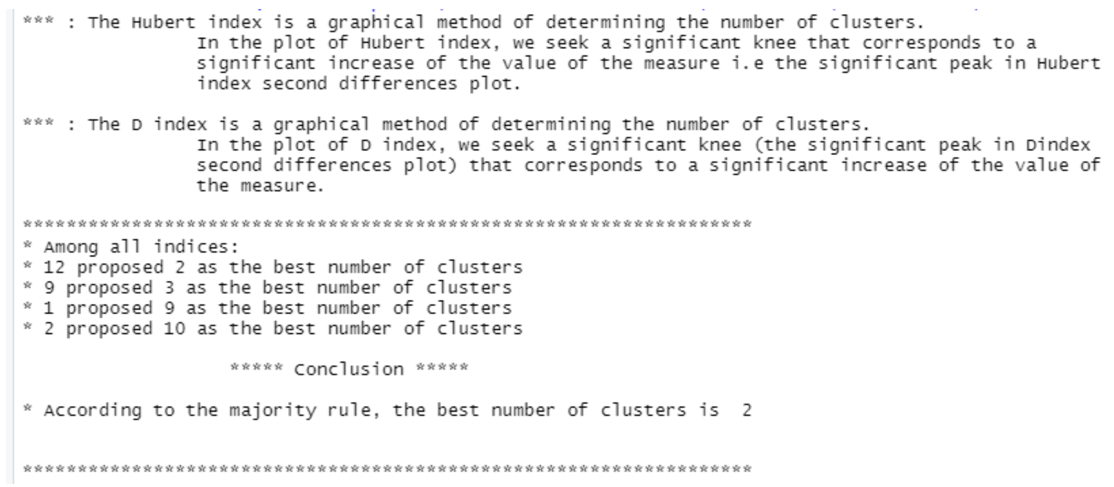
The NbClust software suggests 30 indices for calculating the number of clusters to the user. based on the varied findings achieved by altering the number of clusters, distance measurements, and clustering algorithms Distance is measured in this project using Euclidean distance, however Manhattan distance can also be used.



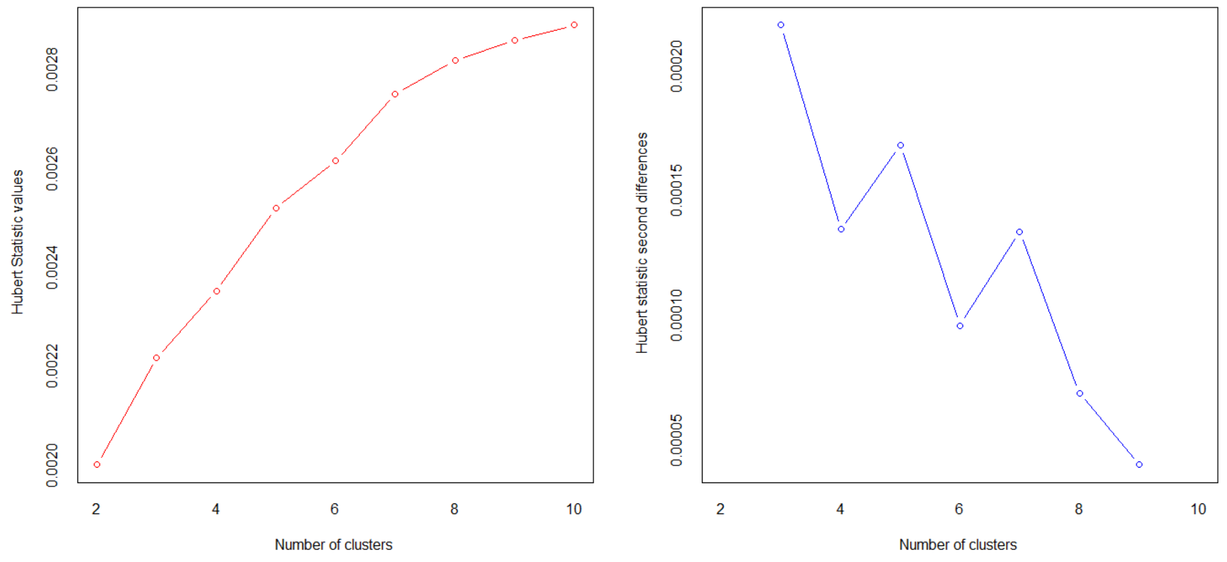
```r
# Using NbClust methood, determining the number of clusters ( Automated )
library(NbClust)
set.seed(26)
clusterNo = NbClust(whiteNormUpdated, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans" , index = "all")
```

Following conclusion and the graph were given as the output,





```
*** : 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:
* 12 proposed 2 as the best number of clusters
* 9 proposed 3 as the best number of clusters
* 1 proposed 9 as the best number of clusters
* 2 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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


*******************************************************************
```



The Hubert index is a graphical approach for calculating cluster size. In the Hubert index second differences plot, we look for a substantial knee that corresponds to a large increase in the measure's value, i.e., the significant peak.

The D index is a graphical way for calculating cluster size. We look for a major knee (the substantial peak in the Dindex second differences plot) in the D index plot that correlates to a significant increase in the measure's value.

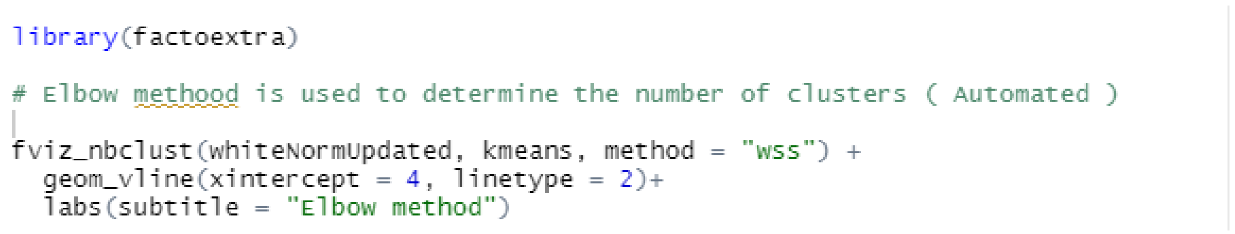
And, after running all indices(seed value), the outcome was that 12 offered 2 as the best number of clusters, and 8 proposed 3 as the best number of clusters.

The optimal number of clusters is two, according to this approach.

Counting clusters using the elbow technique

WSS is the sum of squared distances between each point and the centroid of a cluster. When we plot the WSS with the K value, the plot appears like an Elbow. As the number of clusters grows, the WSS value decreases.

Code,

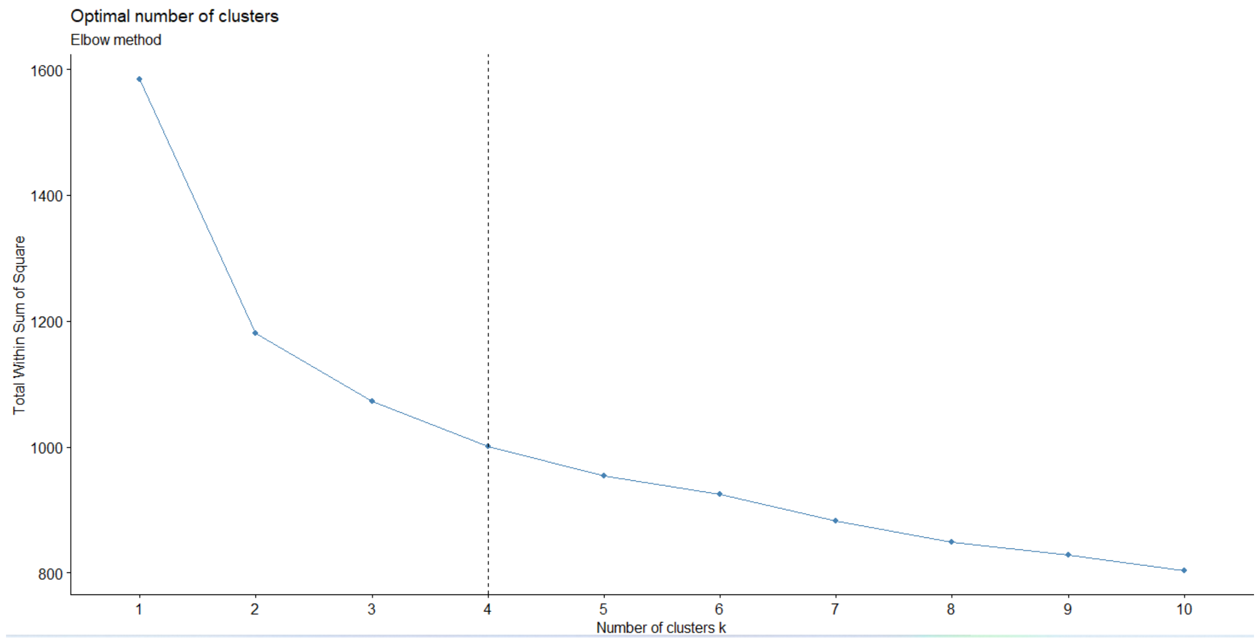


```
library(factoextra)

# Elbow methood is used to determine the number of clusters ( Automated )

fviz_nbclust(whiteNormUpdated, kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2)+
  labs(subtitle = "Elbow method")
```

The number of clusters (K) in the Elbow method is modified between 1 and 10. For each value of K, we calculate WSS. (Within-Cluster Sum of Square). WSS is the sum of squared distances between each point and the centroid of a cluster. When we plot the WSS with the K value, the plot appears like an Elbow. As the number of clusters grows, the WSS value decreases. The WSS value is greatest when K = 1. We can see from the graph that it will shift quickly at one point, generating an elbow shape.

At this point, the graph begins to travel practically parallel to the X-axis. This point corresponds to the ideal K value, or the optimal number of clusters.

The elbow form is attained at the point of 4, thus we can infer that the optimal number of clusters is 4 using this approach.



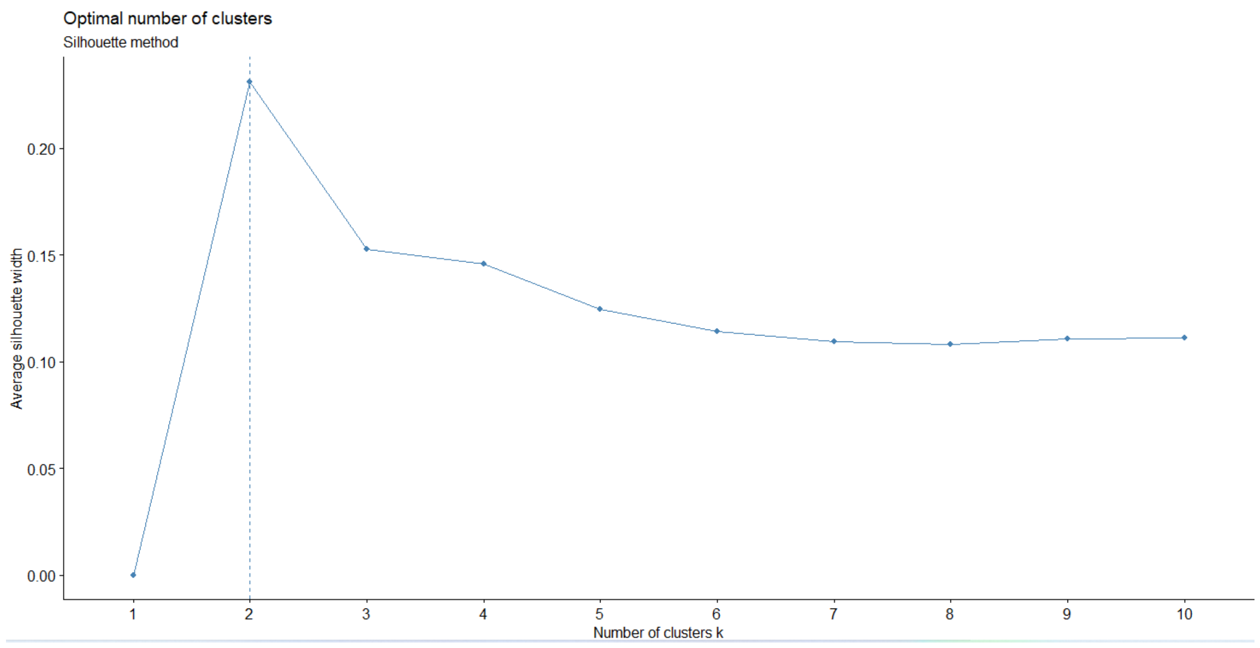
Method of determining the number of clusters using silhouettes

The silhouette Method is also used to calculate the ideal number of clusters, as well as to understand and confirm data cluster consistency. The silhouette technique produces the silhouette coefficients for each point, which compare how similar a point is to its own cluster to other clusters. by providing a rapid visual picture of each object's classification accuracy.

```
# Using the silhouette approach to determine the number of clusters ( Automated )

fviz_nbclust(whiteNormUpdated, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method")
```

The silhouette graph will be as below,(R output)

Using the silhouette approach, we can establish that the ideal number of clusters is two.



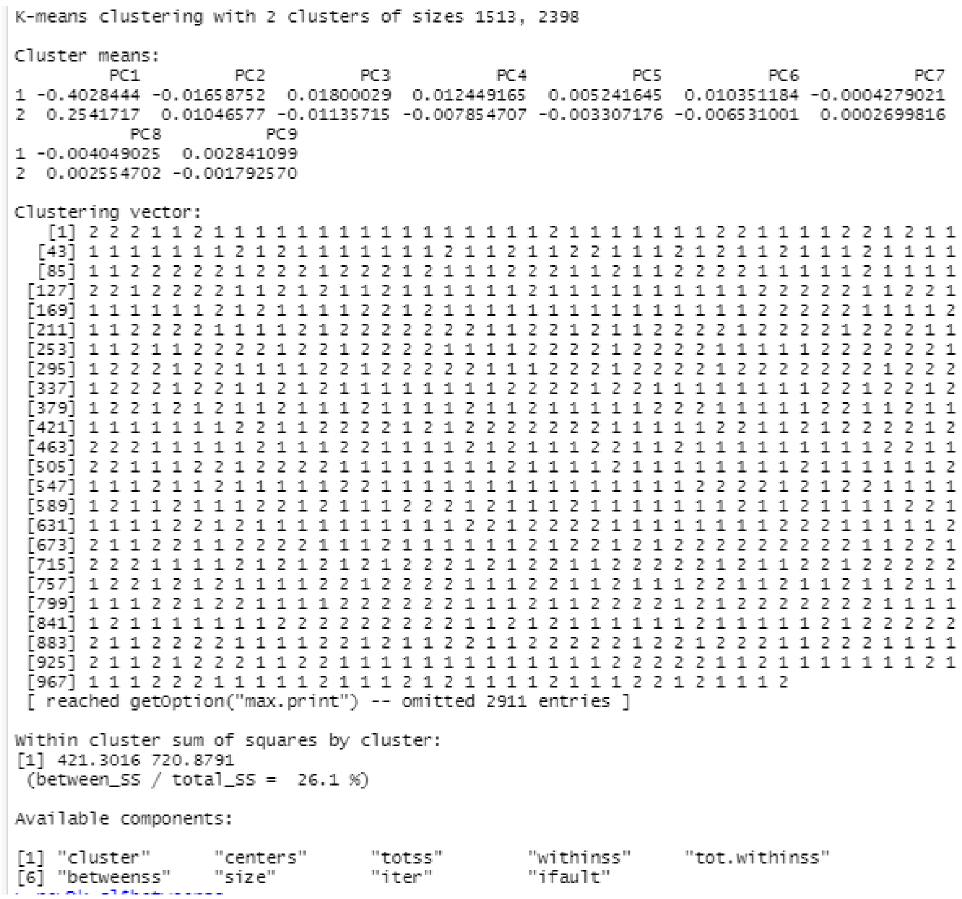
K means calculations with specific K values

K means calculations with specific K values

To make creating the confusion matrix easier, I assigned the values of the classes to a variable named y. Giving the k number of 2 to execute the k-mean clustering algorithm results in a k-mean cluster.

```
# Adding the values of the classes to a variable
y <- updated_whitewine$quality

# Performing k-means with K == 2
k_cl2 <- kmeans(whiteNormUpdated,2)
k_cl2
fviz_cluster(k_cl2, whiteNormUpdated, geom = "point")
```

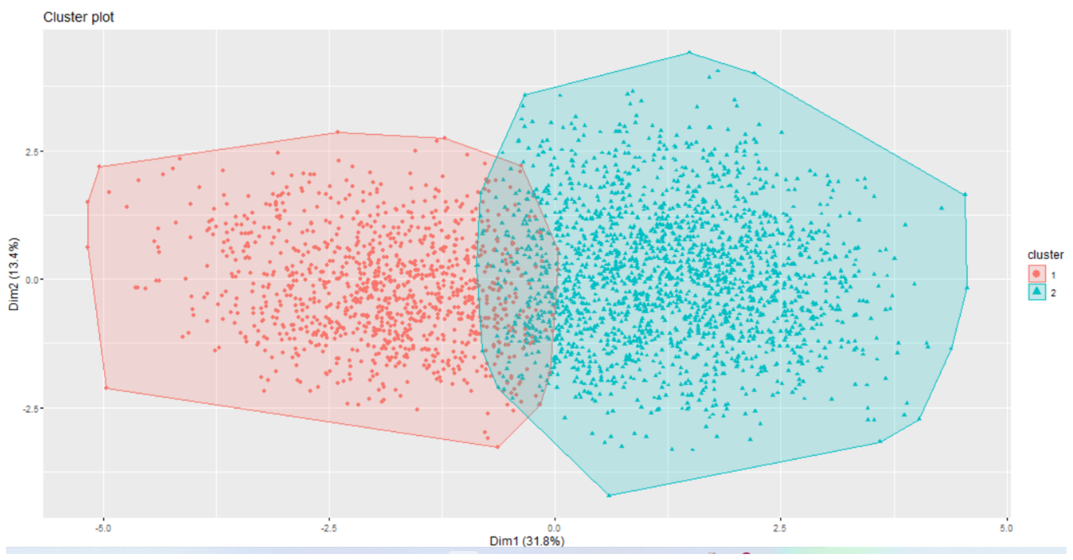
R output,



```
K-means clustering with 2 clusters of sizes 1513, 2398

Cluster means:
         PC1         PC2         PC3          PC4          PC5          PC6           PC7
1 -0.4028444 -0.01658752  0.01800029  0.012449165  0.005241645  0.010351184 -0.0004279021
2  0.2541717  0.01046577 -0.01135715 -0.007854707 -0.003307176 -0.006531001  0.0002699816
          PC8          PC9
1 -0.004049025  0.002841099
2  0.002554702 -0.001792570

Clustering vector:
   [1] 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 1 2 1 1
  [43] 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 2 2 1 1 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1
  [85] 1 1 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 1 1 1 2 2 2 1 1 2 1 1 2 2 2 2 1 1 1 1 1 2 1 1 1 1
 [127] 2 2 1 2 2 2 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 2 1
 [169] 1 1 1 1 1 1 2 1 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 1 1 2
 [211] 1 1 2 2 2 2 1 1 1 1 2 1 2 2 2 2 2 2 2 1 1 2 2 1 2 1 1 2 2 2 2 1 2 2 2 2 1 2 2 2 1 1
 [253] 1 1 2 1 1 2 2 2 2 1 2 2 1 2 2 2 2 1 1 1 1 2 2 2 2 1 2 2 2 2 1 1 1 1 1 2 2 2 2 2 2 1
 [295] 1 2 2 2 1 2 2 1 1 1 1 2 2 1 2 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2
 [337] 1 2 2 2 1 2 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 1 1 1 1 1 1 1 2 2 1 2 2 1 2
 [379] 1 2 2 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 2 1 1
 [421] 1 1 1 1 1 1 1 2 2 1 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 2 1 2 2 2 2 1 2
 [463] 2 2 2 1 1 1 1 1 2 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1
 [505] 2 2 1 1 1 2 2 1 2 2 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2
 [547] 1 1 1 2 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 1 2 2 1 1 1 1
 [589] 1 2 1 1 2 1 1 1 2 2 1 2 1 1 1 2 2 2 1 2 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2 1
 [631] 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 2 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2
 [673] 2 1 1 2 2 1 1 2 2 2 2 1 1 1 2 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1
 [715] 2 2 2 1 1 1 1 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 1 2 2 1 2 2 2 2 2
 [757] 1 2 2 1 2 1 2 1 1 1 1 2 2 1 2 2 2 2 1 1 1 2 2 1 1 2 1 1 1 2 2 1 1 2 1 1 2 1 1 2 1 1
 [799] 1 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 2 2 1 1 1 2 1 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 1
 [841] 1 2 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2 2 2 2
 [883] 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 2 2 1 1 2 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 2 1 1 1 1
 [925] 2 1 1 2 1 2 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 2 1
 [967] 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 2 1 1 1 2 2 1 2 1 1 1 2
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 421.3016 720.8791
 (between_SS / total_SS =  26.1 %)

Available components:

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

When k-mean is conducted with k=2, the result of between sum of square/ total sum of square is 26.1 percent, as indicated in the preceding output.

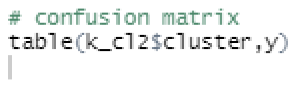
The graphic below shows how the data has been interrupted when there are two clusters.



Confusion Matrix

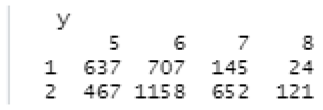
In a confusion matrix, the number of accurate guesses vs the number of wrong predictions is shown. For a binary classifier, this would be the ratio of true negatives and true positives (right predictions) to false negatives and false positives (incorrect predictions).

The confusion matrix may be obtained by executing the following code.



```
# confusion matrix
table(k_cl2$cluster,y)
```

matrix,



```
   y
      5    6    7    8
  1 637  707  145   24
  2 467 1158  652  121
```

According to the aforementioned results, cluster 1 has 467 values from class 5, 1158 values from class 6, 652 values from class 7, and 121 values from class 8, whereas cluster 2 contains 637 values from class 5, 707 values from class 6, 145 values from cluster 7, and 24 values from class 8.

Accuracy

Accuracy is the number of correct predictions your model made over the full test dataset.

When analyzing a model's performance, accuracy is an excellent place to start. Simple accuracy has the disadvantage of working best on well-balanced datasets. In imbalanced datasets, accuracy, on the other hand, becomes a weaker metric.

It is calculated using the formula below.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

K=2;

accuracy = (637 + 1158)/3911
= 0.4589
= 45.89%

Recall

The true positive rate, also known as recall, is a metric that indicates how many true positives are predicted out of all the positives in a dataset. Another term for it is sensitivity. The metric is calculated using the formula below.

$$Recall = \frac{TP}{TP + FN}$$

Precision

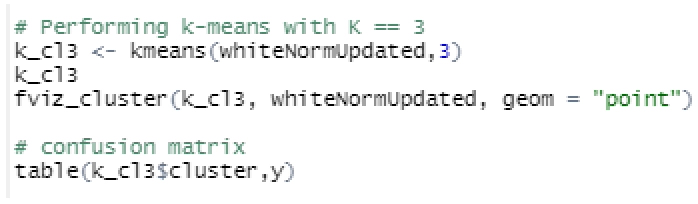
Precision is a statistic used to assess the accuracy of a positive forecast. To put it another way, how certain can you be that a positive outcome is positive if it is projected to be so? It is calculated using the following formula:

$$Pr\,ecision = \frac{TP}{TP + FP}$$

Precision, like recall, may be tweaked by changing the parameters and hyperparameters of your model.
While adjusting, you'll observe that a greater accuracy usually means a lesser recall, while a higher recall means a lower precision.

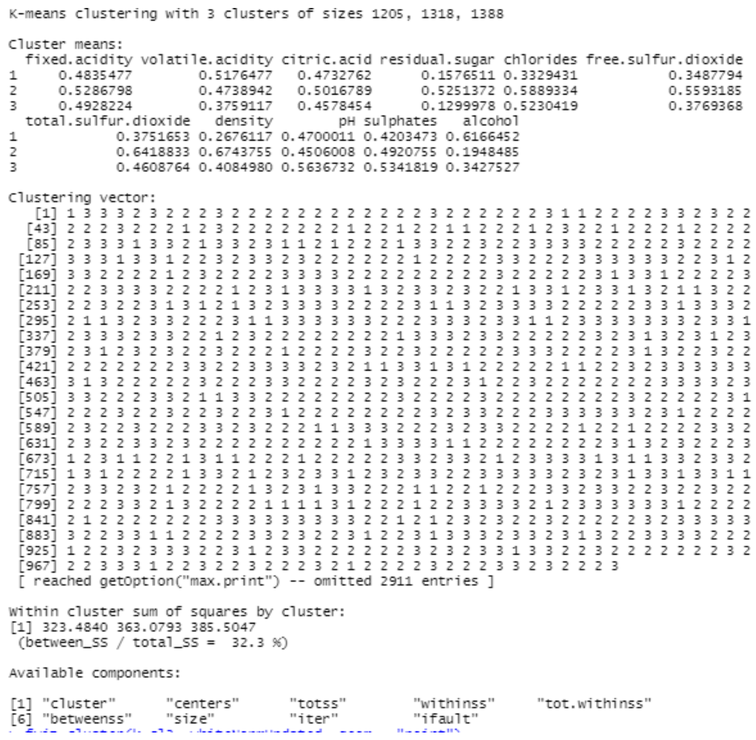
k-mean calculation when k = 3

Code,



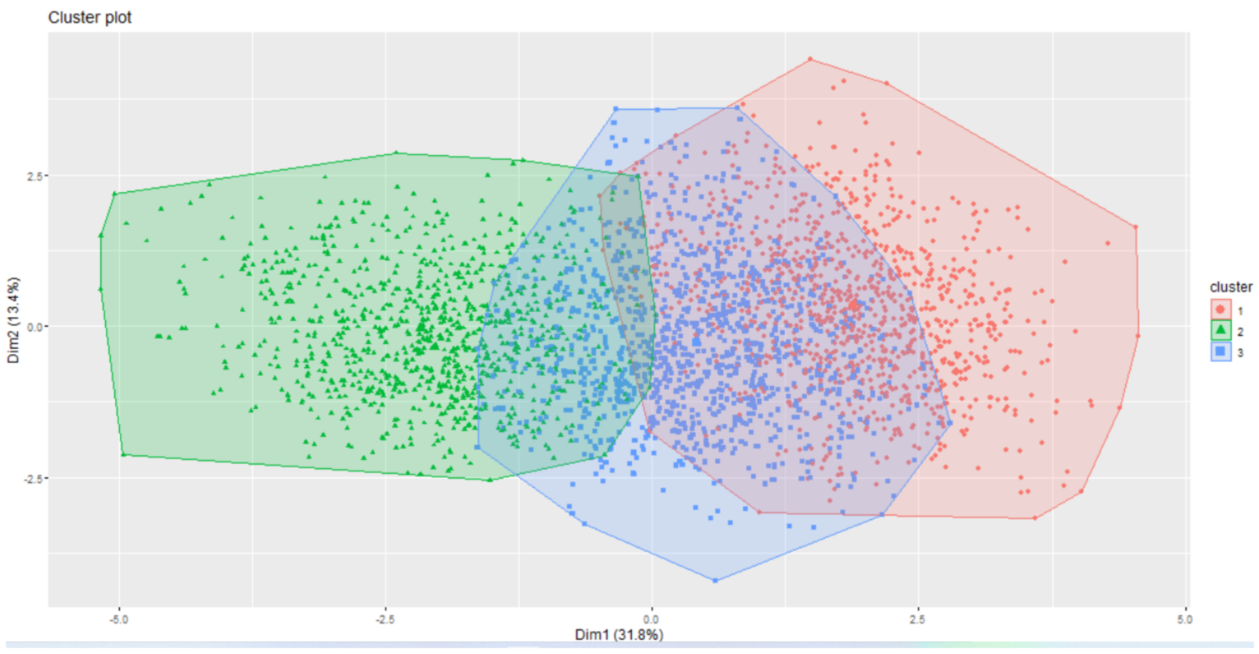
```r
# Performing k-means with K == 3
k_cl3 <- kmeans(whiteNormUpdated,3)
k_cl3
fviz_cluster(k_cl3, whiteNormUpdated, geom = "point")

# confusion matrix
table(k_cl3$cluster,y)
```

R Output,

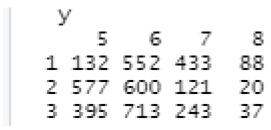


```
K-means clustering with 3 clusters of sizes 1205, 1318, 1388

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4835477        0.5176477   0.4732762      0.1576511 0.3329431           0.3487794
2     0.5286798        0.4738942   0.5016789      0.5251372 0.5889334           0.5593185
3     0.4928224        0.3759117   0.4578454      0.1299978 0.5230419           0.3769368
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.3751653 0.2676117 0.4700011 0.4203473 0.6166452
2            0.6418833 0.6743755 0.4506008 0.4920755 0.1948485
3            0.4608764 0.4084980 0.5636732 0.5341819 0.3427527

Clustering vector:
   [1] 1 3 3 3 2 3 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 3 1 1 2 2 2 2 3 3 2 3 2 2
  [43] 2 2 2 3 2 2 2 1 2 3 2 2 2 2 2 2 2 1 2 2 1 2 2 1 1 2 2 2 1 2 3 2 2 1 2 2 2 1 2 2 2 2
  [85] 2 3 3 3 1 3 3 2 1 3 3 2 3 1 1 2 1 2 2 2 1 3 3 2 2 3 2 2 3 3 3 3 2 2 2 2 2 3 2 2 2 2
 [127] 3 3 3 1 3 3 1 2 2 3 2 3 3 2 3 2 2 2 2 2 2 1 2 2 2 2 3 3 2 2 2 3 3 3 3 3 3 2 2 3 1 2
 [169] 3 3 2 2 2 2 1 2 3 2 2 2 2 3 3 3 3 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 3 1 3 3 1 2 2 2 2 3
 [211] 2 2 3 3 3 3 2 2 2 2 1 2 3 1 3 3 3 3 1 3 2 3 3 2 3 2 2 1 3 3 1 2 3 3 1 3 2 1 1 3 2 2
 [253] 2 2 3 2 2 3 1 3 1 2 1 3 2 3 3 3 3 2 2 2 2 3 1 1 3 2 3 3 3 3 2 2 2 2 2 3 3 1 3 3 3 2
 [295] 2 1 1 3 2 3 3 2 2 2 3 1 1 3 3 3 3 3 3 2 2 2 3 3 3 2 3 3 1 1 2 3 3 3 3 3 3 3 2 3 3 1
 [337] 2 3 3 3 2 3 3 2 2 1 2 3 2 2 2 2 2 2 2 2 1 3 3 3 2 3 3 2 2 2 2 2 3 2 3 1 3 2 3 1 2 3
 [379] 2 3 1 2 3 2 3 2 2 3 2 2 2 1 2 2 2 2 3 2 2 3 2 2 2 2 2 3 3 3 2 2 2 2 3 1 3 2 2 3 2 2
 [421] 2 2 2 2 2 2 2 3 3 2 2 3 3 3 3 2 3 2 1 1 3 3 1 3 1 2 2 2 2 2 1 1 2 2 3 2 3 3 3 3 3 3
 [463] 3 1 3 2 2 2 2 2 3 2 2 2 3 3 2 2 2 2 3 2 3 2 2 2 3 1 2 2 3 2 2 2 2 2 2 2 3 3 3 3 2 3
 [505] 3 3 2 2 2 3 3 2 1 1 3 3 2 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 3 1
 [547] 2 2 2 3 2 2 3 2 2 3 2 2 3 1 2 2 2 2 2 2 2 2 3 2 3 3 2 2 2 3 3 3 3 3 3 2 3 1 2 2 2 2
 [589] 2 3 2 2 3 2 2 2 3 3 2 3 2 2 2 1 1 3 3 3 2 2 2 3 2 3 3 2 2 2 2 1 2 2 1 2 2 2 2 3 3 2
 [631] 2 3 2 2 3 3 2 3 2 2 2 2 2 2 2 2 2 2 1 3 3 3 3 1 1 2 2 2 2 2 2 2 2 3 1 3 2 3 2 2 2 3
 [673] 1 2 3 1 1 2 2 1 3 1 1 2 2 2 1 2 2 2 2 2 3 3 2 3 3 2 1 2 3 3 3 3 1 3 1 1 3 3 2 3 3 2
 [715] 1 3 1 2 2 2 2 1 3 3 2 1 2 3 2 3 3 1 2 3 2 3 3 2 2 3 3 3 3 3 2 3 2 3 1 3 3 1 3 3 1 1
 [757] 2 3 3 2 3 2 1 2 2 2 2 1 3 2 3 1 3 3 2 2 2 1 1 2 2 1 2 2 2 3 3 2 3 3 2 2 3 2 2 3 2 2
 [799] 2 2 2 3 3 2 1 3 2 2 2 2 1 1 1 1 3 1 2 2 2 1 2 2 3 3 3 3 2 1 2 3 3 3 3 3 3 1 2 2 2 2
 [841] 2 1 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 2 2 1 2 1 2 3 2 3 2 2 3 2 2 2 2 2 3 2 3 3 3 3 3
 [883] 3 2 2 3 3 1 1 2 2 2 2 3 3 2 3 2 2 3 1 2 2 3 1 3 3 3 2 3 3 2 3 1 3 2 2 3 3 3 3 2 2 2
 [925] 1 2 2 3 2 3 3 3 2 2 3 1 2 3 3 2 2 2 2 2 2 3 2 3 2 3 3 1 3 3 2 2 3 2 2 2 2 2 2 2 3 2
 [967] 2 2 3 3 3 1 2 2 3 2 2 3 2 2 2 3 2 1 2 2 2 2 3 2 2 2 3 3 2 3 2 2 2 3
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 323.4840 363.0793 385.5047
 (between_SS / total_SS =  32.3 %)

Available components:

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

1205, 1318, and 1388 are the sizes of the three clusters and the value of between_sum_fo_square/ total_sum_of_square when k-mean is runned with k=3 is 32.3% as shown in the above output.



Confusion Matrix



```
   y
     5   6   7   8
  1 132 552 433  88
  2 577 600 121  20
  3 395 713 243  37
```

K = 3 ;

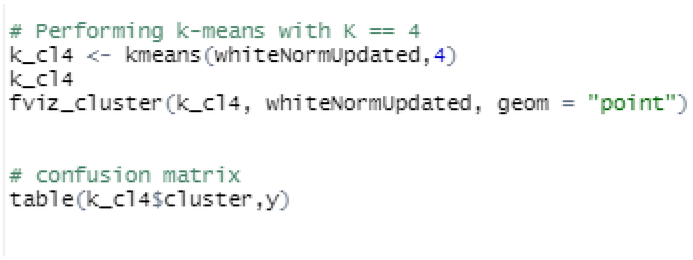
accuracy = (132 + 600 + 243)/3911

= 0.2492

= 24.92%

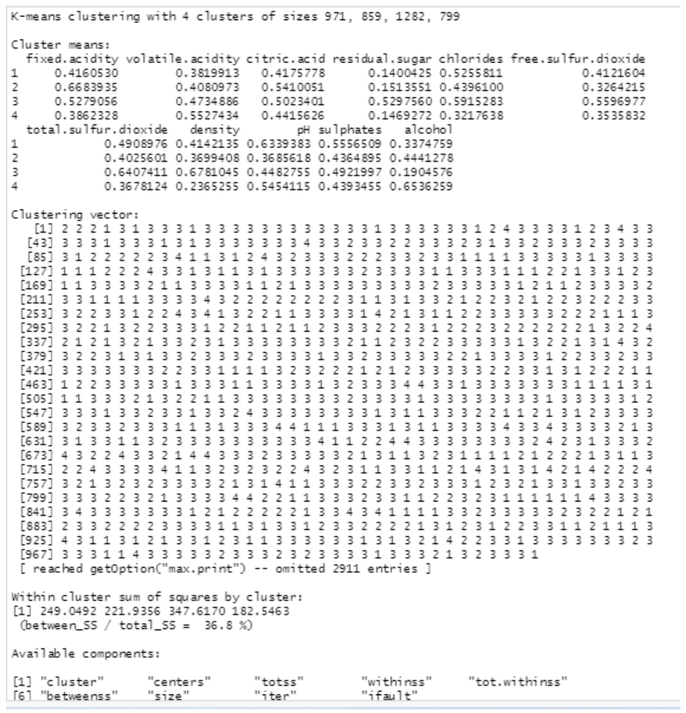
k-mean calculation for k =4

Code,



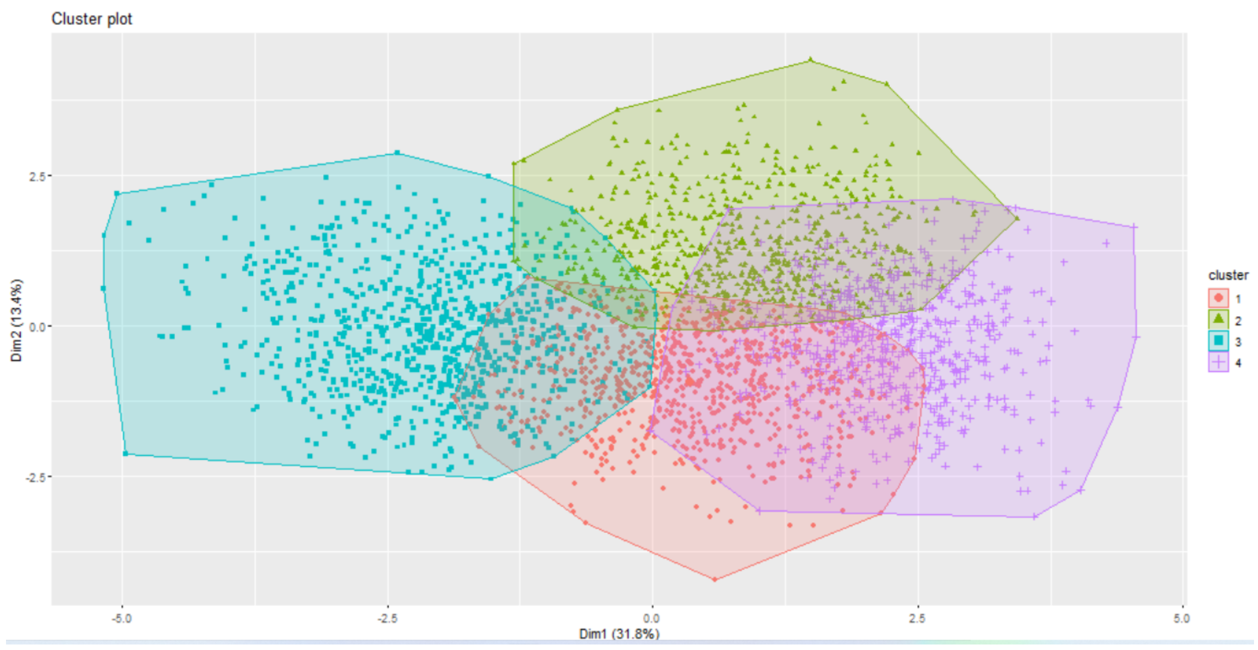
```
# Performing k-means with K == 4
k_cl4 <- kmeans(whiteNormUpdated,4)
k_cl4
fviz_cluster(k_cl4, whiteNormUpdated, geom = "point")


# confusion matrix
table(k_cl4$cluster,y)
```

R Output,

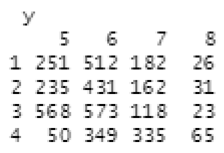


```
K-means clustering with 4 clusters of sizes 971, 859, 1282, 799

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar chlorides free.sulfur.dioxide
1     0.4160530        0.3819913   0.4175778      0.1400425 0.5255811           0.4121604
2     0.6683935        0.4080973   0.5410051      0.1513551 0.4396100           0.3264215
3     0.5279056        0.4734886   0.5023401      0.5297560 0.5915283           0.5596977
4     0.3862328        0.5527434   0.4415626      0.1469272 0.3217638           0.3535832
  total.sulfur.dioxide   density        pH sulphates   alcohol
1            0.4908976 0.4142135 0.6339383 0.5556509 0.3374759
2            0.4025601 0.3699408 0.3685618 0.4364895 0.4441278
3            0.6407411 0.6781045 0.4482755 0.4921997 0.1904576
4            0.3678124 0.2365255 0.5454115 0.4393455 0.6536259

Clustering vector:
   [1] 2 2 2 1 3 1 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 1 2 4 3 3 3 3 1 2 3 4 3 3
  [43] 3 3 3 1 3 3 3 1 3 1 3 3 3 3 3 3 3 4 3 3 2 3 3 2 2 3 3 3 2 3 1 3 3 2 3 3 3 2 3 3 3 3
  [85] 3 1 2 2 2 2 2 3 4 1 1 3 1 2 4 3 2 3 3 3 2 2 2 3 3 2 3 3 1 1 1 1 3 3 3 3 3 1 3 3 3 3
 [127] 1 1 1 2 2 2 4 3 3 1 3 1 1 3 1 3 3 3 3 3 3 2 3 3 3 3 1 1 3 3 3 1 1 1 2 2 1 3 3 1 2 3
 [169] 1 1 3 3 3 3 2 1 1 3 3 3 3 1 1 2 1 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 1 2 1 1 2 3 3 3 3 2
 [211] 3 3 1 1 1 1 3 3 3 3 4 3 2 2 2 2 2 2 2 2 3 1 1 3 1 3 3 2 1 2 2 3 2 1 2 2 3 2 2 2 3 3
 [253] 3 2 2 3 3 1 2 2 4 3 4 1 3 2 2 1 1 3 3 3 3 1 4 2 1 3 1 1 2 2 3 3 3 3 3 2 2 2 1 1 1 3
 [295] 3 2 2 1 3 2 2 3 3 3 1 2 2 1 1 2 1 1 2 3 3 3 2 2 2 3 1 2 2 2 3 2 2 2 2 2 2 1 3 2 2 4
 [337] 2 1 2 1 3 2 1 3 3 2 3 1 3 3 3 3 3 3 3 3 2 1 1 2 3 2 2 3 3 3 3 3 1 3 2 2 1 3 1 4 3 2
 [379] 3 2 2 3 1 3 1 3 3 2 3 3 3 2 3 3 3 3 1 3 3 2 3 3 3 3 3 2 2 1 3 3 3 3 1 2 2 3 3 2 3 3
 [421] 3 3 3 3 3 3 3 2 2 3 3 1 1 1 1 3 2 3 2 2 2 1 2 1 2 3 3 3 3 3 2 2 3 3 1 3 1 2 2 2 1 1
 [463] 1 2 2 3 3 3 3 3 1 3 3 3 1 1 3 3 3 3 1 3 2 3 3 3 4 4 3 3 1 3 3 3 3 3 3 3 1 1 1 1 3 1
 [505] 1 1 3 3 3 2 1 3 2 2 1 1 3 3 3 3 3 3 3 3 2 3 3 3 3 1 3 3 3 3 3 3 3 3 1 3 3 3 3 3 1 2
 [547] 3 3 3 1 3 3 2 3 3 1 3 3 2 4 3 3 3 3 3 3 3 3 1 3 1 1 3 3 3 2 2 1 1 2 1 3 1 2 3 3 3 3
 [589] 3 2 3 3 2 3 3 3 1 1 3 1 3 3 3 4 4 1 1 1 3 3 3 1 3 1 1 3 3 3 3 4 3 3 4 3 3 3 3 2 1 3
 [631] 3 1 3 3 1 1 3 2 3 3 3 3 3 3 3 3 3 4 1 1 2 2 4 4 3 3 3 3 3 3 3 3 2 4 2 3 1 3 3 3 3 2
 [673] 4 3 2 2 4 3 3 2 1 4 4 3 3 3 2 3 3 3 3 3 2 1 3 1 1 3 2 3 1 1 1 1 2 1 2 2 2 1 3 1 1 3
 [715] 2 2 4 3 3 3 3 4 1 1 3 2 3 2 3 2 2 4 3 2 3 1 1 3 3 1 1 2 1 4 3 1 3 1 4 2 1 4 2 2 2 4
 [757] 3 2 1 3 2 3 2 3 3 3 3 2 1 3 1 4 1 1 3 3 3 2 2 3 3 2 3 3 3 1 2 3 2 1 3 3 1 3 3 2 3 3
 [799] 3 3 3 2 2 3 2 1 3 3 3 3 4 4 2 2 1 1 3 3 3 2 3 3 1 1 2 2 3 2 3 1 1 1 1 1 1 4 3 3 3 3
 [841] 3 4 3 3 3 3 3 3 3 1 2 1 2 2 2 2 2 1 3 3 4 3 4 1 1 1 1 3 3 2 3 3 3 3 3 2 3 2 2 1 2 1
 [883] 2 3 3 2 2 2 2 3 3 3 3 1 1 3 1 3 3 1 2 3 3 2 2 2 2 1 3 1 2 3 1 2 2 3 3 1 1 2 1 1 1 3
 [925] 4 3 1 1 3 1 2 1 3 3 1 2 3 1 1 3 3 3 3 3 3 1 3 1 3 2 1 4 2 2 3 3 1 3 3 3 3 3 3 3 2 3
 [967] 3 3 3 1 1 4 3 3 3 3 3 2 3 3 3 2 3 2 3 3 3 3 1 3 3 3 2 1 3 2 3 3 3 1
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 249.0492 221.9356 347.6170 182.5463
 (between_SS / total_SS =  36.8 %)

Available components:

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

The sizes of the three clusters are 971, 859, 1282, and 799, respectively, and the result of between sum fo square/ total sum of square when k-mean is performed with k=3 is 36.8 percent, as indicated in the above output.



Confusion Matrix



```
    y
     5   6   7   8
  1 251 512 182  26
  2 235 431 162  31
  3 568 573 118  23
  4  50 349 335  65
```

K=4;

accuracy = (251 + 431 + 118 + 65)/3911

= 0.2211

= 22.11%

Winner

So, based on the aforementioned descriptions, the forecasting accuracy for two clusters is 45.89 percent, while the predicting accuracy for three clusters and four clusters is 24.92 percent and 22.11 percent, respectively.

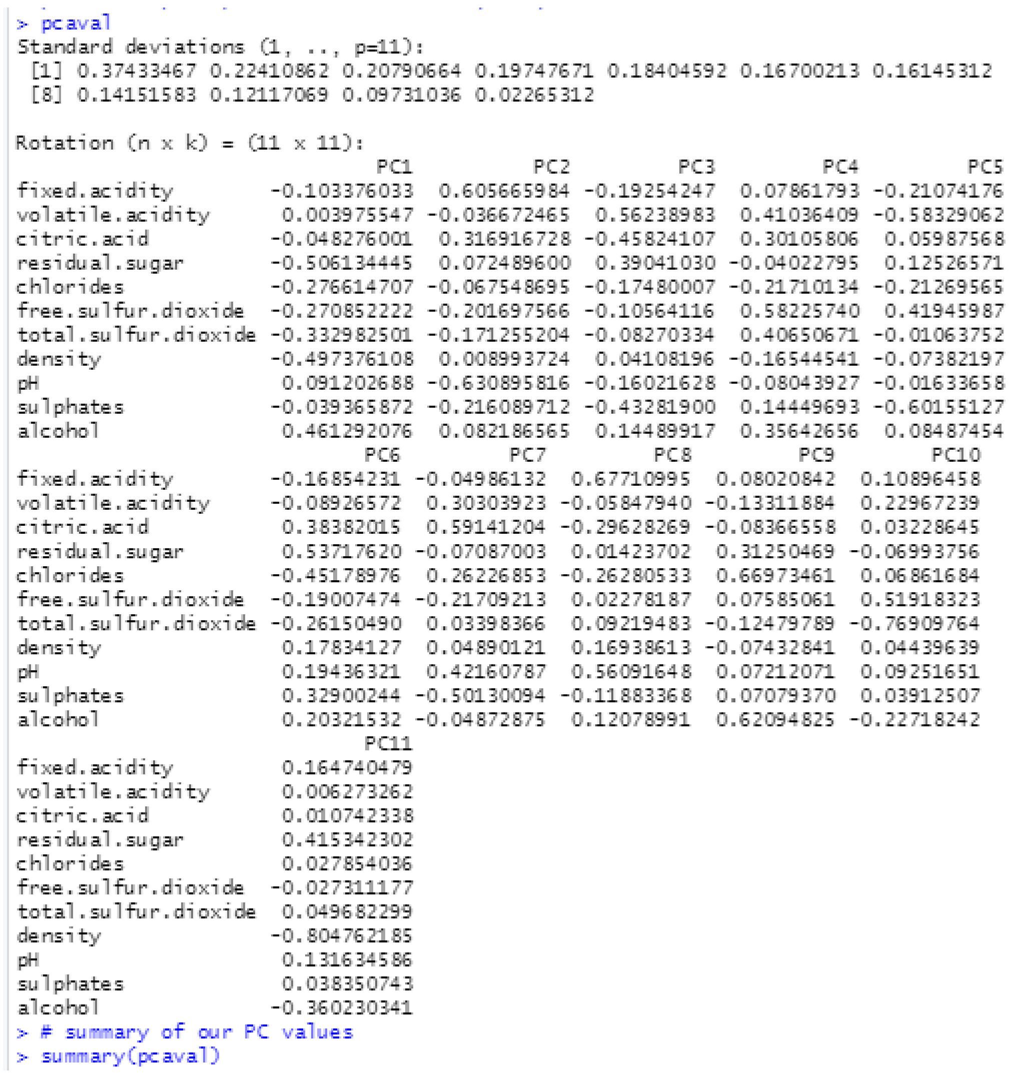
The sensitivity and specificity ratings yield good accuracy when two clusters are compared to four and three clusters. As a consequence, cluster center with count two will be the model's winning scenario.

PCA

When applying PCA to the dataset, the scale function is set to false since the technique is given a scaled input dataset.

```
#PCA
pcaval <- prcomp(WhitewineNormalize[,-12], scale = FALSE)

pcaval
```

R Output,

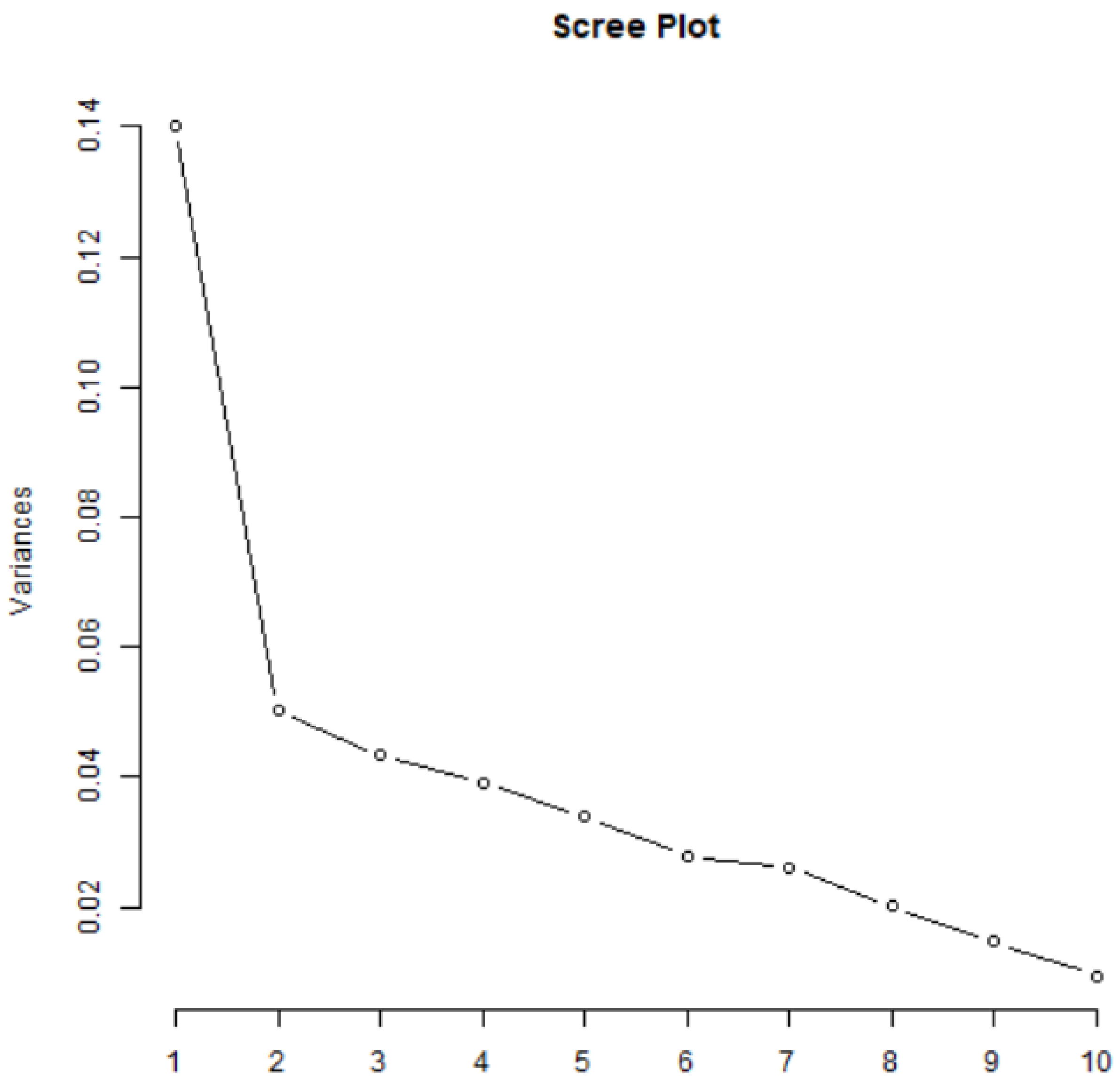


```
> pcaval
Standard deviations (1, .., p=11):
 [1] 0.37433467 0.22410862 0.20790664 0.19747671 0.18404592 0.16700213 0.16145312
 [8] 0.14151583 0.12117069 0.09731036 0.02265312

Rotation (n x k) = (11 x 11):
                              PC1          PC2         PC3         PC4         PC5
fixed.acidity        -0.103376033  0.605665984 -0.19254247  0.07861793 -0.21074176
volatile.acidity      0.003975547 -0.036672465  0.56238983  0.41036409 -0.58329062
citric.acid          -0.048276001  0.316916728 -0.45824107  0.30105806  0.05987568
residual.sugar       -0.506134445  0.072489600  0.39041030 -0.04022795  0.12526571
chlorides            -0.276614707 -0.067548695 -0.17480007 -0.21710134 -0.21269565
free.sulfur.dioxide  -0.270852222 -0.201697566 -0.10564116  0.58225740  0.41945987
total.sulfur.dioxide -0.332982501 -0.171255204 -0.08270334  0.40650671 -0.01063752
density              -0.497376108  0.008993724  0.04108196 -0.16544541 -0.07382197
pH                    0.091202688 -0.630895816 -0.16021628 -0.08043927 -0.01633658
sulphates            -0.039365872 -0.216089712 -0.43281900  0.14449693 -0.60155127
alcohol               0.461292076  0.082186565  0.14489917  0.35642656  0.08487454
                             PC6         PC7         PC8         PC9        PC10
fixed.acidity        -0.16854231 -0.04986132  0.67710995  0.08020842  0.10896458
volatile.acidity     -0.08926572  0.30303923 -0.05847940 -0.13311884  0.22967239
citric.acid           0.38382015  0.59141204 -0.29628269 -0.08366558  0.03228645
residual.sugar        0.53717620 -0.07087003  0.01423702  0.31250469 -0.06993756
chlorides            -0.45178976  0.26226853 -0.26280533  0.66973461  0.06861684
free.sulfur.dioxide  -0.19007474 -0.21709213  0.02278187  0.07585061  0.51918323
total.sulfur.dioxide -0.26150490  0.03398366  0.09219483 -0.12479789 -0.76909764
density               0.17834127  0.04890121  0.16938613 -0.07432841  0.04439639
pH                    0.19436321  0.42160787  0.56091648  0.07212071  0.09251651
sulphates             0.32900244 -0.50130094 -0.11883368  0.07079370  0.03912507
alcohol               0.20321532 -0.04872875  0.12078991  0.62094825 -0.22718242
                             PC11
fixed.acidity         0.164740479
volatile.acidity      0.006273262
citric.acid           0.010742338
residual.sugar        0.415342302
chlorides             0.027854036
free.sulfur.dioxide  -0.027311177
total.sulfur.dioxide  0.049682299
density              -0.804762185
pH                    0.131634586
sulphates             0.038350743
alcohol              -0.360230341
> # summary of our PC values
> summary(pcaval)
```

summary of our pc values,

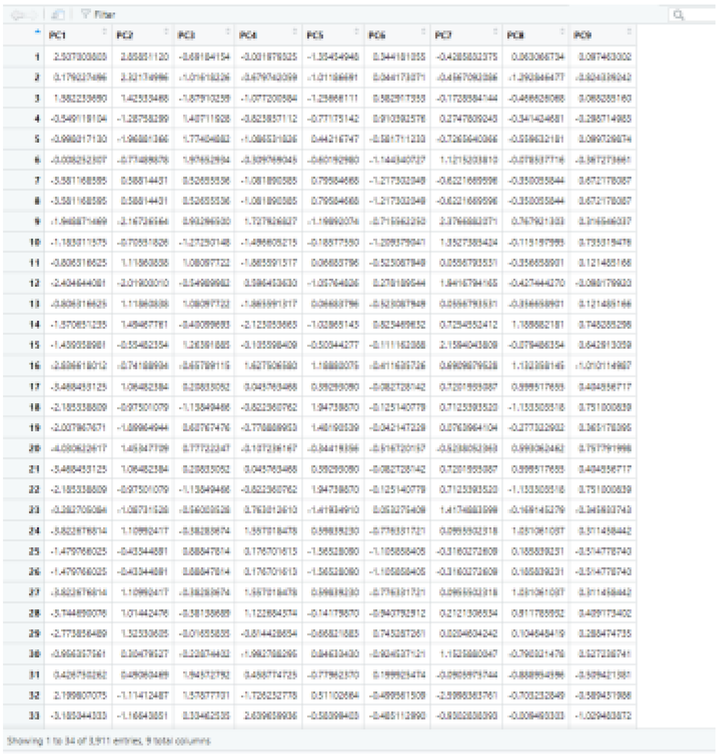
```
> summary(pcaval)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8
Standard deviation     0.3743 0.2241 0.2079 0.19748 0.18405 0.16700 0.16145 0.14152
Proportion of Variance 0.3459 0.1240 0.1067 0.09627 0.08362 0.06885 0.06435 0.04944
Cumulative Proportion  0.3459 0.4699 0.5766 0.67286 0.75648 0.82533 0.88968 0.93911
                           PC9    PC10    PC11
Standard deviation     0.12117 0.09731 0.02265
Proportion of Variance 0.03624 0.02338 0.00127
Cumulative Proportion  0.97536 0.99873 1.00000
```

The relative fraction of variation explained by PCs is shown,



We developed a new dataset using PC1 through PC9 as the properties of our new data set since the cumulative score in the PC9 is greater than 96 percent.

```r
newdataset <- myPr$x[,1:9]
```



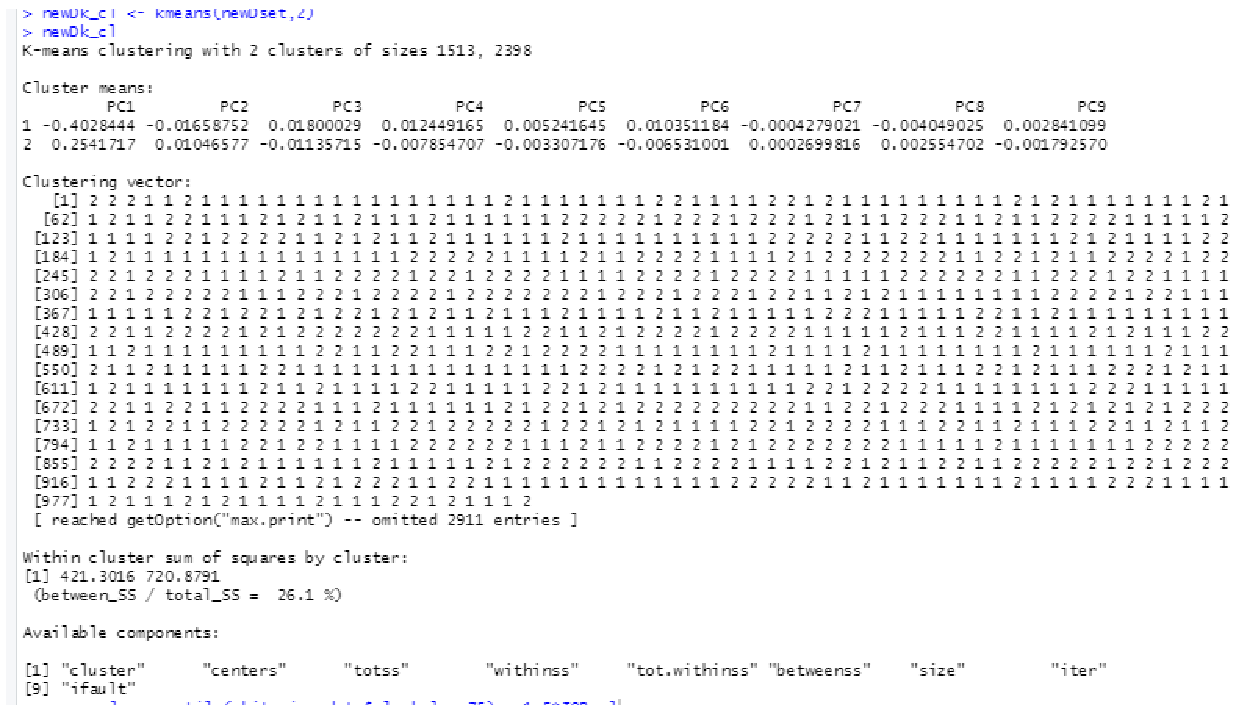
	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9
1	2.507300803	2.85851120	-0.68184154	-0.001878525	-1.35454946	0.344181055	-0.4285832375	0.063066734	0.087463002
2	0.179227496	2.32174986	-1.01618226	-0.479742039	-1.01186691	0.344173071	-0.4567092086	-1.292846477	-0.824339242
3	1.582239690	1.42533468	-1.87910239	-1.077200584	-1.25866111	0.582917353	-0.1728584144	-0.466626068	0.066285160
4	-0.549118104	-1.28758289	1.48711928	-0.823857112	-0.77175142	0.910392576	0.2747809245	-0.341424681	-0.286714985
5	-0.998017130	-1.96881366	1.77404882	-1.086531826	0.44216747	-0.581711233	-0.7265640066	-0.559632181	0.089729874
6	-0.008252307	-0.77489878	1.97652694	-0.309769045	-0.60192980	-1.144340727	1.1215233810	-0.078537716	-0.387273861
7	-3.581168595	0.58814431	0.52655536	-1.081890385	0.79584668	-1.217302348	-0.6221669596	-0.350055844	0.672178087
8	-3.581168595	0.58814431	0.52655536	-1.081890385	0.79584668	-1.217302348	-0.6221669596	-0.350055844	0.672178087
9	-1.948871469	-2.16726564	0.93296530	1.727826827	-1.19892074	-0.715562252	2.3766882071	0.767921303	0.316546037
10	-1.183311575	-0.70551828	-1.27230148	-1.486605215	-0.18577550	-1.209379041	1.3527385424	-0.115157995	0.733319476
11	-0.806318625	1.11863836	1.08097722	-1.865591317	0.06883796	-0.523087849	0.0556793531	-0.356658901	0.121485166
12	-2.404644081	-2.01900010	-0.54989982	0.986453630	-1.05764826	0.278189544	1.8416794165	-0.427444270	-0.098179920
13	-0.806318625	1.11863836	1.08097722	-1.865591317	0.06883796	-0.523087849	0.0556793531	-0.356658901	0.121485166
14	-1.570651235	1.49467781	-0.40089693	-2.123853663	-1.02865143	0.823469632	0.7254552412	1.188882181	0.745285298
15	-1.459558981	-0.55482354	1.26381885	-0.135598409	-0.50044277	-0.111162388	2.1584043808	-0.079486354	0.642813059
16	-2.836618012	-0.74188934	-0.65789115	1.627506580	1.18880075	-0.411635726	0.6909879528	1.132268145	-1.010114987
17	-3.468453125	1.06482384	0.28835052	0.045763468	0.59295080	-0.082728142	0.7201555087	0.998517655	0.484556717
18	-2.185338609	-0.97501079	-1.13849466	-0.822360762	1.94738870	-0.125140779	0.7125393520	-1.133505518	0.751000839
19	-2.007867671	-1.89964944	0.60767476	-0.778888953	1.48190539	-0.042147228	0.0763964104	-0.277322902	0.363178395
20	-4.050622617	1.45347709	0.77722247	-0.137236167	-0.84419356	-0.516720157	-0.5238052363	0.993062482	0.757791998
21	-3.468453125	1.06482384	0.28835052	0.045763468	0.59295080	-0.082728142	0.7201555087	0.998517655	0.484556717
22	-2.185338609	-0.97501079	-1.13849466	-0.822360762	1.94738870	-0.125140779	0.7125393520	-1.133505518	0.751000839
23	-0.282709084	-1.08731526	-0.56035528	0.763812610	-1.41834910	0.053275409	1.4174883099	-0.169145279	-0.345833743
24	-3.822676814	1.10992417	-0.38283674	1.557018478	0.59839230	-0.776331721	0.0955532318	1.031061087	0.311458442
25	-1.479766025	-0.43344881	0.88847814	0.176701613	-1.56528080	-1.105858405	-0.3160272809	0.185839231	-0.514778740
26	-1.479766025	-0.43344881	0.88847814	0.176701613	-1.56528080	-1.105858405	-0.3160272809	0.185839231	-0.514778740
27	-3.822676814	1.10992417	-0.38283674	1.557018478	0.59839230	-0.776331721	0.0955532318	1.031061087	0.311458442
28	-3.744890076	1.01442476	-0.38135689	1.122884374	-0.14179870	-0.940792912	0.2121306534	0.811785952	0.489173402
29	-2.773856489	1.52530605	-0.01655835	-0.814428654	-0.66821683	0.745287261	0.0204604242	0.104646419	0.288474735
30	-0.956357561	0.30478127	-0.22874432	-1.982768295	0.84633430	-0.924537121	1.1525880047	-0.790321476	0.527238741
31	0.426750262	0.49060489	1.94572782	0.458774725	-0.77962370	0.199823474	-0.0905975744	-0.888954586	-0.509421581
32	2.199807075	-1.11412487	1.57677701	-1.726252778	0.51102664	-0.499581508	-2.5998363761	-0.703232849	-0.589431988
33	-3.185344333	-1.16643851	0.33462535	2.639658936	-0.58398403	-0.485112890	-0.8502838393	-0.008469333	-1.029483872

Showing 1 to 34 of 3,911 entries, 9 total columns

The k-mean clustering approach was then applied to the new dataset with a value of 2.

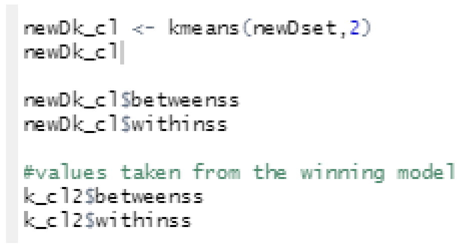
```r
# Using k mean clustering for the new dataset

newDk_cl <- kmeans(newDset,2)
newDk_cl
fviz_cluster(k_cl4, whiteNormUpdated, geom = "point")
```

R Output,



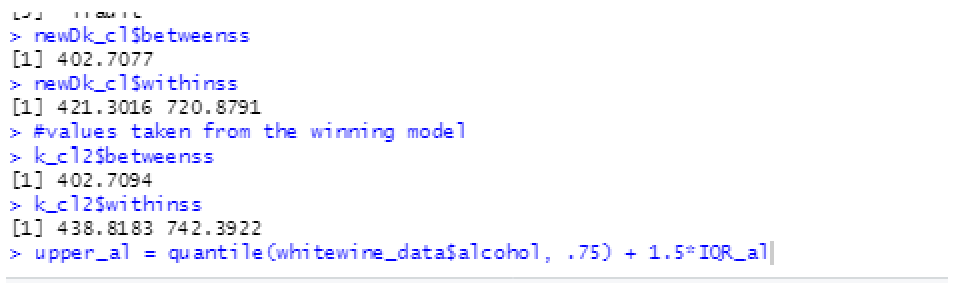
```
> newDk_cl <- kmeans(newDset,2)
> newDk_cl
K-means clustering with 2 clusters of sizes 1513, 2398

Cluster means:
         PC1         PC2         PC3          PC4          PC5          PC6           PC7          PC8          PC9
1 -0.4028444 -0.01658752  0.01800029  0.012449165  0.005241645  0.010351184 -0.0004279021 -0.004049025  0.002841099
2  0.2541717  0.01046577 -0.01135715 -0.007854707 -0.003307176 -0.006531001  0.0002699816  0.002554702 -0.001792570

Clustering vector:
   [1] 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 2 1
  [62] 1 2 1 1 2 2 1 1 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 1 1 1 2 2 2 1 1 2 1 1 2 2 2 2 1 1 1 1 1 2
 [123] 1 1 1 1 2 2 1 2 2 2 2 1 1 2 1 2 1 1 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 2 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 2
 [184] 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 1 1 2 1 1 2 2 2 2 1 1 1 1 2 1 2 2 2 2 2 2 2 1 1 2 2 1 2 1 1 2 2 2 2 1 2 2
 [245] 2 2 1 2 2 2 1 1 1 1 2 1 1 2 2 2 2 1 2 2 1 2 2 2 2 1 1 1 1 2 2 2 2 1 2 2 2 2 1 1 1 1 1 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 1 1 1
 [306] 2 2 1 2 2 2 2 2 1 1 1 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 1 1
 [367] 1 1 1 1 1 2 2 1 2 2 1 2 1 2 2 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1 2 1 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1
 [428] 2 2 1 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 1 2 1 2 2 2 2 1 2 2 2 2 1 1 1 1 1 2 1 1 1 2 2 1 1 1 1 2 1 2 1 1 1 2 2
 [489] 1 1 2 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 1 1 1 2 2 1 2 2 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1
 [550] 2 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 1 2 2 1 1 1 1 1 2 1 1 2 1 1 1 2 2 1 2 1 1 1 2 2 2 1 2 1 1
 [611] 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 2 2 2 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1
 [672] 2 2 1 1 2 2 1 1 2 2 2 2 1 1 1 2 1 1 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2 2 1 1 1 1 2 1 2 1 2 1 2 1 2 2 2
 [733] 1 2 1 2 2 1 1 2 2 2 2 2 1 2 1 1 2 2 1 2 2 2 2 2 1 2 2 1 2 1 2 1 1 1 1 2 2 1 2 2 2 2 1 1 1 2 2 1 1 2 1 1 1 2 2 1 1 2 1 1 2
 [794] 1 1 2 1 1 1 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 2 2 1 1 1 2 1 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 2 2 2 2
 [855] 2 2 2 2 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 2 2 1 1 2 2 2 2 2 1 2 2 1 2 2 2
 [916] 1 1 2 2 2 1 1 1 1 2 1 1 2 1 2 2 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 1 1
 [977] 1 2 1 1 1 2 1 2 1 1 1 1 2 1 1 1 2 2 1 2 1 1 1 2
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 421.3016 720.8791
 (between_SS / total_SS =  26.1 %)

Available components:

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

We can use the following codes to get the BSS and WSS values of our clusters, and the BSS and TSS ratio can be taken from the above figure.



```
newDk_cl <- kmeans(newDset,2)
newDk_cl

newDk_cl$betweenss
newDk_cl$withinss

#values taken from the winning model
k_cl2$betweenss
k_cl2$withinss
```

R Output,



```
> newDk_cl$betweenss
[1] 402.7077
> newDk_cl$withinss
[1] 421.3016 720.8791
> #values taken from the winning model
> k_cl2$betweenss
[1] 402.7094
> k_cl2$withinss
[1] 438.8183 742.3922
> upper_al = quantile(whitewine_data$alcohol, .75) + 1.5*IQR_al
```

Comparison Table

	k_cl2	newDk_cl
BSS	402.7094	402.7077
Ratio BSS/TSS	25.4	26.1
WSS	438.8183 742.3922	421.3016 720.8791

PCA simplifies high-dimensional data while maintaining trends and patterns. It does it by compressing the data into a smaller number of dimensions that act as feature summaries. We utilized k-mean clustering with 11 attributes in the k_cl2 and PCA to decrease dimensionality to 9 attributes in the newDk_cl, as seen in the examples above. Both strategies, however, provide essentially equal results.

Within each cluster, the sum of squares represents the variability of the observations. In general, a cluster with a small sum of squares is more compact than one with a big total of squares. When the WSS of the above two models are compared, we can observe that the data in newDk cl, which was done using PCA, has more compact data in its clusters than k cl2 since newDk cl's WSS values are smaller. As a consequence, there have been minor adjustments in the BSS/TSS ratio and BSS values favorable to the k-mean model utilizing the PC values dataset.

2nd Objective

In the electrical loading forecasting issue, there are several approaches for defining the input vectors.

Despite the fact that various forecasting methods and models have been created to calculate reliable load forecasts, it is difficult to identify an effective forecasting model for a given electrical network, and none of them can be extended to all demand patterns. Models for predicting electric load can be divided into two categories:

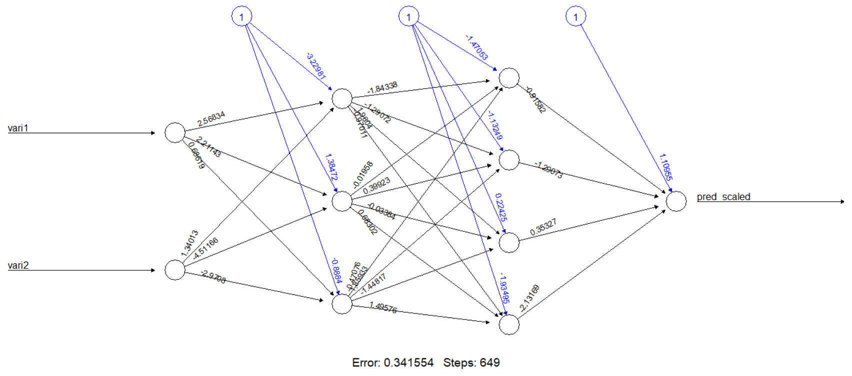
- Multi-factor forecasting methods
- Time series forecasting methods

The multi-factor/cross-sectional forecasting approach is focused on finding causal links between multiple influencing factors and forecasted values. On the other hand, the time series forecasting approach is more reliant on previous data. As a result, many academics are using time series forecasting to anticipate electric demand in order to avoid the complex and non-objective aspects that might impair the accuracy of a multi-factor forecasting model. Time series forecasting has become considerably easier and faster as a result. Statistical models, machine learning models, and hybrid models are the three most prevalent and commonly utilized time series forecasting models (Hammad et al., 2020).

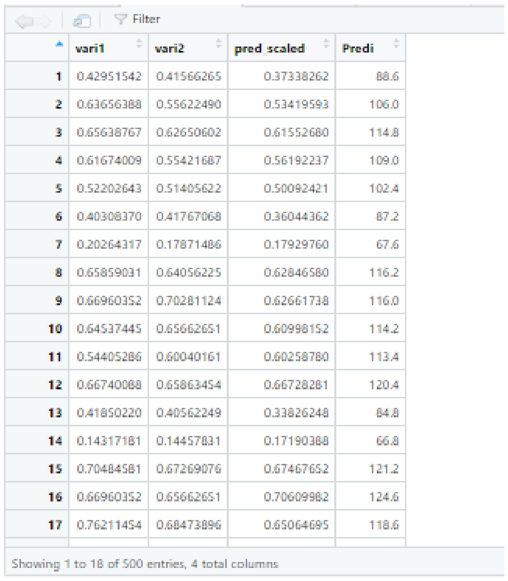
Evidence of numerous input vectors and input/output matrices that have been used

Columns vari1 and vari2 in the table below reflect the input vectors, whereas pred scaled represents the output vectors utilized in the AR technique.

The next image demonstrates how we utilize vari1 and vari2 as input vectors and pred scaled as the output prediction for model 1 that we trained using the AR technique.



The input vectors used in the NARX approach are represented in the table below by column norm-pred, which is taken from scaling.



	vari1	vari2	pred_scaled	Predi
1	0.42951542	0.41566265	0.37338262	88.6
2	0.63656388	0.55622490	0.53419593	106.0
3	0.65638767	0.62650602	0.61552680	114.8
4	0.61674009	0.55421687	0.56192237	109.0
5	0.52202643	0.51405622	0.50092421	102.4
6	0.40308370	0.41767068	0.36044362	87.2
7	0.20264317	0.17871486	0.17929760	67.6
8	0.65859031	0.64056225	0.62846580	116.2
9	0.66960352	0.70281124	0.62661738	116.0
10	0.64537445	0.65662651	0.60998152	114.2
11	0.54405286	0.60040161	0.60258780	113.4
12	0.66740088	0.65863454	0.66728281	120.4
13	0.41850220	0.40562249	0.33826248	84.8
14	0.14317181	0.14457831	0.17190388	66.8
15	0.70484581	0.67269076	0.67467652	121.2
16	0.66960352	0.65662651	0.70609982	124.6
17	0.76211454	0.68473896	0.65064695	118.6

Showing 1 to 18 of 500 entries, 4 total columns

Normalisation

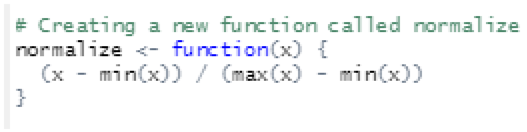
Evidence of correct normalization

The difference between any value and the minimum value is divided by the difference between the maximum and minimum values in the normalization procedure. Normalization is seen in the diagram below.

$$X_{norm} = \frac{x - min(x)}{max(x) - min(x)}$$

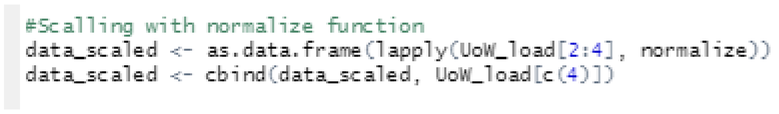
Where x is any value from the feature x, min(X) is the feature's minimum value, and max(x) is the feature's maximum value.

The graphic below depicts the normalizing function that we utilized in this project.



```r
# Creating a new function called normalize
normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}
```

The values alter and come between 1 and 0 after applying the normalizing function to chosen columns of our dataset.



```r
#Scalling with normalize function
data_scaled <- as.data.frame(lapply(UoW_load[2:4], normalize))
data_scaled <- cbind(data_scaled, UoW_load[c(4)])
```

Why is the Normalization procedure necessary ?

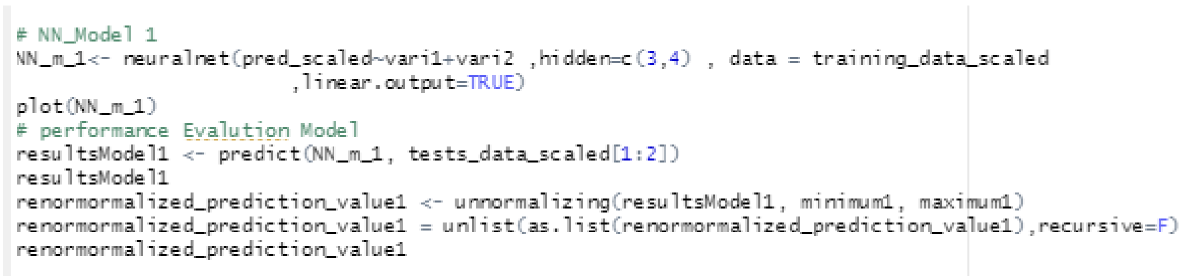
The most basic method of normalization is merging all values measured on multiple scales into a single scale. Normalization is a statistical rescaling procedure that seeks to bring all data points closer together by fitting them into a range of 0 to 1. It's a common data scaling technique. The smallest value of each feature is transformed to 0 in this technique of data scaling, while the highest value is turned to 1. To guarantee that the table only contains data that is directly connected to the primary key, that each data field has only one data element, and that duplicated (unneeded) data is deleted, normalization is necessary.One of the best strategies for training a neural network is to normalize data to a mean around zero. Data normalization, in general, speeds up learning and leads to faster convergence. The (logistic) sigmoid function is never utilized as an activation function in the buried layers of the neural network since the tanh function (among others) seems to be substantially inflated.

While it may not appear so at first glance, there are several reasons behind this. In appearance, the tan function resembles a logistic sigmoid. The primary distinction is that the tan function provides values ranging from -1 to 1, whereas the sigmoid function returns values ranging from 0 to 1, making them both positive.

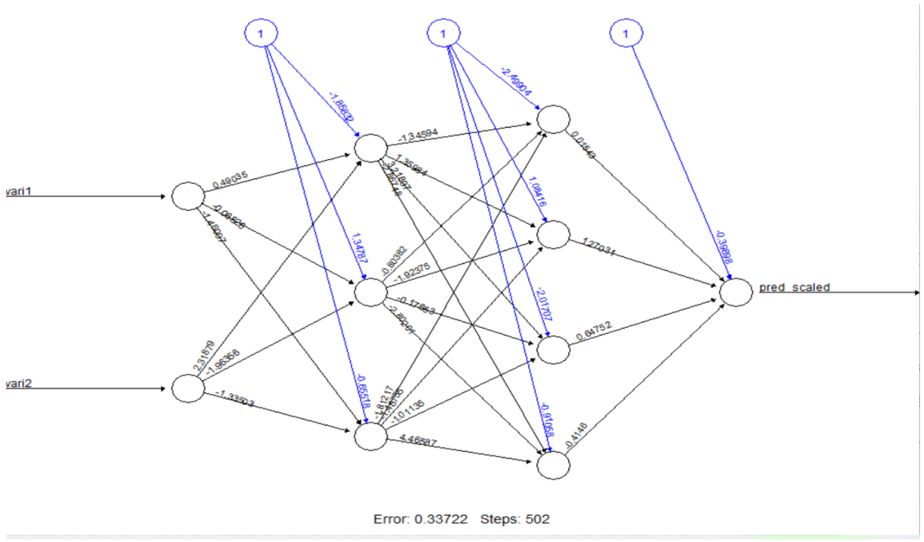
<u>Use various structures to implement numerous MLPs for the AR and NARX approaches.</u>

The input and node layers (hidden layers) were varied in the AR technique to generate three models, and the scripts and R output are listed below.

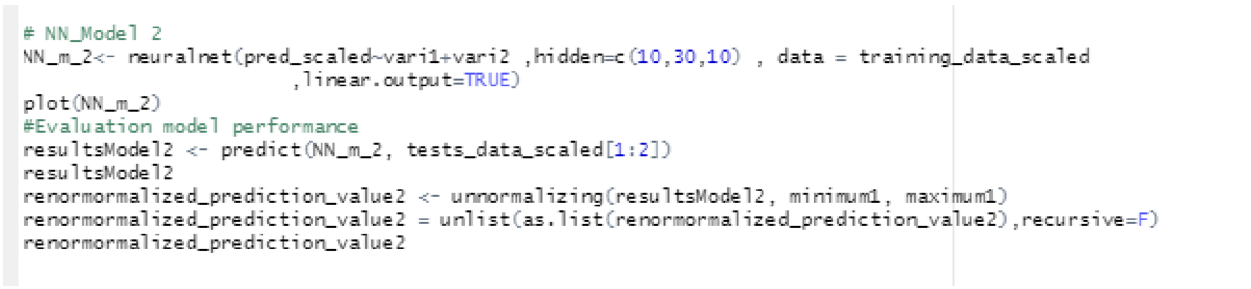
Model_1



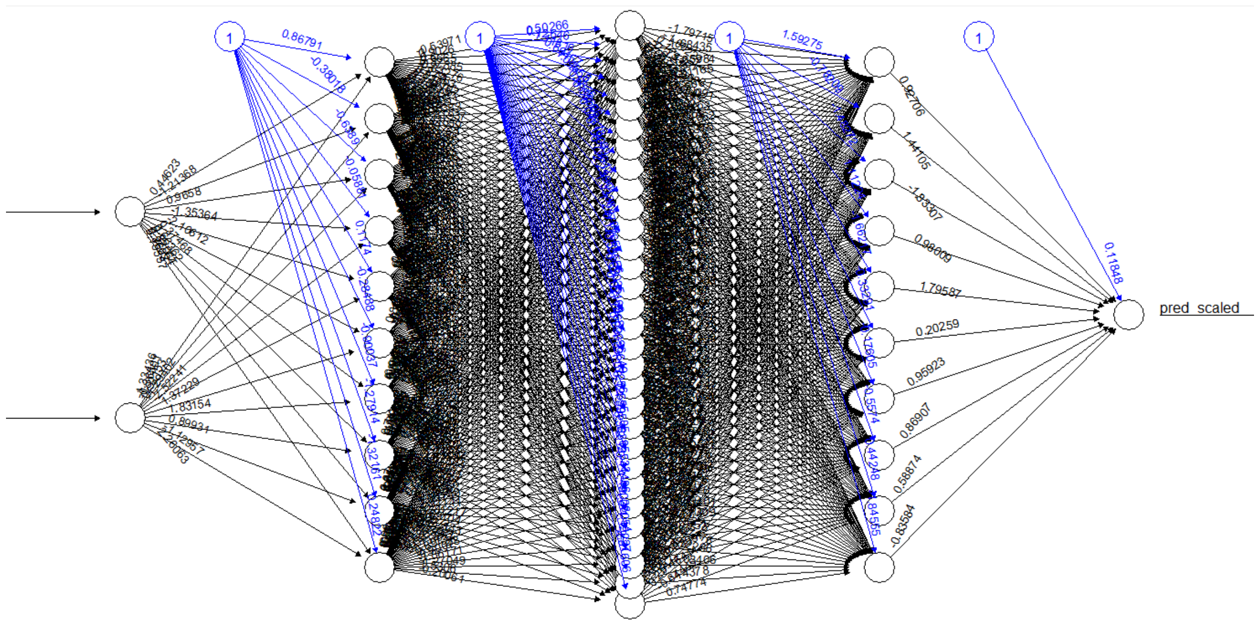
```
# NN_Model 1
NN_m_1<- neuralnet(pred_scaled~vari1+vari2 ,hidden=c(3,4) , data = training_data_scaled
                       ,linear.output=TRUE)
plot(NN_m_1)
# performance Evalution Model
resultsModel1 <- predict(NN_m_1, tests_data_scaled[1:2])
resultsModel1
renormormalized_prediction_value1 <- unnormalizing(resultsModel1, minimum1, maximum1)
renormormalized_prediction_value1 = unlist(as.list(renormormalized_prediction_value1),recursive=F)
renormormalized_prediction_value1
```



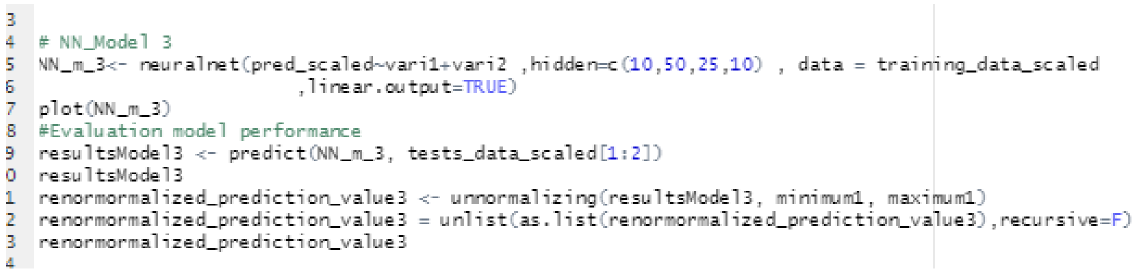
Mode_2



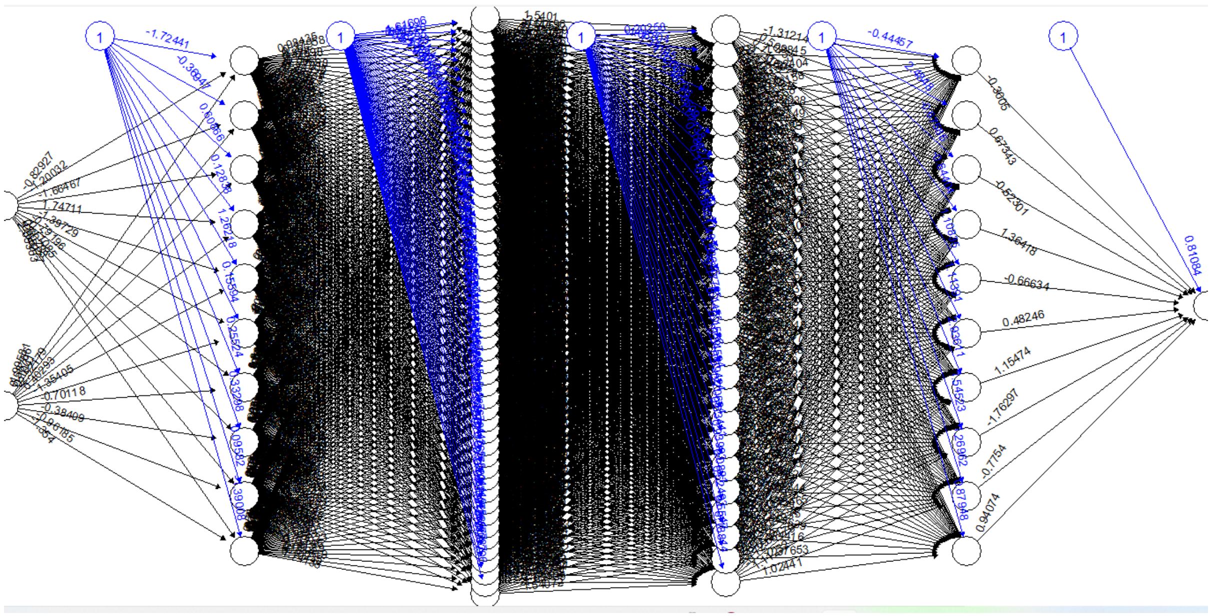
```
# NN_Model 2
NN_m_2<- neuralnet(pred_scaled~vari1+vari2 ,hidden=c(10,30,10) , data = training_data_scaled
                       ,linear.output=TRUE)
plot(NN_m_2)
#Evaluation model performance
resultsModel2 <- predict(NN_m_2, tests_data_scaled[1:2])
resultsModel2
renormormalized_prediction_value2 <- unnormalizing(resultsModel2, minimum1, maximum1)
renormormalized_prediction_value2 = unlist(as.list(renormormalized_prediction_value2),recursive=F)
renormormalized_prediction_value2
```



Model_3

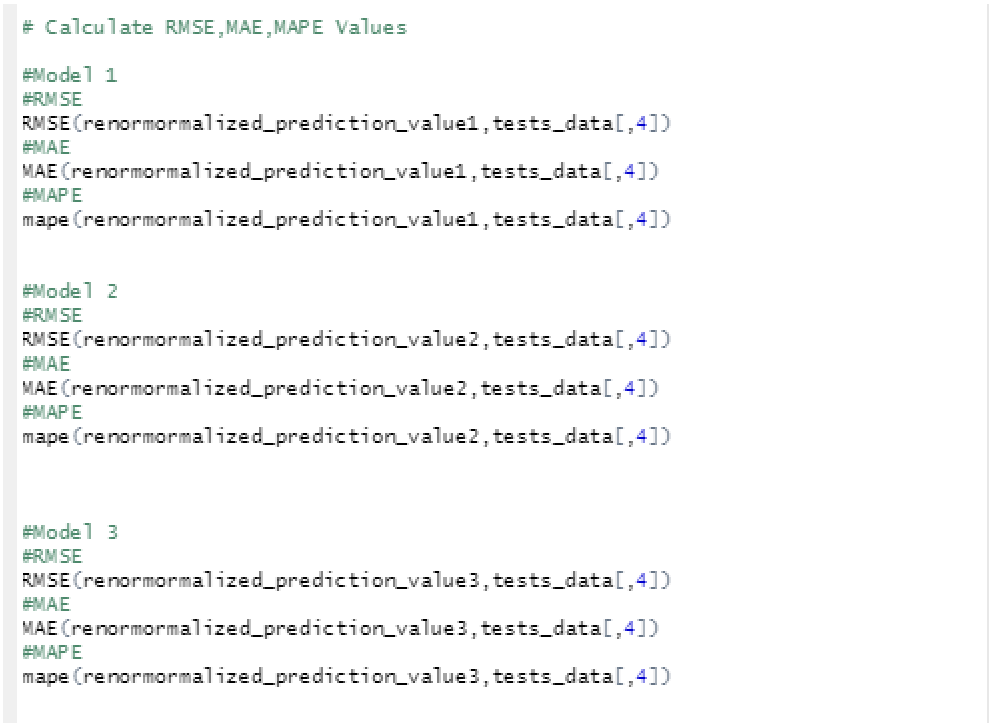


```
# NN_Model 3
NN_m_3<- neuralnet(pred_scaled~vari1+vari2 ,hidden=c(10,50,25,10) , data = training_data_scaled
                        ,linear.output=TRUE)
plot(NN_m_3)
#Evaluation model performance
resultsModel3 <- predict(NN_m_3, tests_data_scaled[1:2])
resultsModel3
renormormalized_prediction_value3 <- unnormalizing(resultsModel3, minimum1, maximum1)
renormormalized_prediction_value3 = unlist(as.list(renormormalized_prediction_value3),recursive=F)
renormormalized_prediction_value3
```



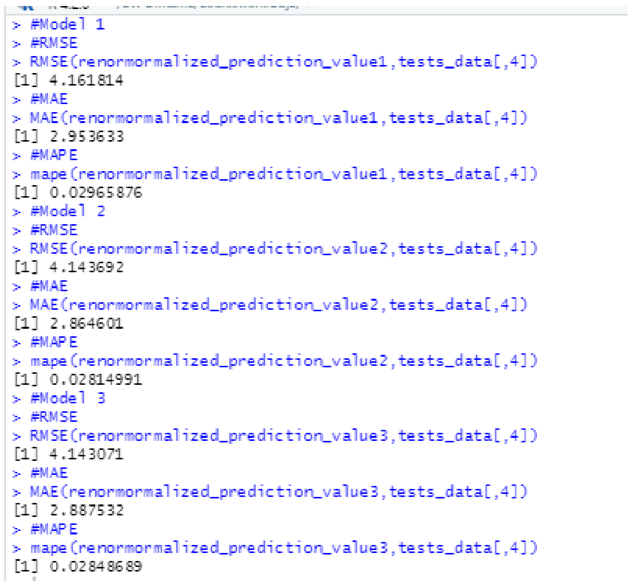
Comparison Table

Below is the evidence for how we arrived at the RMSE, MAE, and MAPE values for the above models, as well as a comparison table of those values.



```
# Calculate RMSE,MAE,MAPE Values

#Model 1
#RMSE
RMSE(renormormalized_prediction_value1,tests_data[,4])
#MAE
MAE(renormormalized_prediction_value1,tests_data[,4])
#MAPE
mape(renormormalized_prediction_value1,tests_data[,4])


#Model 2
#RMSE
RMSE(renormormalized_prediction_value2,tests_data[,4])
#MAE
MAE(renormormalized_prediction_value2,tests_data[,4])
#MAPE
mape(renormormalized_prediction_value2,tests_data[,4])



#Model 3
#RMSE
RMSE(renormormalized_prediction_value3,tests_data[,4])
#MAE
MAE(renormormalized_prediction_value3,tests_data[,4])
#MAPE
mape(renormormalized_prediction_value3,tests_data[,4])
```

R Output,

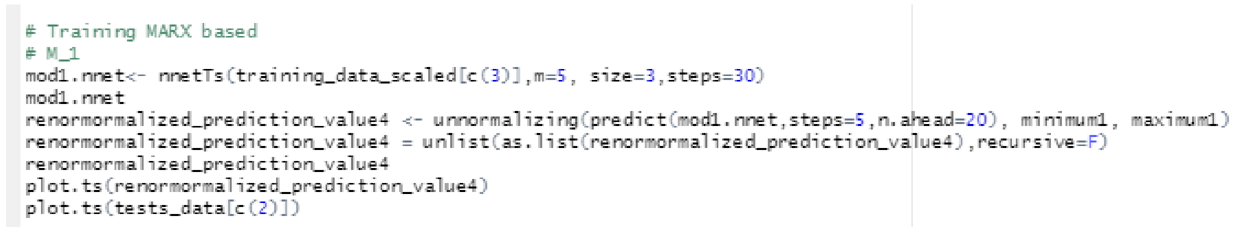


```
> #Model 1
> #RMSE
> RMSE(renormormalized_prediction_value1,tests_data[,4])
[1] 4.161814
> #MAE
> MAE(renormormalized_prediction_value1,tests_data[,4])
[1] 2.953633
> #MAPE
> mape(renormormalized_prediction_value1,tests_data[,4])
[1] 0.02965876
> #Model 2
> #RMSE
> RMSE(renormormalized_prediction_value2,tests_data[,4])
[1] 4.143692
> #MAE
> MAE(renormormalized_prediction_value2,tests_data[,4])
[1] 2.864601
> #MAPE
> mape(renormormalized_prediction_value2,tests_data[,4])
[1] 0.02814991
> #Model 3
> #RMSE
> RMSE(renormormalized_prediction_value3,tests_data[,4])
[1] 4.143071
> #MAE
> MAE(renormormalized_prediction_value3,tests_data[,4])
[1] 2.887532
> #MAPE
> mape(renormormalized_prediction_value3,tests_data[,4])
[1] 0.02848689
```

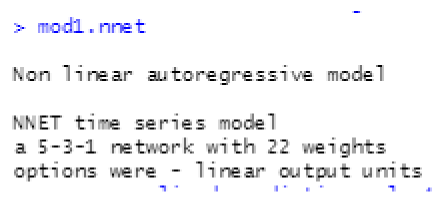
	RMSE	MAE	MAPE
Model_ 1	4.161814	2.953633	0.02965876
Model_ 2	4.143692	2.864601	0.02814991
Model _3	4.143071	2.887532	0.02848689

We utilized the NAXR technique to generate three models by altering the input R, and the codes and R output are presented below.

Model_1

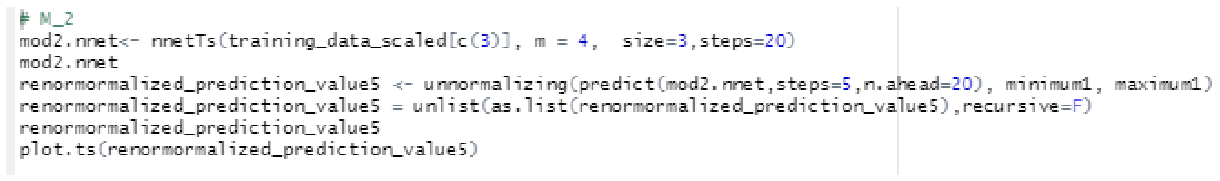


```
# Training MARX based
# M_1
mod1.nnet<- nnetTs(training_data_scaled[c(3)],m=5, size=3,steps=30)
mod1.nnet
renormormalized_prediction_value4 <- unnormalizing(predict(mod1.nnet,steps=5,n.ahead=20), minimum1, maximum1)
renormormalized_prediction_value4 = unlist(as.list(renormormalized_prediction_value4),recursive=F)
renormormalized_prediction_value4
plot.ts(renormormalized_prediction_value4)
plot.ts(tests_data[c(2)])
```

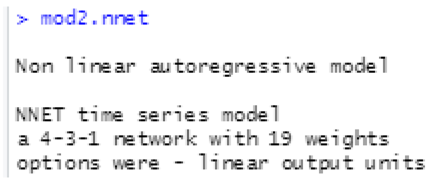


```
> mod1.nnet

Non linear autoregressive model

NNET time series model
a 5-3-1 network with 22 weights
options were - linear output units
```

Model_2

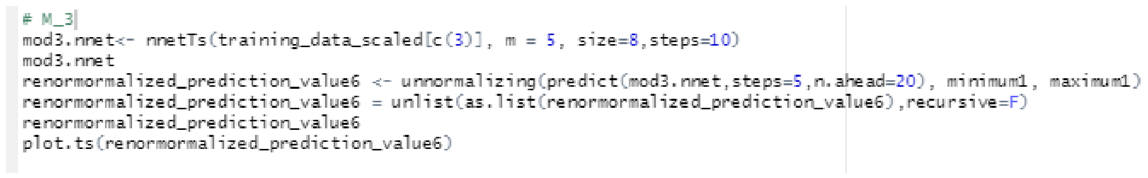


```
# M_2
mod2.nnet<- nnetTs(training_data_scaled[c(3)], m = 4,  size=3,steps=20)
mod2.nnet
renormormalized_prediction_value5 <- unnormalizing(predict(mod2.nnet,steps=5,n.ahead=20), minimum1, maximum1)
renormormalized_prediction_value5 = unlist(as.list(renormormalized_prediction_value5),recursive=F)
renormormalized_prediction_value5
plot.ts(renormormalized_prediction_value5)
```

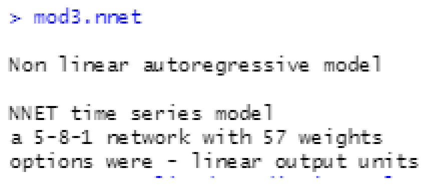


```
> mod2.nnet

Non linear autoregressive model

NNET time series model
a 4-3-1 network with 19 weights
options were - linear output units
```

Model_3



```
# M_3
mod3.nnet<- nnetTs(training_data_scaled[c(3)], m = 5, size=8,steps=10)
mod3.nnet
renormormalized_prediction_value6 <- unnormalizing(predict(mod3.nnet,steps=5,n.ahead=20), minimum1, maximum1)
renormormalized_prediction_value6 = unlist(as.list(renormormalized_prediction_value6),recursive=F)
renormormalized_prediction_value6
plot.ts(renormormalized_prediction_value6)
```



```
> mod3.nnet

Non linear autoregressive model

NNET time series model
 a 5-8-1 network with 57 weights
options were - linear output units
```

Comparison Table

	Output with network and weights
Model_1	5-3-1 network with 22 weights
Model_2	4-3-1 network with 19 weights
Model_3	5-8-1 network with 57 weights

<u>The significance of these statistics indices</u>

Statistical indices

These values describe the distribution of items in a set of elements. These indices may be used to look into the relationships between subsets of components, as well as to validate or disprove some of the laws and correlations that control their behavior.

- **RMSE (Root Mean Square Error)**
  - The Root Mean Square Error is the standard deviation of the residuals (that is, the differences between model predictions and true values (training data)) (RMSE).The RMSE provides an approximate assessment of the residuals' dispersion.
- **MAE (Mean Absolute Error)**
  - The MAE represents the average magnitude of error in the regression model. If MAE is equal to zero, the model's predictions are accurate.
  - The Mean Absolute Error is calculated using the absolute difference between the model predictions and the true (actual) data (MAE).

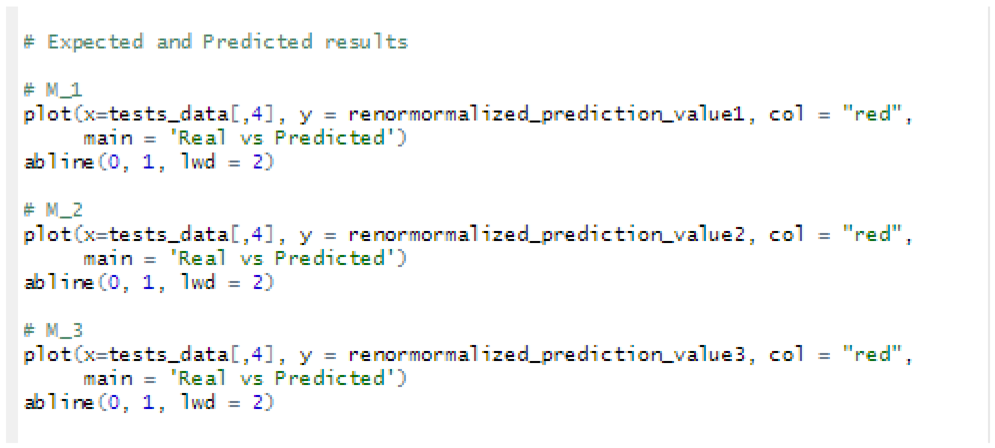
- **MAPE (Mean Absolute Percentage Error)**
  - The Mean Absolute % Mistake (MAPE), which delivers the error in percentage form and so overcomes MAE's constraints, is the equivalent of MAE. MAPE may have certain restrictions if the data point value is zero (due to the division operation).

<u>With your two finest NN architectures, discuss the subject of "efficiency."</u>

The multi-factor/cross-sectional forecasting approach is focused on finding causal links between multiple influencing factors and forecasted values. On the other hand, the time series forecasting approach is more reliant on previous data. As a result, predicting time series is more easier and faster.

<u>Best results (prediction output vs. desired output)</u>

To draw the prediction vs. desired output for the AR models, use the following code.



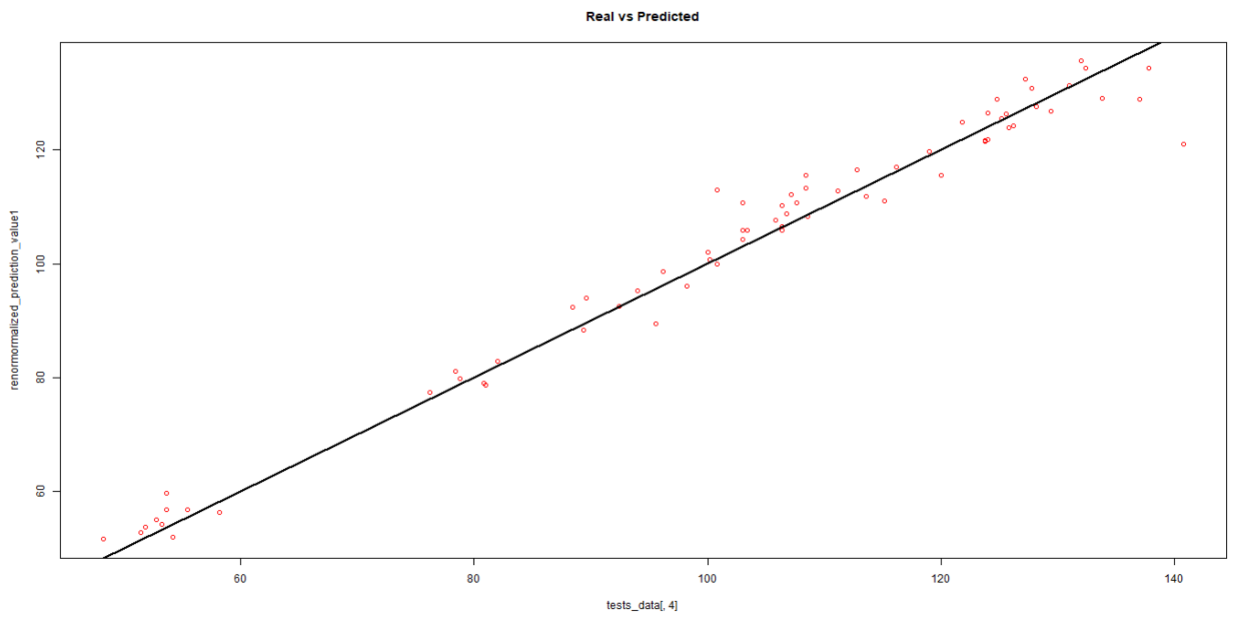
```r
# Expected and Predicted results

# M_1
plot(x=tests_data[,4], y = renormormalized_prediction_value1, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)

# M_2
plot(x=tests_data[,4], y = renormormalized_prediction_value2, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)

# M_3
plot(x=tests_data[,4], y = renormormalized_prediction_value3, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)
```

R Outputs,

Model_1

Black Line“ ___ ” shows Predicted data

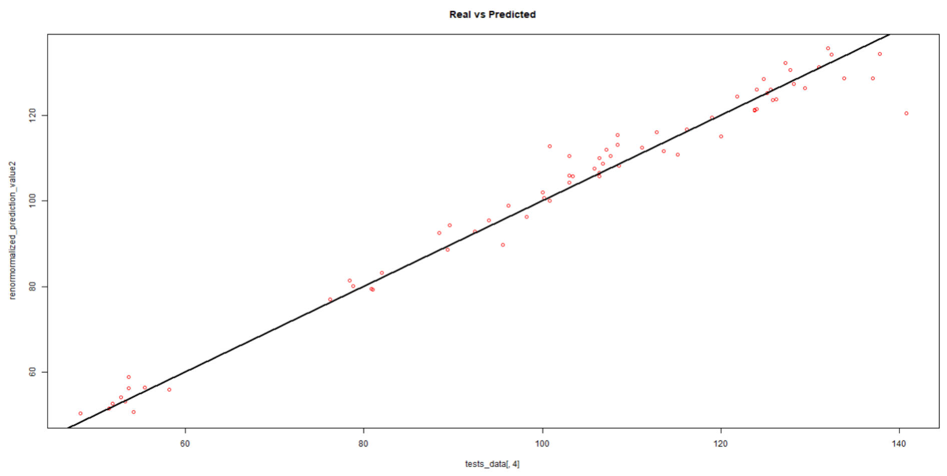
Red Circles “○” shows Real data



Model_2

Black Line“ ___ ” shows Predicted data

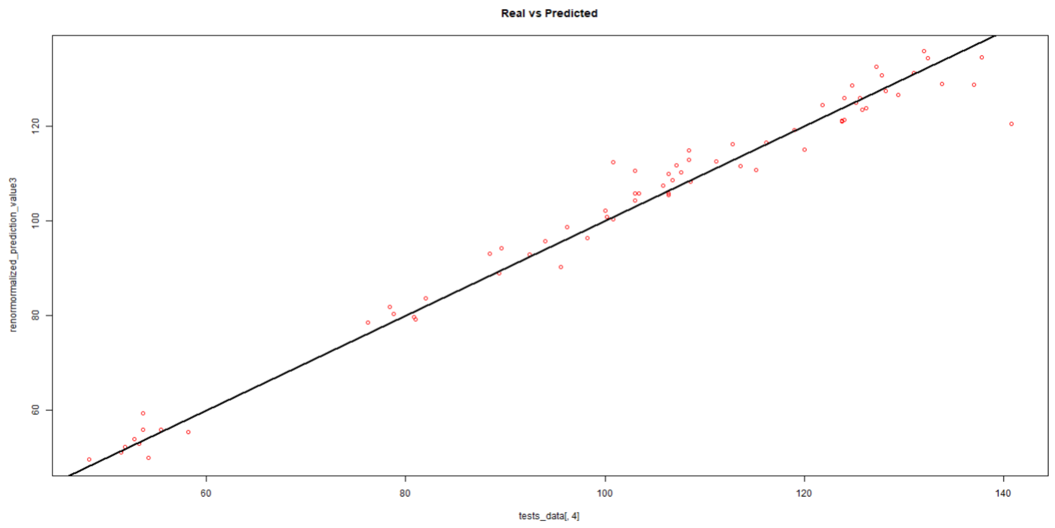
Red Circles “○” shows Real data



Model_3

Black Line" ___ " shows Predicted data

Red Circles "○" shows Real data



Appendix

1st Objective

```
library(dplyr)
library(readxl)
library(factoextra)
library(NbClust)
library(caret)
library(e1071)

# Importing the Data set
library(readxl)
whitewine_data <- read_excel("C:/Users/Oji/Documents/CW
DM&ML/Coursework.Obj1/Whitewine_v2 (2).xlsx")

# number of data check
dim(whitewine_data)

# Checking for is there are any na values
sum(is.na(whitewine_data))

# using the box plot method to check outliers
boxplot(whitewine_data[,-12])
```

```
# first column outliers removal
IQR_fa <- IQR(whitewine_data$`fixed acidity`)
lower_fa = quantile(whitewine_data$`fixed acidity`, .25) - 1.5*IQR_fa
upper_fa = quantile(whitewine_data$`fixed acidity`, .75) + 1.5*IQR_fa

# second column outliers removal
IQR_va <- IQR(whitewine_data$`volatile acidity`)
lower_va = quantile(whitewine_data$`volatile acidity`, .25) - 1.5*IQR_va
upper_va = quantile(whitewine_data$`volatile acidity`, .75) + 1.5*IQR_va

# 3 column outliers removal
IQR_ca <- IQR(whitewine_data$`citric acid`)
lower_ca = quantile(whitewine_data$`citric acid`, .25) - 1.5*IQR_ca
upper_ca = quantile(whitewine_data$`citric acid`, .75) + 1.5*IQR_ca

# 4 column outliers removal
IQR_rsu <- IQR(whitewine_data$`residual sugar`)
lower_rsu = quantile(whitewine_data$`residual sugar`, .25) - 1.5*IQR_rsu
upper_rsu = quantile(whitewine_data$`residual sugar`, .75) + 1.5*IQR_rsu

# 5 column outliers removal
IQR_cl <- IQR(whitewine_data$chlorides)
lower_cl = quantile(whitewine_data$chlorides, .25) - 1.5*IQR_cl
upper_cl = quantile(whitewine_data$chlorides, .75) + 1.5*IQR_cl
```

```
# 6 column outliers removal
IQR_fsd <- IQR(whitewine_data$`free sulfur dioxide`)
lower_fsd = quantile(whitewine_data$`free sulfur dioxide`, .25) - 1.5*IQR_fsd
upper_fsd = quantile(whitewine_data$`free sulfur dioxide`, .75) + 1.5*IQR_fsd

# 7 column outliers removal
IQR_tsd <- IQR(whitewine_data$`total sulfur dioxide`)
lower_tsd = quantile(whitewine_data$`total sulfur dioxide`, .25) - 1.5*IQR_tsd
upper_tsd = quantile(whitewine_data$`total sulfur dioxide`, .75) + 1.5*IQR_tsd

# 8 column outliers removal
IQR_de <- IQR(whitewine_data$density)
lower_de = quantile(whitewine_data$density, .25) - 1.5*IQR_de
upper_de = quantile(whitewine_data$density, .75) + 1.5*IQR_de

# 9 column outliers removal
IQR_ph <- IQR(whitewine_data$pH)
lower_ph = quantile(whitewine_data$pH, .25) - 1.5*IQR_ph
upper_ph = quantile(whitewine_data$pH, .75) + 1.5*IQR_ph

# 10 column outliers removal
IQR_sul <- IQR(whitewine_data$sulphates)
lower_sul = quantile(whitewine_data$sulphates, .25) - 1.5*IQR_sul
upper_sul = quantile(whitewine_data$sulphates, .75) + 1.5*IQR_sul
```

```
# 11 column outliers removal

IQR_al <- IQR(whitewine_data$alcohol)

lower_al = quantile(whitewine_data$alcohol, .25) - 1.5*IQR_al

upper_al = quantile(whitewine_data$alcohol, .75) + 1.5*IQR_al


updated_whitewine <- subset(whitewine_data, whitewine_data$`fixed acidity` >= lower_fa &
whitewine_data$`fixed acidity` <= upper_fa

                        & whitewine_data$`volatile acidity` >= lower_va & whitewine_data$`volatile
acidity` <= upper_va

                        & whitewine_data$`citric acid` >= lower_ca & whitewine_data$`citric acid` <=
upper_ca

                        & whitewine_data$`residual sugar` >= lower_rsu & whitewine_data$`residual
sugar` <= upper_rsu

                        & whitewine_data$chlorides >= lower_cl & whitewine_data$chlorides <=
upper_cl

                        & whitewine_data$`free sulfur dioxide` >= lower_fsd & whitewine_data$`free
sulfur dioxide` <= upper_fsd

                        & whitewine_data$`total sulfur dioxide` >= lower_tsd & whitewine_data$`total
sulfur dioxide` <= upper_tsd

                        & whitewine_data$density >= lower_de & whitewine_data$density <=
upper_de

                        & whitewine_data$pH >= lower_ph & whitewine_data$pH <= upper_ph

                        & whitewine_data$sulphates >= lower_sul & whitewine_data$sulphates <=
upper_sul

                        & whitewine_data$alcohol >= lower_al & whitewine_data$alcohol <=
upper_al)
```

```
# Boxplot
boxplot(updated_whitewine)



# Plotting the box plot for before and after removing outliers
boxplot(whitewine_data[,1])
boxplot(updated_whitewine[,1])

boxplot(whitewine_data[,2])
boxplot(updated_whitewine[,2])

boxplot(whitewine_data[,3])
boxplot(updated_whitewine[,3])

boxplot(whitewine_data[,4])
boxplot(updated_whitewine[,4])

boxplot(whitewine_data[,5])
boxplot(updated_whitewine[,5])

boxplot(whitewine_data[,6])
boxplot(updated_whitewine[,6])

boxplot(whitewine_data[,7])
boxplot(updated_whitewine[,7])
boxplot(whitewine_data[,8])
boxplot(updated_whitewine[,8])

boxplot(whitewine_data[,9])
boxplot(updated_whitewine[,9])

boxplot(whitewine_data[,10])
boxplot(updated_whitewine[,10])

boxplot(whitewine_data[,11])
boxplot(updated_whitewine[,11])


# Normalizing the data ( Scaling )

# Creating a new function called normalize
normalize  <- function(x){
  return ((x - min(x))/(max(x) - min(x)))
}

WhitewineNormalize <- as.data.frame(lapply(updated_whitewine, normalize))
whiteNormUpdated <- WhitewineNormalize[,-12]
boxplot(whiteNormUpdated)
```


```
#  Find number of clusters


# Manual method
# Creating an empty list
kval <- list()


for (i in 1 : 10){
  kval[[i]] <- kmeans(whiteNormUpdated, i)
}


kval


#  bss/tss Calculation
# list to store the val
betweenSS_totss <- list()
for (i in 1:10) {
  betweenSS_totss[[i]] <- kval[[i]]$betweenss/kval[[i]]$totss
}


# Plotting the sum of squares ratio vs the number of clusters
plot(1:10, betweenSS_totss, type = "b",
     ylab = "Between SS/Total SS", xlab = "Clusters (k)")


# Using NbClust method, determining the number of clusters ( Automated )
library(NbClust)
set.seed(26)
clusterNo = NbClust(whiteNormUpdated, distance = "euclidean", min.nc = 2, max.nc = 10,
method = "kmeans" , index = "all")
```

```
library(factoextra)

# Elbow methood is used to determine the number of clusters ( Automated )

fviz_nbclust(whiteNormUpdated, kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2)+
  labs(subtitle = "Elbow method")

# Using the silhouette approach to determine the number of clusters ( Automated )

fviz_nbclust(whiteNormUpdated, kmeans, method = "silhouette")+
  labs(subtitle = "Silhouette method")

# Adding the values of the classes to a variable
y <- updated_whitewine$quality

# Performing k-means with K == 2
k_cl2 <- kmeans(whiteNormUpdated,2)
k_cl2
fviz_cluster(k_cl2, whiteNormUpdated, geom = "point")

# confusion matrix
table(k_cl2$cluster,y)
```

```
# Performing k-means with K == 3
k_cl3 <- kmeans(whiteNormUpdated,3)
k_cl3
fviz_cluster(k_cl3, whiteNormUpdated, geom = "point")

# confusion matrix
table(k_cl3$cluster,y)


# Performing k-means with K == 4
k_cl4 <- kmeans(whiteNormUpdated,4)
k_cl4
fviz_cluster(k_cl4, whiteNormUpdated, geom = "point")


# confusion matrix
table(k_cl4$cluster,y)
```

```
#PCA
pcaval <- prcomp(WhitewineNormalize[,-12], scale = FALSE)

pcaval

# summary of our PC values
summary(pcaval)

# The fraction of variation explained by PCs is shown.
screeplot(x = pcaval, type = "line", main = "Scree Plot")
boxplot(pcaval)

newDset <- pcaval$x[,1:9]

# Using k mean clustering for the new dataset

newDk_cl <- kmeans(newDset,2)
newDk_cl

newDk_cl$betweenss
newDk_cl$withinss

#values taken from the winning model
k_cl2$betweenss
k_cl2$withinss
```

2nd Objective

```
library("neuralnet")
library("Metrics")
library("MLmetrics")
library("tsDyn")

UoW_load <- read.csv("UoW_load.csv", header = TRUE)

tests_data = tail(UoW_load, n =70)
training_data = head(UoW_load, n =430)

View(UoW_load)

View(tests_data)

# Creating a new function called normalize
normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

#Scalling with normalize function
data_scaled <- as.data.frame(lapply(UoW_load[2:4], normalize))
data_scaled <- cbind(data_scaled, UoW_load[c(4)])
```

```
# Changing column names
names(data_scaled)[1] <- "vari1"
names(data_scaled)[2] <- "vari2"
names(data_scaled)[3] <- "pred_scaled"
names(data_scaled)[4] <- "Predi"

View(data_scaled)

# Extracting data for testing and training
tests_data_scaled = tail(data_scaled, n =70)
training_data_scaled = head(data_scaled, n =430)

# un normalize function
unnormalizing <- function(x, min, max) {
  return( (max - min)*x + min )
}

minimum1 <- min(data_scaled[4])
maximum1 <- max(data_scaled[4])
```

```
# NN_Model 1
NN_m_1<- neuralnet(pred_scaled~vari1+vari2 ,hidden=c(3,4) , data = training_data_scaled
                  ,linear.output=TRUE)
plot(NN_m_1)
# performance Evalution Model
resultsModel1 <- predict(NN_m_1, tests_data_scaled[1:2])
resultsModel1
renormormalized_prediction_value1 <- unnormalizing(resultsModel1, minimum1, maximum1)
renormormalized_prediction_value1 =
unlist(as.list(renormormalized_prediction_value1),recursive=F)
renormormalized_prediction_value1


# NN_Model 2
NN_m_2<- neuralnet(pred_scaled~vari1+vari2 ,hidden=c(10,30,10) , data =
training_data_scaled
                  ,linear.output=TRUE)
plot(NN_m_2)
#Evaluation model performance
resultsModel2 <- predict(NN_m_2, tests_data_scaled[1:2])
resultsModel2
renormormalized_prediction_value2 <- unnormalizing(resultsModel2, minimum1, maximum1)
renormormalized_prediction_value2 =
unlist(as.list(renormormalized_prediction_value2),recursive=F)
renormormalized_prediction_value2
```

```
# NN_Model 3

NN_m_3<- neuralnet(pred_scaled~vari1+vari2 ,hidden=c(10,50,25,10) , data =
training_data_scaled

                    ,linear.output=TRUE)

plot(NN_m_3)

#Evaluation model performance

resultsModel3 <- predict(NN_m_3, tests_data_scaled[1:2])

resultsModel3

renormormalized_prediction_value3 <- unnormalizing(resultsModel3, minimum1, maximum1)

renormormalized_prediction_value3 =
unlist(as.list(renormormalized_prediction_value3),recursive=F)

renormormalized_prediction_value3


# Training MARX based

# M_1

mod1.nnet<- nnetTs(training_data_scaled[c(3)],m=5, size=3,steps=30)

mod1.nnet

renormormalized_prediction_value4 <- unnormalizing(predict(mod1.nnet,steps=5,n.ahead=20),
minimum1, maximum1)

renormormalized_prediction_value4 =
unlist(as.list(renormormalized_prediction_value4),recursive=F)

renormormalized_prediction_value4

plot.ts(renormormalized_prediction_value4)

plot.ts(tests_data[c(2)])
```

```
# M_2

mod2.nnet<- nnetTs(training_data_scaled[c(3)], m = 4,  size=3,steps=20)

mod2.nnet

renormormalized_prediction_value5 <- unnormalizing(predict(mod2.nnet,steps=5,n.ahead=20),
minimum1, maximum1)

renormormalized_prediction_value5 =
unlist(as.list(renormormalized_prediction_value5),recursive=F)

renormormalized_prediction_value5

plot.ts(renormormalized_prediction_value5)


# M_3

mod3.nnet<- nnetTs(training_data_scaled[c(3)], m = 5, size=8,steps=10)

mod3.nnet

renormormalized_prediction_value6 <- unnormalizing(predict(mod3.nnet,steps=5,n.ahead=20),
minimum1, maximum1)

renormormalized_prediction_value6 =
unlist(as.list(renormormalized_prediction_value6),recursive=F)

renormormalized_prediction_value6

plot.ts(renormormalized_prediction_value6)


y_tests = as.list(tests_data[4])


tests_data$X11.00

tests_data[4]
```

```
# Calculate RMSE,MAE,MAPE Values
#Model 1
#RMSE
RMSE(renormormalized_prediction_value1,tests_data[,4])
#MAE
MAE(renormormalized_prediction_value1,tests_data[,4])
#MAPE
mape(renormormalized_prediction_value1,tests_data[,4])


#Model 2
#RMSE
RMSE(renormormalized_prediction_value2,tests_data[,4])
#MAE
MAE(renormormalized_prediction_value2,tests_data[,4])
#MAPE
mape(renormormalized_prediction_value2,tests_data[,4])


#Model 3
#RMSE
RMSE(renormormalized_prediction_value3,tests_data[,4])
#MAE
MAE(renormormalized_prediction_value3,tests_data[,4])
#MAPE
mape(renormormalized_prediction_value3,tests_data[,4])
```

```
# Expected and Predicted results


# M_1
plot(x=tests_data[,4], y = renormormalized_prediction_value1, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)


# M_2
plot(x=tests_data[,4], y = renormormalized_prediction_value2, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)


# M_3
plot(x=tests_data[,4], y = renormormalized_prediction_value3, col = "red",
     main = 'Real vs Predicted')
abline(0, 1, lwd = 2)
```

References

- Hammad, M.A. et al. (2020). Methods and Models for Electric Load Forecasting: A Comprehensive Review. Logistics & Sustainable Transport, 11 (1), 51–76. Available from https://doi.org/10.2478/jlst-2020-0004.
- https://thenewstack.io/3-new-techniques-for-data-dimensionality-reduction-in-machine-learning/
- https://www.delftstack.com/howto/r/visualize-confusion-matrix-in-r/#use-theconfusionmatrix-function-to-create-a-confusion-matrix-in-r
- (lonvia), S., 2022. The importance of normalization. [online] Nominatim.org. Available at: [Accessed 7 May 2022].
- Khintibidze, L., 2022. Visualize Confusion Matrix Using Caret Package in R. [online] Delft Stack. Available at: [Accessed 8 May 2022]