Informatics Institute of Technology
Department of Computing
(B.Sc.) in Computer Science

Module: 5DATA001C

Machine Learning and Data Mining

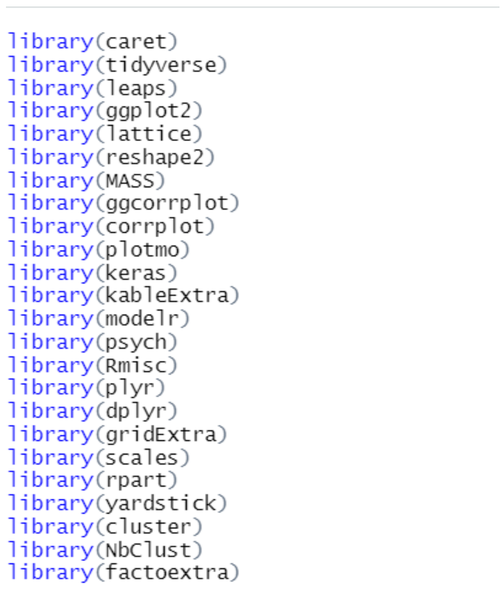
Module Leader - Mr. Achala Aponso

Coursework

Date of Submission: 15th May 2022
Student IIT ID: 20200651
Student UOW ID: w1833691
Tutorial Group: E
Student Name: Ajeevan Sivanandhan

1st Objective (Partition Clustering)

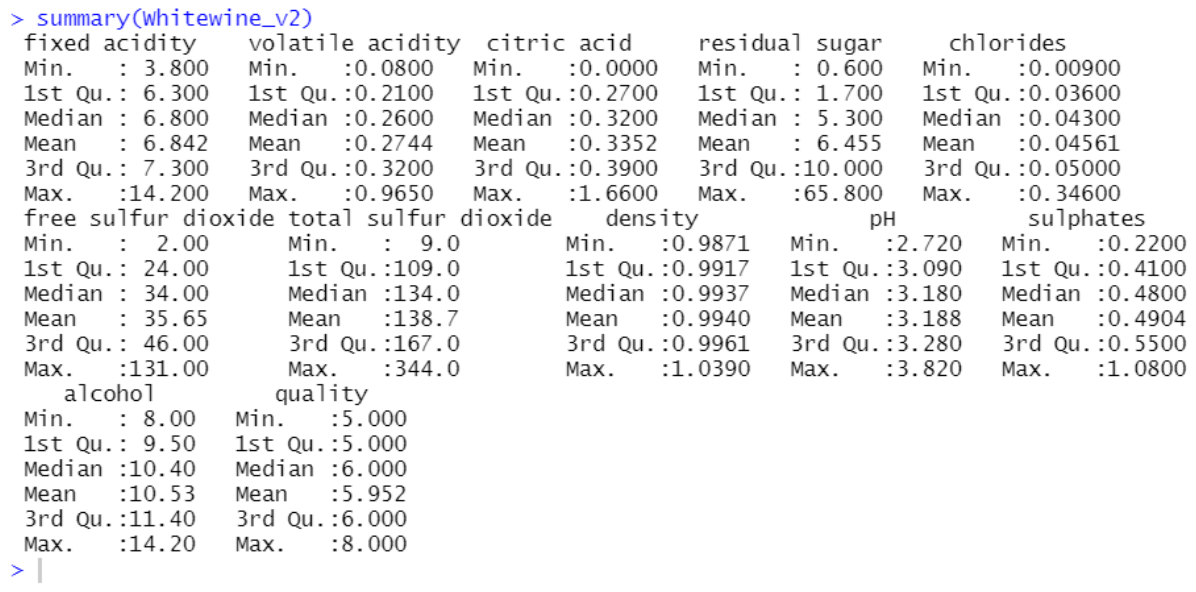
First and foremost, import all the required libraries in order to perform all the tasks and specifications to be performed.



```
library(caret)
library(tidyverse)
library(leaps)
library(ggplot2)
library(lattice)
library(reshape2)
library(MASS)
library(ggcorrplot)
library(corrplot)
library(plotmo)
library(keras)
library(kableExtra)
library(modelr)
library(psych)
library(Rmisc)
library(plyr)
library(dplyr)
library(gridExtra)
library(scales)
library(rpart)
library(yardstick)
library(cluster)
library(NbClust)
library(factoextra)
```

Figure 1: Importing Libraries

Next step is to import the required Dataset and get the summary of the Dataset



```
> summary(Whitewine_v2)
 fixed acidity    volatile acidity  citric acid     residual sugar     chlorides      
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900  
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600  
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.300   Median :0.04300  
 Mean   : 6.842   Mean   :0.2744   Mean   :0.3352   Mean   : 6.455   Mean   :0.04561  
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.:10.000   3rd Qu.:0.05000  
 Max.   :14.200   Max.   :0.9650   Max.   :1.6600   Max.   :65.800   Max.   :0.34600  
 free sulfur dioxide total sulfur dioxide    density             pH          sulphates     
 Min.   :  2.00      Min.   :  9.0          Min.   :0.9871   Min.   :2.720   Min.   :0.2200  
 1st Qu.: 24.00      1st Qu.:109.0          1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100  
 Median : 34.00      Median :134.0          Median :0.9937   Median :3.180   Median :0.4800  
 Mean   : 35.65      Mean   :138.7          Mean   :0.9940   Mean   :3.188   Mean   :0.4904  
 3rd Qu.: 46.00      3rd Qu.:167.0          3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500  
 Max.   :131.00      Max.   :344.0          Max.   :1.0390   Max.   :3.820   Max.   :1.0800  
    alcohol         quality     
 Min.   : 8.00   Min.   :5.000  
 1st Qu.: 9.50   1st Qu.:5.000  
 Median :10.40   Median :6.000  
 Mean   :10.53   Mean   :5.952  
 3rd Qu.:11.40   3rd Qu.:6.000  
 Max.   :14.20   Max.   :8.000  
> 
```

Figure 2: Summary of the Dataset

Next the Preprocessing tasks such as Scaling and Outlies removal, need to be done to the Data set given in order get a significant and accurate data. Outliers can generate issues such as incorrect data or forecast outcomes. Outliers increase data variability, which reduces statistical power.

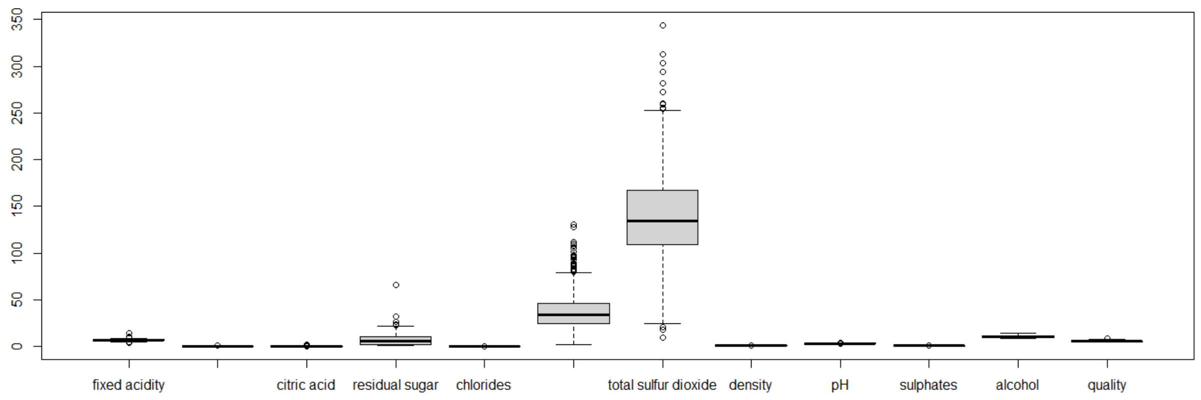
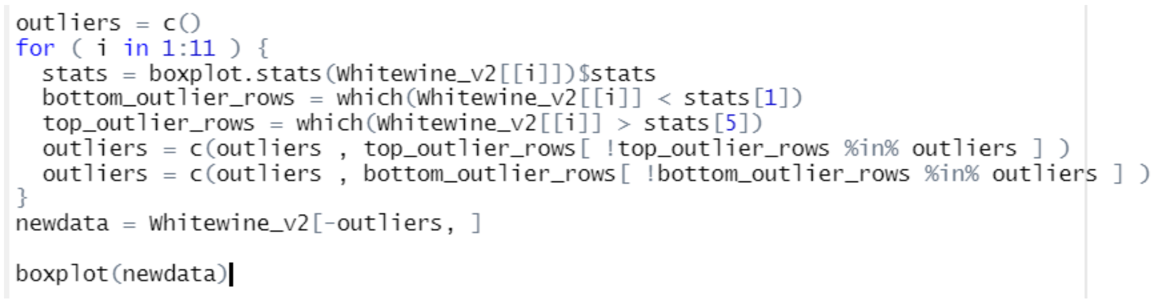


Figure 3: Box Plot before removing outliers

Removing Outliers for the data set given, and generating a new data set ("newdata")



```
outliers = c()
for ( i in 1:11 ) {
  stats = boxplot.stats(Whitewine_v2[[i]])$stats
  bottom_outlier_rows = which(Whitewine_v2[[i]] < stats[1])
  top_outlier_rows = which(Whitewine_v2[[i]] > stats[5])
  outliers = c(outliers , top_outlier_rows[ !top_outlier_rows %in% outliers ] )
  outliers = c(outliers , bottom_outlier_rows[ !bottom_outlier_rows %in% outliers ] )
}
newdata = Whitewine_v2[-outliers, ]

boxplot(newdata)
```

Figure 5: Removing Outliers

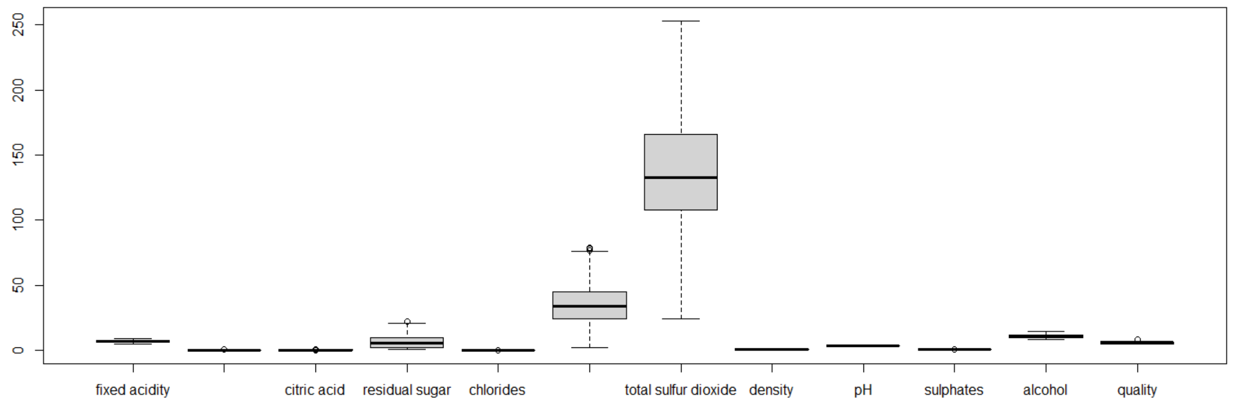


Figure 4:Box plot after removing outliers

As a part of preprocessing Scaling was also done to this data, using **scale** function,

```
### Scaling Data ####
scaleData <- scale(newdata)
```

Box Plot after Scaling the Dataset:

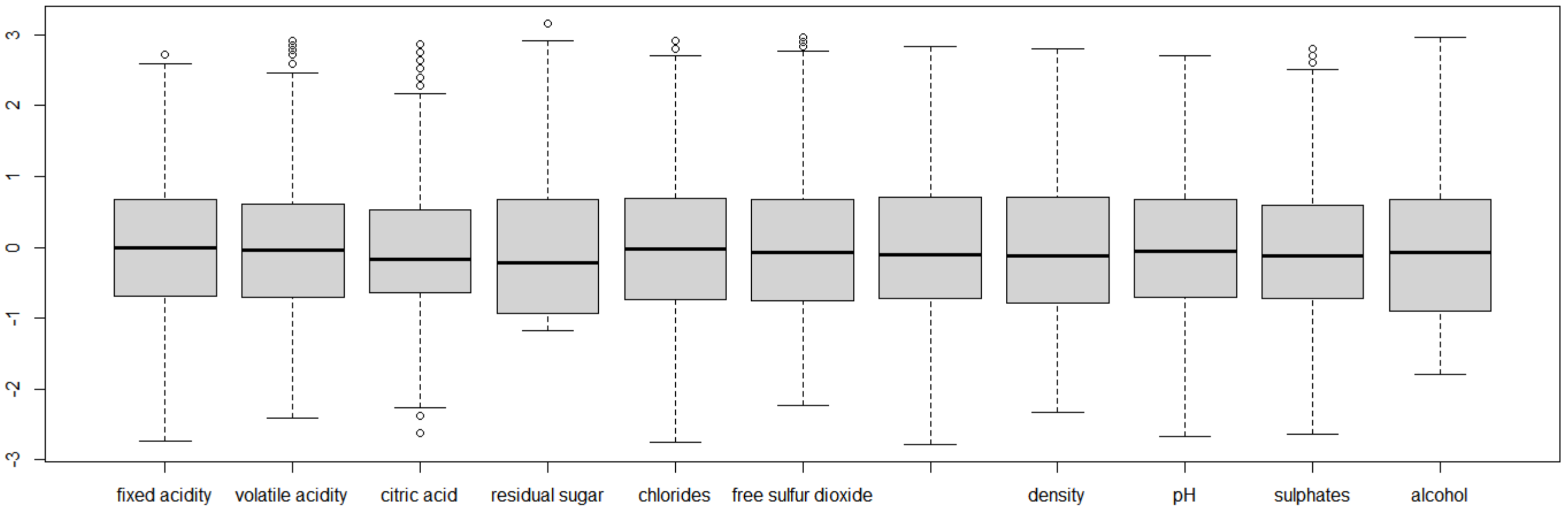
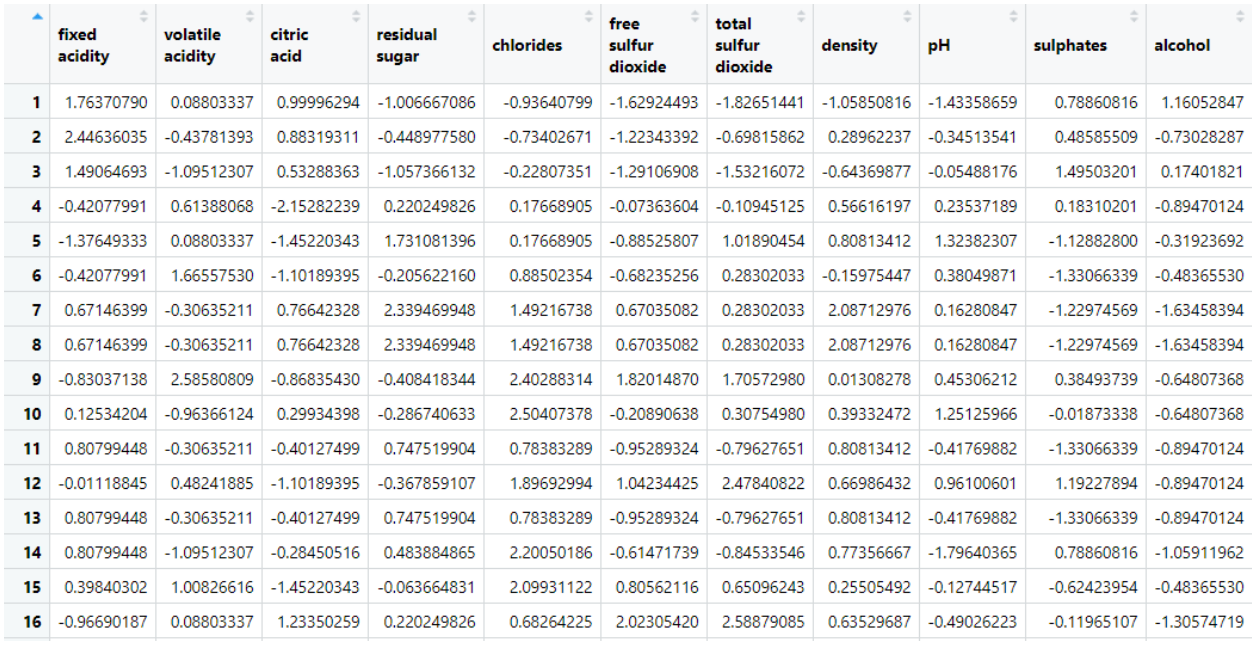


Figure 6: Box plot after scaling



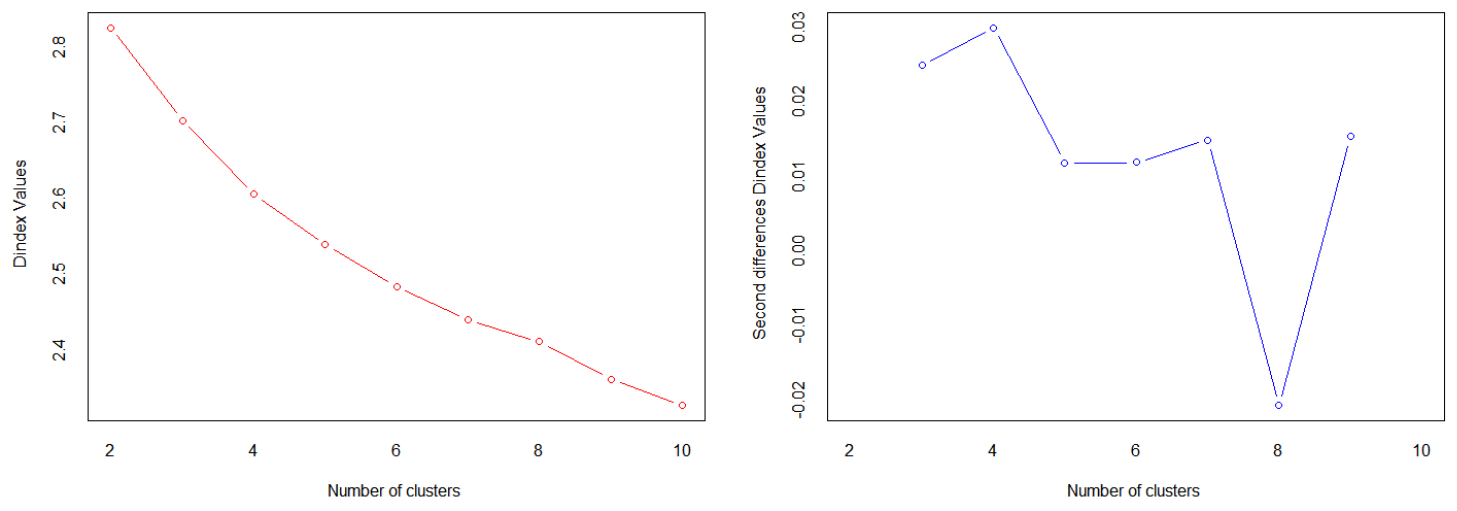
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
1	1.76370790	0.08803337	0.99996294	-1.006667086	-0.93640799	-1.62924493	-1.82651441	-1.05850816	-1.43358659	0.78860816	1.16052847
2	2.44636035	-0.43781393	0.88319311	-0.448977580	-0.73402671	-1.22343392	-0.69815862	0.28962237	-0.34513541	0.48585509	-0.73028287
3	1.49064693	-1.09512307	0.53288363	-1.057366132	-0.22807351	-1.29106908	-1.53216072	-0.64369877	-0.05488176	1.49503201	0.17401821
4	-0.42077991	0.61388068	-2.15282239	0.220249826	0.17668905	-0.07363604	-0.10945125	0.56616197	0.23537189	0.18310201	-0.89470124
5	-1.37649333	0.08803337	-1.45220343	1.731081396	0.17668905	-0.88525807	1.01890454	0.80813412	1.32382307	-1.12882800	-0.31923692
6	-0.42077991	1.66557530	-1.10189395	-0.205622160	0.88502354	-0.68235256	0.28302033	-0.15975447	0.38049871	-1.33066339	-0.48365530
7	0.67146399	-0.30635211	0.76642328	2.339469948	1.49216738	0.67035082	0.28302033	2.08712976	0.16280847	-1.22974569	-1.63458394
8	0.67146399	-0.30635211	0.76642328	2.339469948	1.49216738	0.67035082	0.28302033	2.08712976	0.16280847	-1.22974569	-1.63458394
9	-0.83037138	2.58580809	-0.86835430	-0.408418344	2.40288314	1.82014870	1.70572980	0.01308278	0.45306212	0.38493739	-0.64807368
10	0.12534204	-0.96366124	0.29934398	-0.286740633	2.50407378	-0.20890638	0.30754980	0.39332472	1.25125966	-0.01873338	-0.64807368
11	0.80799448	-0.30635211	-0.40127499	0.747519904	0.78383289	-0.95289324	-0.79627651	0.80813412	-0.41769882	-1.33066339	-0.89470124
12	-0.01118845	0.48241885	-1.10189395	-0.367859107	1.89692994	1.04234425	2.47840822	0.66986432	0.96100601	1.19227894	-0.89470124
13	0.80799448	-0.30635211	-0.40127499	0.747519904	0.78383289	-0.95289324	-0.79627651	0.80813412	-0.41769882	-1.33066339	-0.89470124
14	0.80799448	-1.09512307	-0.28450516	0.483884865	2.20050186	-0.61471739	-0.84533546	0.77356667	-1.79640365	0.78860816	-1.05911962
15	0.39840302	1.00826616	-1.45220343	-0.063664831	2.09931122	0.80562116	0.65096243	0.25505492	-0.12744517	-0.62423954	-0.48365530
16	-0.96690187	0.08803337	1.23350259	0.220249826	0.68264225	2.02305420	2.58879085	0.63529687	-0.49026223	-0.11965107	-1.30574719

Figure 7: Final Dataset after scaling

Next step is to Define the Cluster Centers as required, Using Automated and manual tools

NBClust Method

According to the NBClust method the best number of clusters is considered as 2

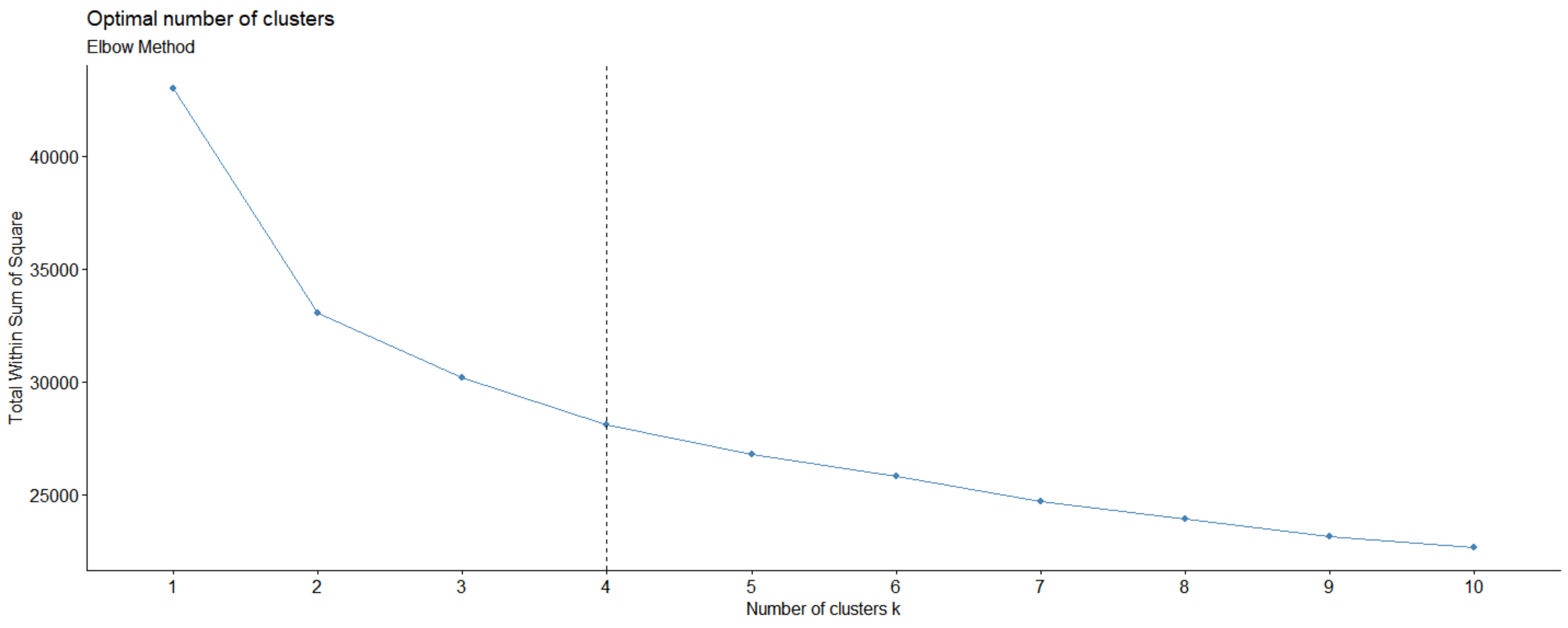


```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index 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
* 5 proposed 3 as the best number of clusters
* 2 proposed 4 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 2 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


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

Elbow Method

The elbow approach examines the proportion of variation explained as a function of cluster number: A number of clusters should be chosen so that adding another cluster does not significantly improve data modeling.

```
#finding the number of clusters(k) automatically by the elbow method.
fviz_nbclust(wineDataset_final, kmeans, method = "wss")+
  geom_vline(xintercept = 4, linetype = 2)+
  labs(subtitle = "Elbow Method")
```



Therefore, for k=4 the ratio between **sum of squares/total sum of squares ratio** tends to change slowly and remain less changing so that 4 is considered as the best number of Clusters.

Silhouette Method

The silhouette Method is also used to determine the ideal number of clusters as well as to interpret and validate consistency within data clusters. The silhouette technique computes silhouette coefficients for each point, which quantify how similar a point is to its own cluster in comparison to other clusters. According to this method the best number of Clusters is considered as **2**

```
########################## Silhoutte Method ########

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

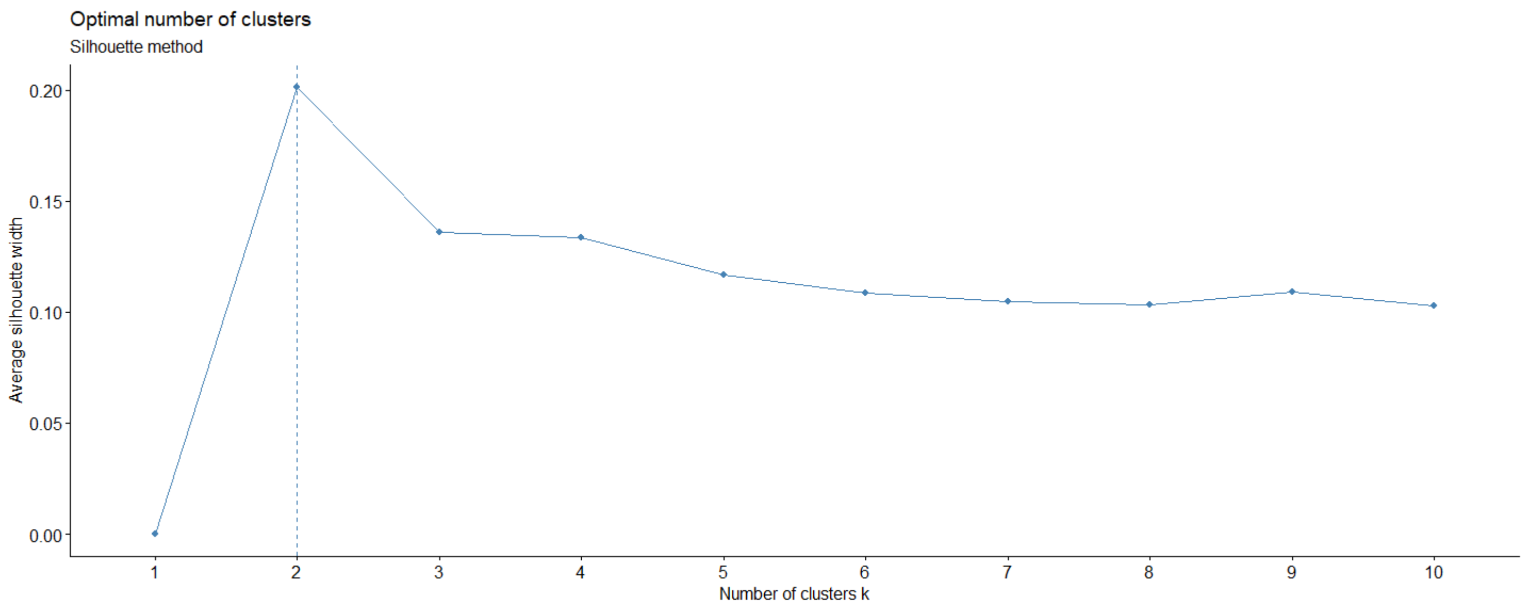
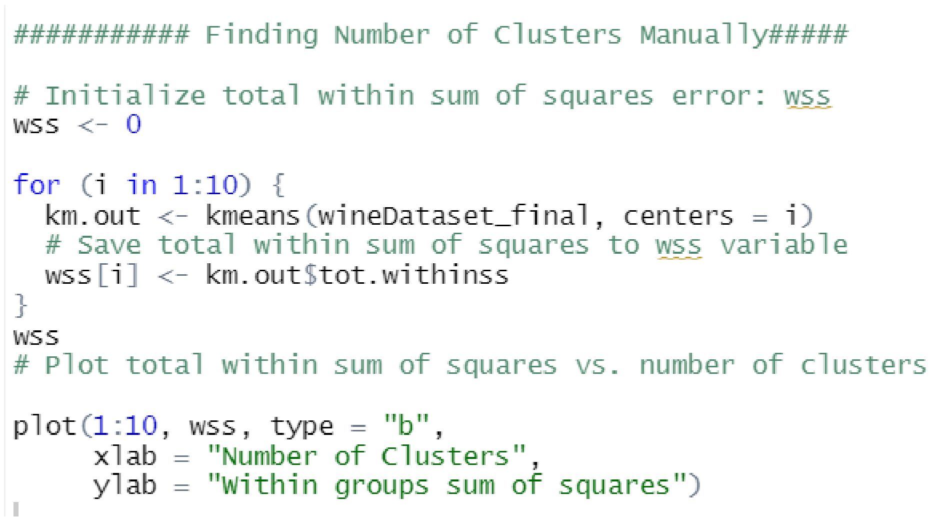
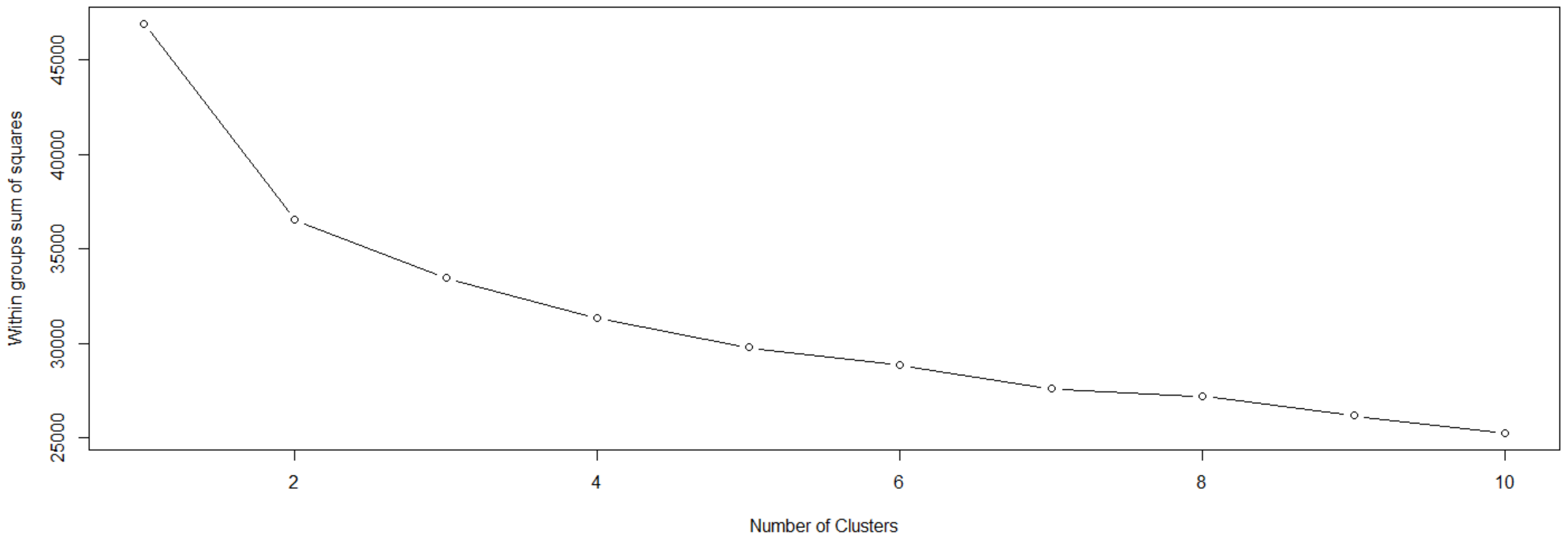


Figure 9 : Silhouette method

Defining Number of Clusters Manually



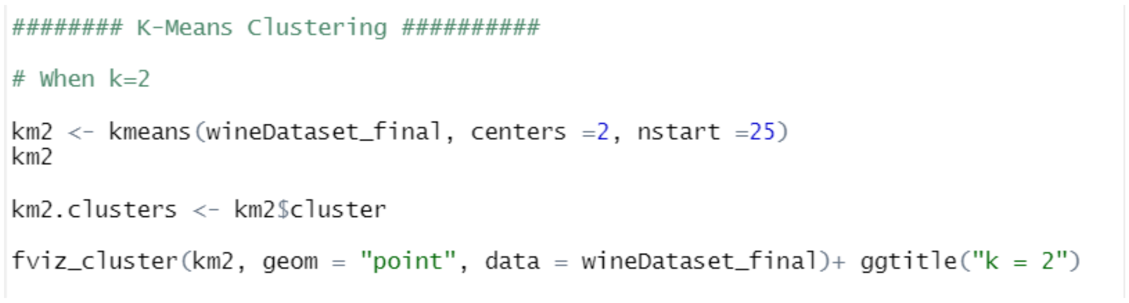
```
########### Finding Number of Clusters Manually#####

# Initialize total within sum of squares error: wss
wss <- 0

for (i in 1:10) {
  km.out <- kmeans(wineDataset_final, centers = i)
  # Save total within sum of squares to wss variable
  wss[i] <- km.out$tot.withinss
}
wss
# Plot total within sum of squares vs. number of clusters

plot(1:10, wss, type = "b",
     xlab = "Number of Clusters",
     ylab = "Within groups sum of squares")
```



The graph above depicts the variance within the clusters. It decreases as K rises, with a bend visible at k = 4. According to this bent, further clusters beyond the fourth have low utility.

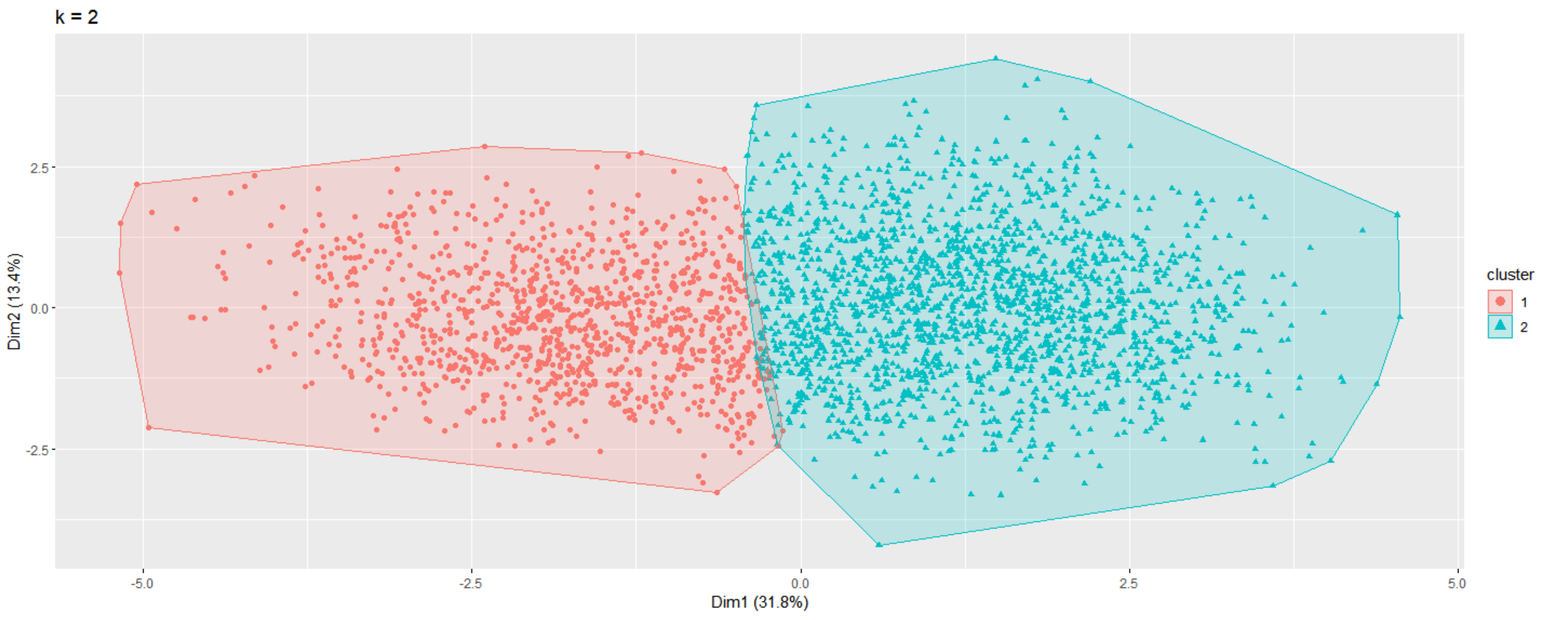
Applying K means Clustering to the Data with K=2, 3 and 4.

When K=2

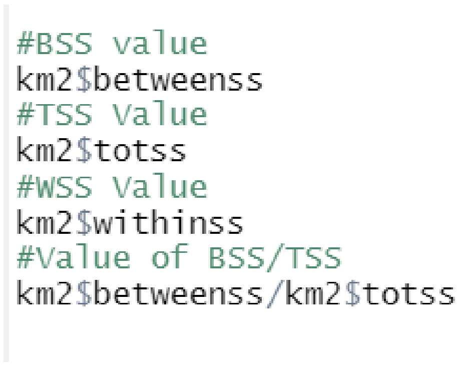


```
######## K-Means Clustering ##########

# When k=2

km2 <- kmeans(wineDataset_final, centers =2, nstart =25)
km2

km2.clusters <- km2$cluster

fviz_cluster(km2, geom = "point", data = wineDataset_final)+ ggtitle("k = 2")
```

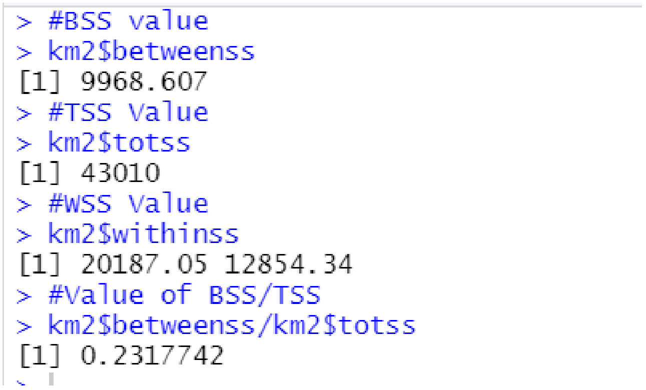
Cluster plot When K=2



Finding all the BSS,TSS, WSS, BSS/TSS values

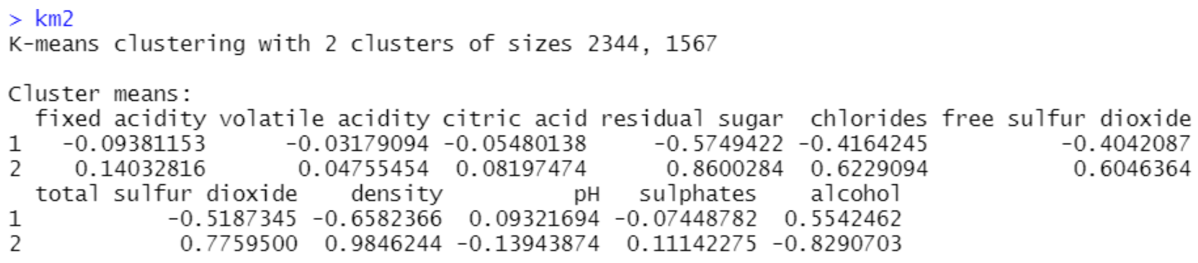


```
#BSS value
km2$betweenss
#TSS Value
km2$totss
#WSS Value
km2$withinss
#Value of BSS/TSS
km2$betweenss/km2$totss
```

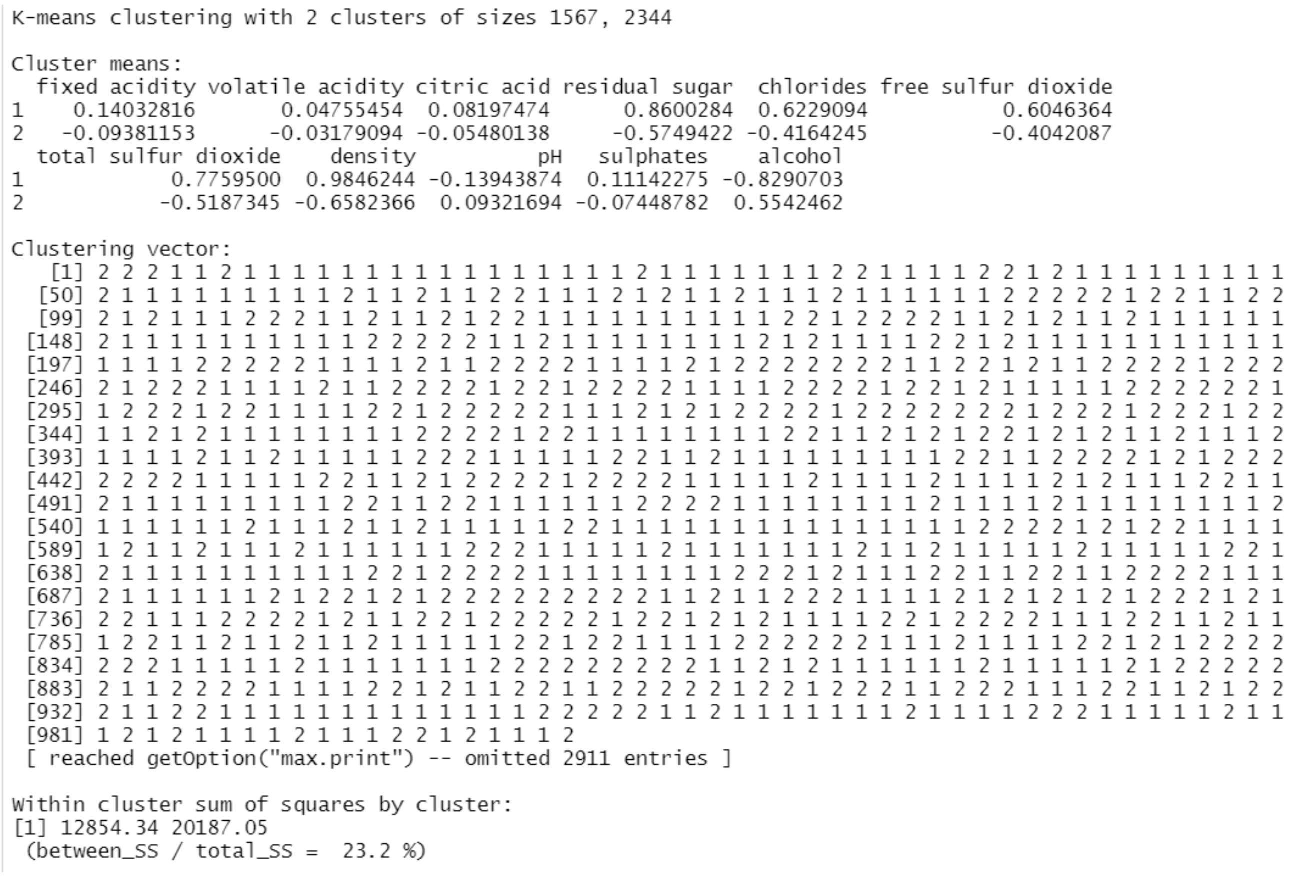


```
> #BSS value
> km2$betweenss
[1] 9968.607
> #TSS Value
> km2$totss
[1] 43010
> #WSS Value
> km2$withinss
[1] 20187.05 12854.34
> #Value of BSS/TSS
> km2$betweenss/km2$totss
[1] 0.2317742
```

Centers



```
> km2
K-means clustering with 2 clusters of sizes 2344, 1567

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1   -0.09381153      -0.03179094 -0.05480138     -0.5749422 -0.4164245          -0.4042087
2    0.14032816       0.04755454  0.08197474      0.8600284  0.6229094           0.6046364
  total sulfur dioxide    density          pH   sulphates    alcohol
1           -0.5187345 -0.6582366  0.09321694 -0.07448782  0.5542462
2            0.7759500  0.9846244 -0.13943874  0.11142275 -0.8290703
```

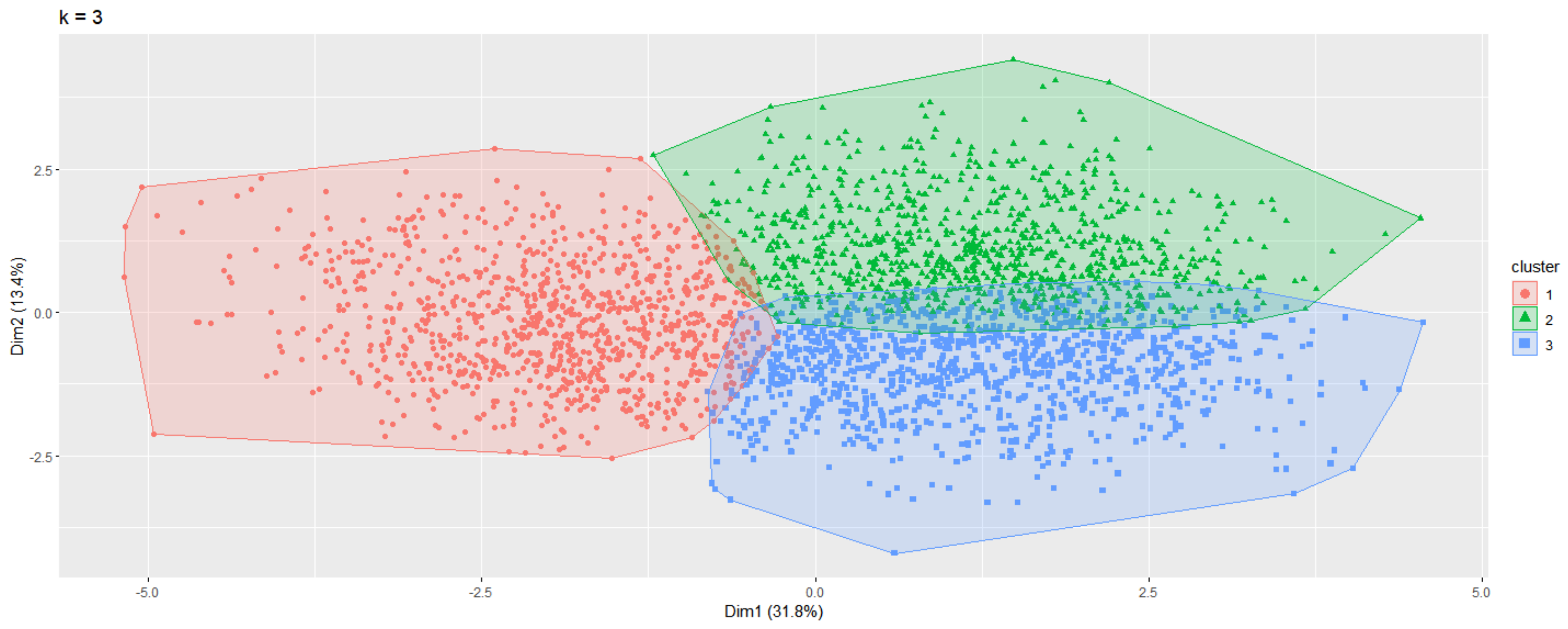


```
K-means clustering with 2 clusters of sizes 1567, 2344

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1    0.14032816       0.04755454  0.08197474      0.8600284  0.6229094           0.6046364
2   -0.09381153      -0.03179094 -0.05480138     -0.5749422 -0.4164245          -0.4042087
  total sulfur dioxide    density          pH   sulphates    alcohol
1            0.7759500  0.9846244 -0.13943874  0.11142275 -0.8290703
2           -0.5187345 -0.6582366  0.09321694 -0.07448782  0.5542462

Clustering vector:
   [1] 2 2 2 1 1 2 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
  [50] 2 1 1 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 1 1 2 2 2 2 2 1 2 2 1 1 2 2
  [99] 2 1 2 1 1 1 2 2 2 1 1 2 1 1 2 1 2 2 1 1 1 1 1 1 1 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
 [148] 2 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 1 2 1 2 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1
 [197] 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 2
 [246] 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 1 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 1 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
 [344] 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 1 2 1 2 1 2 2 1 2 1 2 1 1 2 1 1 1 2
 [393] 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 2 2 1 1 2 2 2 2 1 2 1 2 2 2
 [442] 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 1 1 2 1 1 1 1 2 1 2 1 1 1 2 2 1 1
 [491] 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 2
 [540] 1 1 1 1 1 1 2 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 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 2 2 1
 [638] 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 2 1 1 1 2 2 1 1 2 2 1 1 2 2 2 2 1 1 1
 [687] 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 1 1 2 2 2 1 1 1 1 2 1 2 1 2 1 2 1 2 2 2 2 1 2 1
 [736] 2 2 1 1 1 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
 [785] 1 2 2 1 1 2 1 1 2 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 1 1 2 2 1 2 1 2 2 2 2
 [834] 2 2 2 1 1 1 1 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 2 2 1 1 2 1 2 2
 [932] 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 1 1 2 1 1
 [981] 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] 12854.34 20187.05
 (between_SS / total_SS =  23.2 %)
```

When k=3

```
# When k=3
km3 <- kmeans(wineDataset_final, centers =3, nstart =25)
km3

km3.clusters <- km3$cluster

fviz_cluster(km3, geom = "point", data = wineDataset_final)+ ggtitle("k = 3")
```

Cluster Plot when k=3

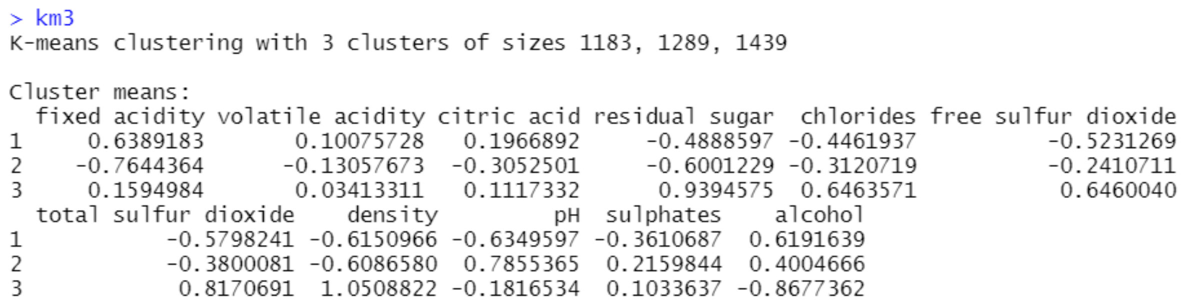


Finding all the BSS,TSS, WSS, BSS/TSS values

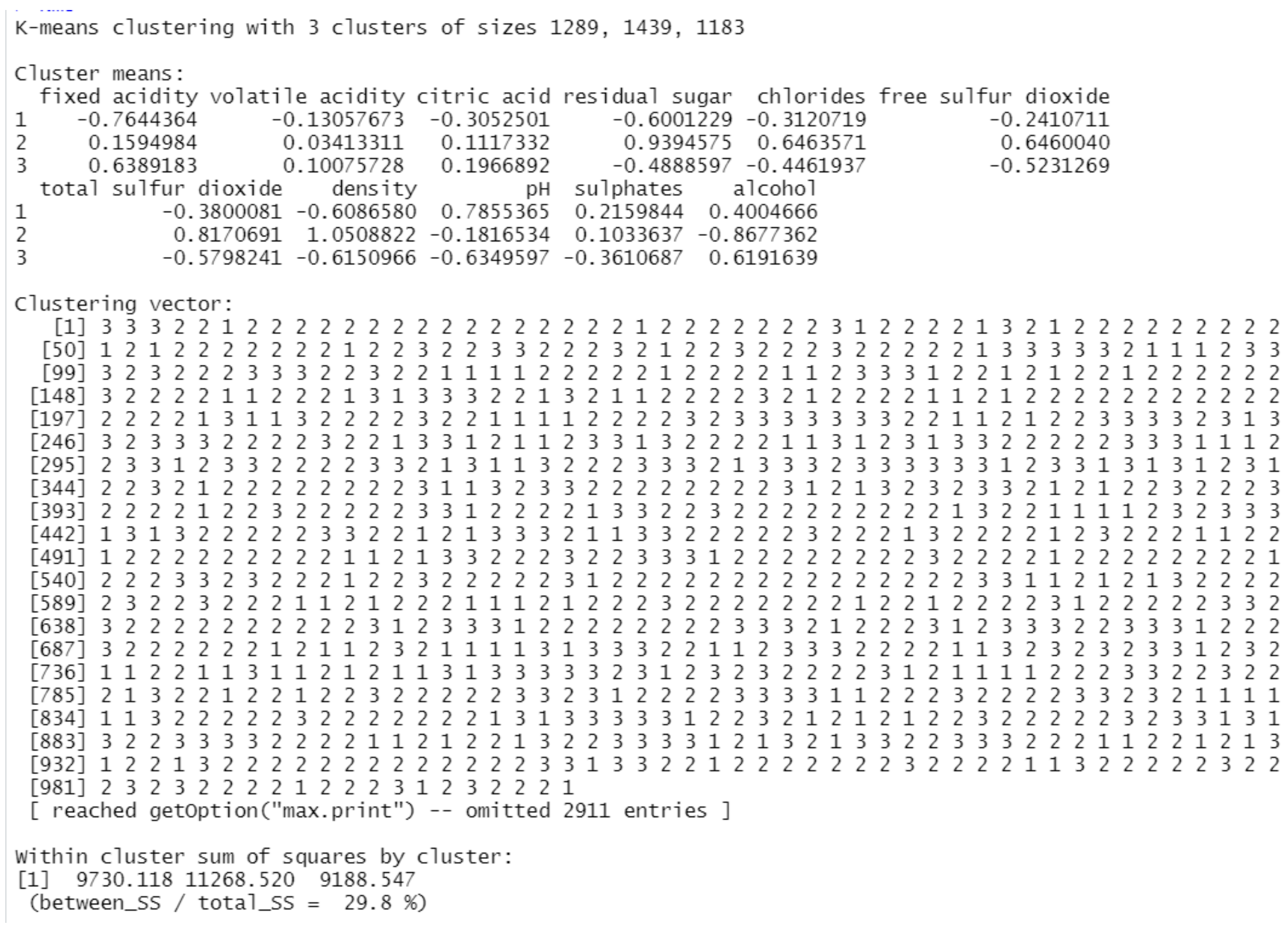
```
#BSS value
km3$betweenss
#TSS Value
km3$totss
#WSS Value
km3$withinss
#Value of BSS/TSS
km3$betweenss/km3$totss
```

```
> #BSS value
> km3$betweenss
[1] 12822.81
> #TSS Value
> km3$totss
[1] 43010
> #WSS Value
> km3$withinss
[1] 11268.520  9188.547  9730.118
> #Value of BSS/TSS
> km3$betweenss/km3$totss
[1] 0.2981357
```

Centers



```
> km3
K-means clustering with 3 clusters of sizes 1183, 1289, 1439

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1     0.6389183       0.10075728   0.1966892     -0.4888597 -0.4461937           -0.5231269
2    -0.7644364      -0.13057673  -0.3052501     -0.6001229 -0.3120719           -0.2410711
3     0.1594984       0.03413311   0.1117332      0.9394575  0.6463571            0.6460040
  total sulfur dioxide    density         pH  sulphates    alcohol
1           -0.5798241 -0.6150966 -0.6349597 -0.3610687  0.6191639
2           -0.3800081 -0.6086580  0.7855365  0.2159844  0.4004666
3            0.8170691  1.0508822 -0.1816534  0.1033637 -0.8677362
```

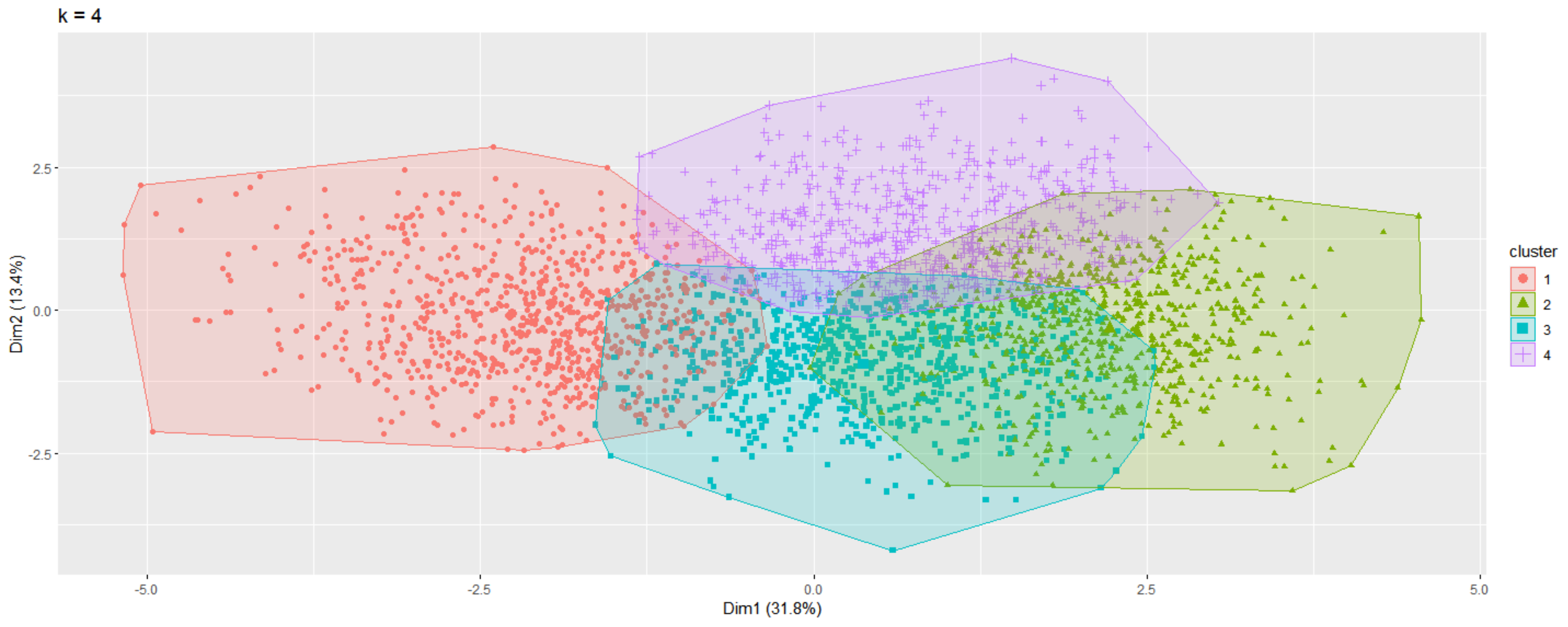


```
K-means clustering with 3 clusters of sizes 1289, 1439, 1183

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1    -0.7644364      -0.13057673  -0.3052501     -0.6001229 -0.3120719           -0.2410711
2     0.1594984       0.03413311   0.1117332      0.9394575  0.6463571            0.6460040
3     0.6389183       0.10075728   0.1966892     -0.4888597 -0.4461937           -0.5231269
  total sulfur dioxide    density         pH  sulphates    alcohol
1           -0.3800081 -0.6086580  0.7855365  0.2159844  0.4004666
2            0.8170691  1.0508822 -0.1816534  0.1033637 -0.8677362
3           -0.5798241 -0.6150966 -0.6349597 -0.3610687  0.6191639

Clustering vector:
   [1] 3 3 3 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 2 3 1 2 2 2 2 1 3 2 1 2 2 2 2 2 2 2 2 2
  [50] 1 2 1 2 2 2 2 2 2 2 1 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 3 3 3 3 3 2 1 1 1 2 3 3
  [99] 3 2 3 2 2 2 3 3 3 2 2 3 2 2 1 1 1 1 2 2 2 2 2 1 2 2 2 2 1 1 2 3 3 3 1 2 2 1 2 1 2 2 1 2 2 2 2 2 2
 [148] 3 2 2 2 2 1 1 2 2 2 1 3 1 3 3 3 2 2 1 3 2 1 1 2 2 2 2 3 2 1 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 2 2 2
 [197] 2 2 2 2 1 3 1 1 3 2 2 2 2 3 2 2 1 1 1 1 2 2 2 2 3 2 3 3 3 3 3 3 3 2 2 1 1 2 1 2 2 3 3 3 3 2 3 1 3
 [246] 3 2 3 3 3 2 2 2 2 3 2 2 1 3 3 1 2 1 1 2 3 3 1 3 2 2 2 2 1 1 3 1 2 3 1 3 3 2 2 2 2 2 3 3 3 1 1 1 2
 [295] 2 3 3 1 2 3 3 2 2 2 2 3 3 2 1 3 1 1 3 2 2 2 3 3 3 2 1 3 3 3 2 3 3 3 3 3 3 1 2 3 3 1 3 1 3 1 2 3 1
 [344] 2 2 3 2 1 2 2 2 2 2 2 2 2 3 1 1 3 2 3 3 2 2 2 2 2 2 2 2 3 1 2 1 3 2 3 2 3 3 2 1 2 1 2 2 3 2 2 2 3
 [393] 2 2 2 2 1 2 2 3 2 2 2 2 2 3 3 1 2 2 2 2 1 3 3 2 2 3 2 2 2 2 2 2 2 2 2 1 3 2 2 1 1 1 1 2 3 2 3 3 3
 [442] 1 3 1 3 2 2 2 2 2 3 3 2 2 1 2 1 3 3 3 2 1 1 3 3 2 2 2 2 2 3 2 2 2 1 3 2 2 2 2 1 2 3 2 2 2 1 1 2 2
 [491] 1 2 2 2 2 2 2 2 2 2 2 1 1 2 1 3 3 2 2 2 3 2 2 3 3 3 1 2 2 2 2 2 2 2 2 3 2 2 2 2 1 2 2 2 2 2 2 2 1
 [540] 2 2 2 3 3 2 3 2 2 2 1 2 2 3 2 2 2 2 2 3 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 1 2 1 2 1 3 2 2 2 2
 [589] 2 3 2 2 3 2 2 2 1 1 2 1 2 2 2 1 1 1 2 1 2 2 2 3 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 3 1 2 2 2 2 2 3 3 2
 [638] 3 2 2 2 2 2 2 2 2 2 2 3 1 2 3 3 3 1 2 2 2 2 2 2 2 2 3 3 3 2 1 2 2 2 3 1 2 3 3 3 2 2 3 3 3 1 2 2 2
 [687] 3 2 2 2 2 2 2 1 2 1 1 2 3 2 1 1 1 1 3 1 3 3 3 2 2 1 1 2 3 3 3 2 2 2 2 1 1 3 2 3 2 3 2 3 3 1 2 3 2
 [736] 1 1 2 2 1 1 3 1 1 2 1 2 1 1 3 1 3 3 3 3 3 2 3 1 2 3 2 3 2 2 2 2 3 1 2 1 1 1 1 2 2 2 3 3 2 2 3 2 2
 [785] 2 1 3 2 2 1 2 2 1 2 2 3 2 2 2 2 2 3 3 2 3 1 2 2 2 2 3 3 3 3 1 1 2 2 2 3 2 2 2 2 3 3 2 3 2 1 1 1 1
 [834] 1 1 3 2 2 2 2 2 3 2 2 2 2 2 2 2 1 3 1 3 3 3 3 3 1 2 2 3 2 1 2 1 2 1 2 2 3 2 2 2 2 2 3 2 3 3 1 3 1
 [883] 3 2 2 3 3 3 3 2 2 2 2 1 1 2 1 2 2 1 3 2 2 3 3 3 3 1 2 1 3 2 1 3 3 2 2 3 3 3 2 2 2 1 1 2 2 1 2 1 3
 [932] 1 2 2 1 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 3 3 2 2 1 2 2 2 2 2 2 2 3 2 2 2 2 1 1 3 2 2 2 2 2 3 2 2
 [981] 2 3 2 3 2 2 2 2 1 2 2 2 3 1 2 3 2 2 2 1
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1]  9730.118 11268.520  9188.547
 (between_SS / total_SS =  29.8 %)
```

When k=4

```r
# When k=4

km4 <- kmeans(wineDataset_final, centers =4, nstart =25)
km4

km4.clusters <- km4$cluster

fviz_cluster(km4, geom = "point", data = wineDataset_final)+ ggtitle("k = 4")
```

Cluster Plot when k=4

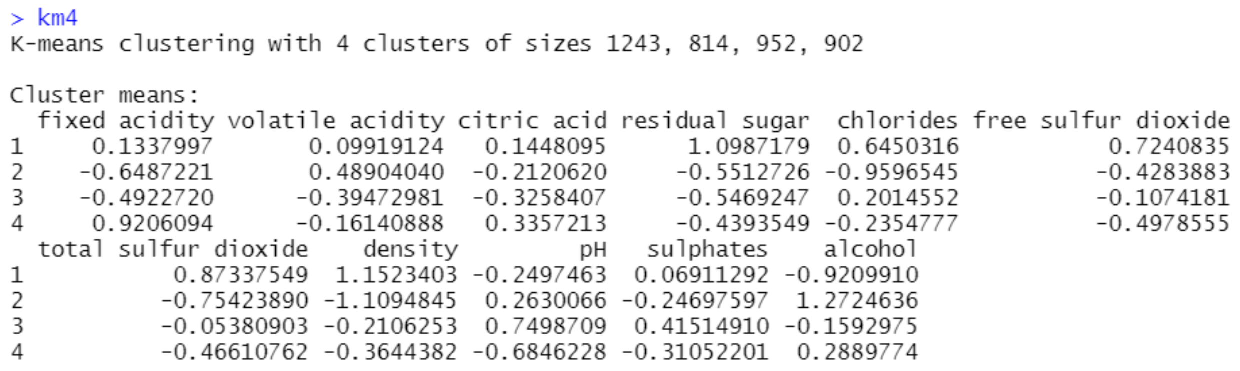


Finding all the BSS, TSS, WSS, BSS/TSS values

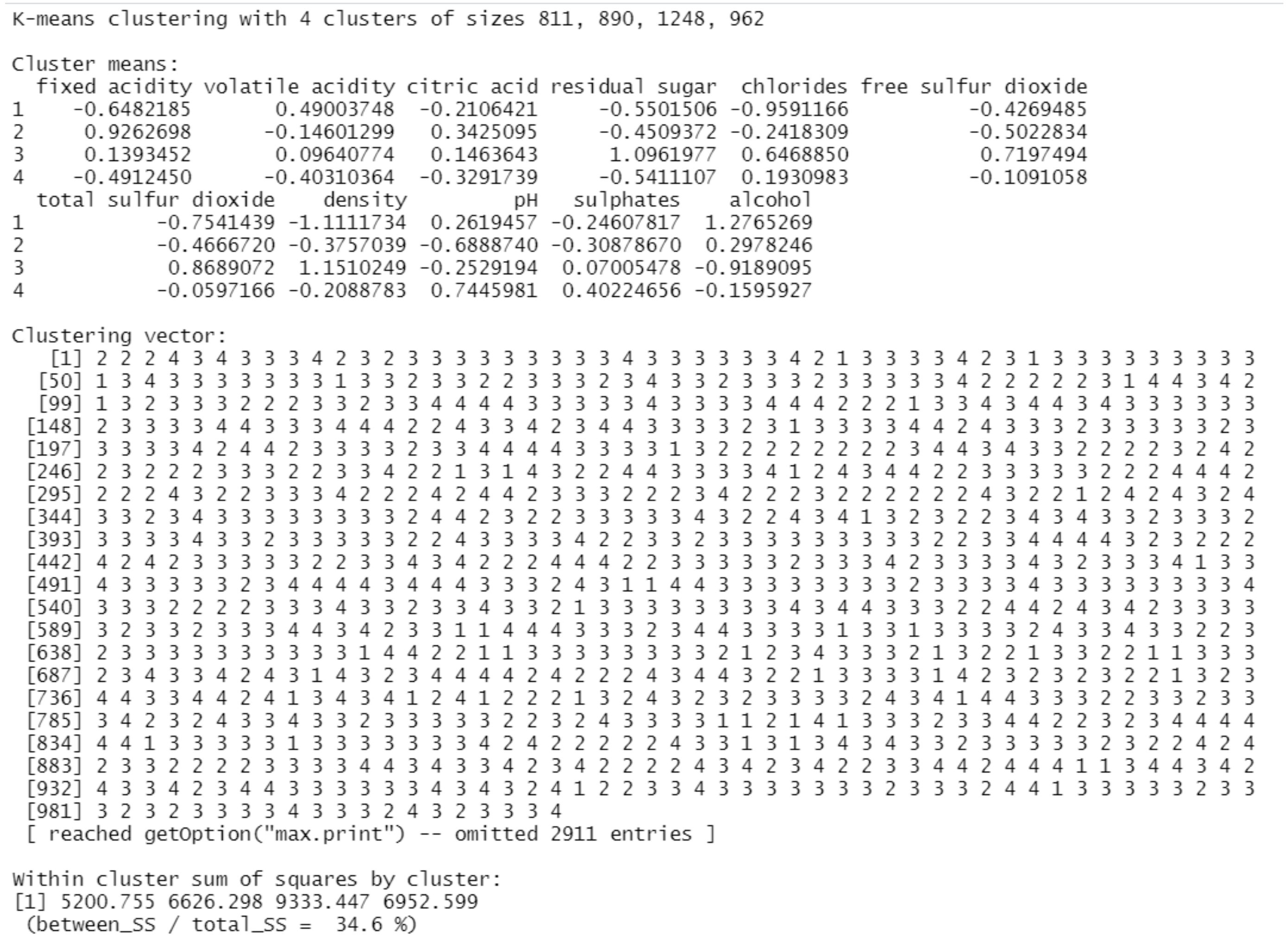
```r
#BSS value
km4$betweenss
#TSS Value
km4$totss
#WSS Value
km4$withinss
#Value of BSS/TSS
km4$betweenss/km4$totss
```

```
> #BSS value
> km4$betweenss
[1] 14896.93
> #TSS Value
> km4$totss
[1] 43010
> #WSS Value
> km4$withinss
[1] 9292.107 5231.654 6868.037 6721.270
> #Value of BSS/TSS
> km4$betweenss/km4$totss
[1] 0.3463597
> 
```

Centers

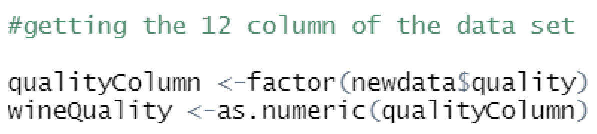


```
> km4
K-means clustering with 4 clusters of sizes 1243, 814, 952, 902

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1     0.1337997       0.09919124   0.1448095      1.0987179  0.6450316           0.7240835
2    -0.6487221       0.48904040  -0.2120620     -0.5512726 -0.9596545          -0.4283883
3    -0.4922720      -0.39472981  -0.3258407     -0.5469247  0.2014552          -0.1074181
4     0.9206094      -0.16140888   0.3357213     -0.4393549 -0.2354777          -0.4978555
  total sulfur dioxide    density         pH  sulphates    alcohol
1           0.87337549  1.1523403 -0.2497463  0.06911292 -0.9209910
2          -0.75423890 -1.1094845  0.2630066 -0.24697597  1.2724636
3          -0.05380903 -0.2106253  0.7498709  0.41514910 -0.1592975
4          -0.46610762 -0.3644382 -0.6846228 -0.31052201  0.2889774
```



```
K-means clustering with 4 clusters of sizes 811, 890, 1248, 962

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1    -0.6482185       0.49003748  -0.2106421     -0.5501506 -0.9591166          -0.4269485
2     0.9262698      -0.14601299   0.3425095     -0.4509372 -0.2418309          -0.5022834
3     0.1393452       0.09640774   0.1463643      1.0961977  0.6468850           0.7197494
4    -0.4912450      -0.40310364  -0.3291739     -0.5411107  0.1930983          -0.1091058
  total sulfur dioxide    density         pH  sulphates    alcohol
1           -0.7541439 -1.1111734  0.2619457 -0.24607817  1.2765269
2           -0.4666720 -0.3757039 -0.6888740 -0.30878670  0.2978246
3            0.8689072  1.1510249 -0.2529194  0.07005478 -0.9189095
4           -0.0597166 -0.2088783  0.7445981  0.40224656 -0.1595927

Clustering vector:
   [1] 2 2 2 4 3 4 3 3 3 4 2 3 2 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 3 4 2 1 3 3 3 3 4 2 3 1 3 3 3 3 3 3 3 3 3
  [50] 1 3 4 3 3 3 3 3 3 3 1 3 3 2 3 3 2 2 3 3 3 2 3 4 3 3 2 3 3 3 2 3 3 3 3 3 4 2 2 2 2 2 3 1 4 4 3 4 2
  [99] 1 3 2 3 3 3 2 2 2 3 3 2 3 3 4 4 4 4 3 3 3 3 3 4 3 3 3 3 4 4 4 2 2 2 1 3 3 4 3 4 4 3 4 3 3 3 3 3 3
 [148] 2 3 3 3 3 4 4 3 3 3 4 4 4 2 2 4 3 3 4 2 3 4 4 3 3 3 3 2 3 1 3 3 3 3 4 4 2 4 3 3 3 2 3 3 3 3 3 2 3
 [197] 3 3 3 3 4 2 4 4 2 3 3 3 3 2 3 3 4 4 4 4 3 3 3 3 1 3 2 2 2 2 2 2 2 2 3 4 4 3 4 3 3 2 2 2 2 3 2 4 2
 [246] 2 3 2 2 2 3 3 3 2 2 3 3 4 2 2 1 3 1 4 3 2 2 4 4 3 3 3 3 4 1 2 4 3 4 4 2 2 3 3 3 3 3 2 2 2 4 4 4 2
 [295] 2 2 2 4 3 2 2 3 3 3 4 2 2 2 4 2 4 4 2 3 3 3 2 2 2 3 4 2 2 2 3 2 2 2 2 2 2 4 3 2 2 1 2 4 2 4 3 2 4
 [344] 3 3 2 3 4 3 3 3 3 3 3 3 3 2 4 4 2 3 2 2 3 3 3 3 3 4 3 2 2 4 3 4 1 3 2 3 2 2 3 4 3 4 3 3 2 3 3 3 2
 [393] 3 3 3 3 4 3 3 2 3 3 3 3 3 2 2 4 3 3 3 3 4 2 2 3 3 2 3 3 3 3 3 3 3 3 3 3 2 2 3 3 4 4 4 4 3 2 3 2 2 2
 [442] 4 2 4 2 3 3 3 3 3 2 2 3 3 4 3 4 2 2 2 4 4 4 2 2 3 3 3 3 3 2 3 3 3 4 2 3 3 3 3 4 3 2 3 3 3 4 1 3 3
 [491] 4 3 3 3 3 3 2 3 4 4 4 4 3 4 4 4 3 3 3 2 4 3 1 1 4 4 3 3 3 3 3 3 3 3 3 2 3 3 3 3 4 3 3 3 3 3 3 3 3 4
 [540] 3 3 3 2 2 2 2 3 3 3 4 3 3 2 3 3 4 3 3 2 1 3 3 3 3 3 3 3 3 4 3 4 4 3 3 3 2 2 4 4 2 4 3 4 2 3 3 3 3
 [589] 3 2 3 3 2 3 3 3 4 4 3 4 2 3 3 1 1 4 4 4 3 3 3 2 3 4 4 3 3 3 3 1 3 3 1 3 3 3 3 2 4 3 3 4 3 3 2 2 3
 [638] 2 3 3 3 3 3 3 3 3 3 3 1 4 4 2 2 1 1 3 3 3 3 3 3 3 3 2 1 2 3 4 3 3 3 2 1 3 2 2 1 3 3 2 2 1 1 3 3 3
 [687] 2 3 4 3 3 4 2 4 3 1 4 3 2 3 4 4 4 4 2 4 2 2 2 4 3 4 4 3 2 2 1 3 3 3 3 1 4 2 3 2 3 2 3 2 2 1 3 2 3
 [736] 4 4 3 3 4 4 2 4 1 3 4 3 4 1 2 4 1 2 2 2 1 3 2 4 3 2 3 2 3 3 3 3 2 4 3 4 1 4 4 3 3 3 2 2 3 3 2 3 3
 [785] 3 4 2 3 2 4 3 3 4 3 3 2 3 3 3 3 3 2 2 3 2 4 3 3 3 3 1 1 2 1 4 1 3 3 3 2 3 3 4 4 2 2 3 2 3 4 4 4 4
 [834] 4 4 1 3 3 3 3 3 1 3 3 3 3 3 3 3 4 2 4 2 2 2 2 2 4 3 3 1 3 1 3 4 3 4 3 3 2 3 3 3 3 3 2 3 2 2 4 2 4
 [883] 2 3 3 2 2 2 2 3 3 3 3 4 4 3 4 3 3 4 2 3 4 2 2 2 2 4 3 4 2 3 4 2 2 3 3 4 4 2 4 4 4 1 1 3 4 4 3 4 2
 [932] 4 3 3 4 2 3 4 4 3 3 3 3 3 3 4 3 4 3 2 4 1 2 2 3 3 4 3 3 3 3 3 3 3 2 3 3 3 2 4 4 1 3 3 3 3 3 2 3 3
 [981] 3 2 3 2 3 3 3 3 4 3 3 3 2 4 3 2 3 3 3 4
 [ reached getOption("max.print") -- omitted 2911 entries ]

Within cluster sum of squares by cluster:
[1] 5200.755 6626.298 9333.447 6952.599
 (between_SS / total_SS =  34.6 %)
```

Comparing with the 12th column of the data

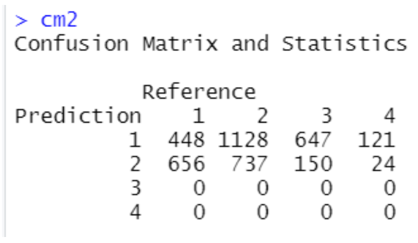


```
#getting the 12 column of the data set

qualityColumn <-factor(newdata$quality)
wineQuality <-as.numeric(qualityColumn)
```

Finding accuracy with Different K values

Accuracy = TP + TN / TP + TN + FP + FN

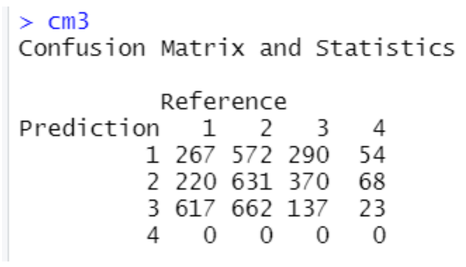
K=2



```
> cm2
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4
         1  448 1128  647  121
         2  656  737  150   24
         3    0    0    0    0
         4    0    0    0    0
```

Accuracy = (448 +737)/3911 * 100 = **30.3%**

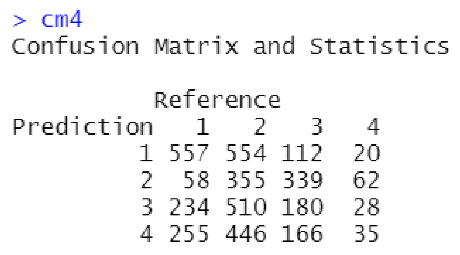
K=3



```
> cm3
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1 267 572 290  54
         2 220 631 370  68
         3 617 662 137  23
         4   0   0   0   0
```

Accuracy = (267+631+137)/3911 *100 = **26.4%**

K=4



```
> cm4
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1 557 554 112  20
         2  58 355 339  62
         3 234 510 180  28
         4 255 446 166  35
```

Accuracy = (557+355+180+35) * 100 = **28.8%**

According to the comparison **K=2** is the winner because k=2 has a greater accuracy than all other clusters, the highest accuracy model is the 2-cluster model which has an accuracy of 30.3%

Explaining Accuracy, Recall, Precision

Accuracy - Accuracy is the number of right guesses your model produced throughout the whole test dataset.

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

Precision - Precision is a statistic used to assess the accuracy of a positive forecast.

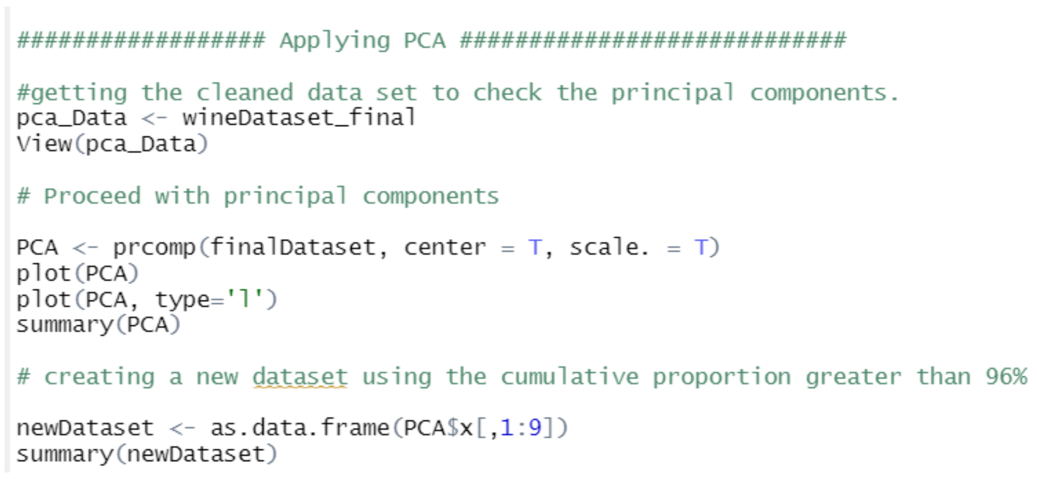
$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

Recall - The detection accuracy, also known as recall, is a statistic that defines how many true positives should be anticipated from any positives in a dataset**.**

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

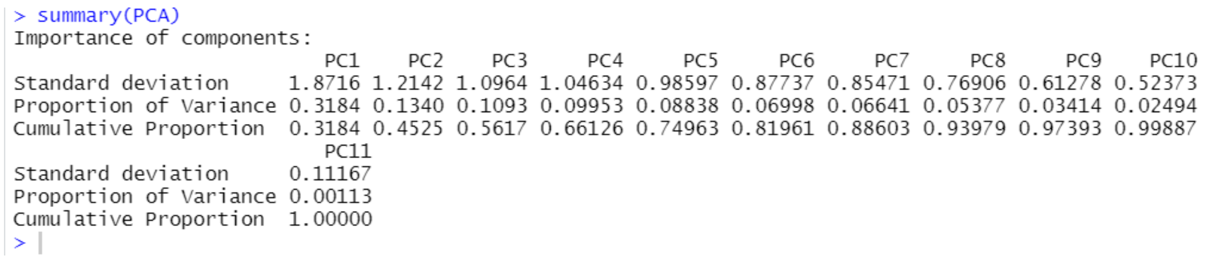
Applying Principal Component Analysis for the Dataset

In a nutshell, PCA is a dimensional reduction approach that reduces a collection of features in a dataset into a fewer number of features known as principle components while retaining as much information as possible in the original dataset:

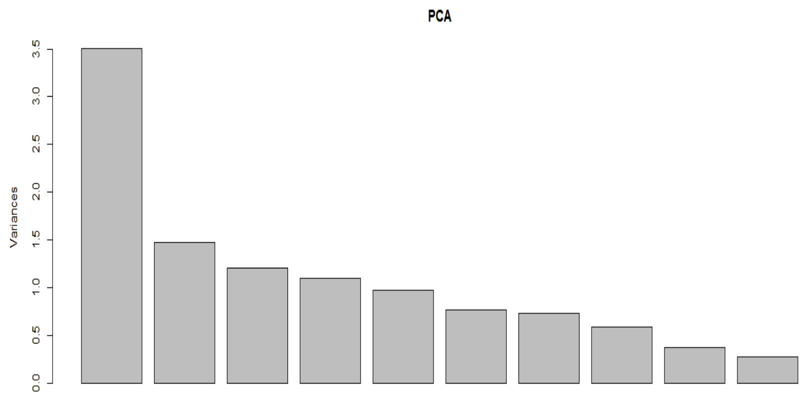


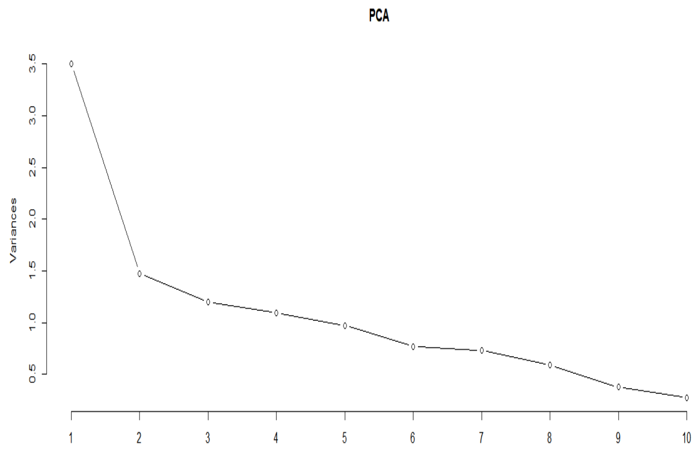
```
################# Applying PCA ###########################

#getting the cleaned data set to check the principal components.
pca_Data <- wineDataset_final
View(pca_Data)

# Proceed with principal components

PCA <- prcomp(finalDataset, center = T, scale. = T)
plot(PCA)
plot(PCA, type='l')
summary(PCA)

# creating a new dataset using the cumulative proportion greater than 96%

newDataset <- as.data.frame(PCA$x[,1:9])
summary(newDataset)
```

Summary of PCA



```
> summary(PCA)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10
Standard deviation     1.8716 1.2142 1.0964 1.04634 0.98597 0.87737 0.85471 0.76906 0.61278 0.52373
Proportion of Variance 0.3184 0.1340 0.1093 0.09953 0.08838 0.06998 0.06641 0.05377 0.03414 0.02494
Cumulative Proportion  0.3184 0.4525 0.5617 0.66126 0.74963 0.81961 0.88603 0.93979 0.97393 0.99887
                          PC11
Standard deviation     0.11167
Proportion of Variance 0.00113
Cumulative Proportion  1.00000
> 
```





<u>Applying K-means to the New PCA Dataset</u>

Apply winning Clusters model K=2

```
## Apply K-Means to new PCA Data ###

# When k=2(Winning Clusters method) ###

kmnew <- kmeans(newDataset, centers =2, nstart =25)
kmnew

kmnew.clusters <- kmnew$cluster

fviz_cluster(kmnew, geom = "point", data = newDataset)+ ggtitle("k = 2")
```

Finding all the BSS, TSS, WSS, BSS/TSS values for the PCA Data

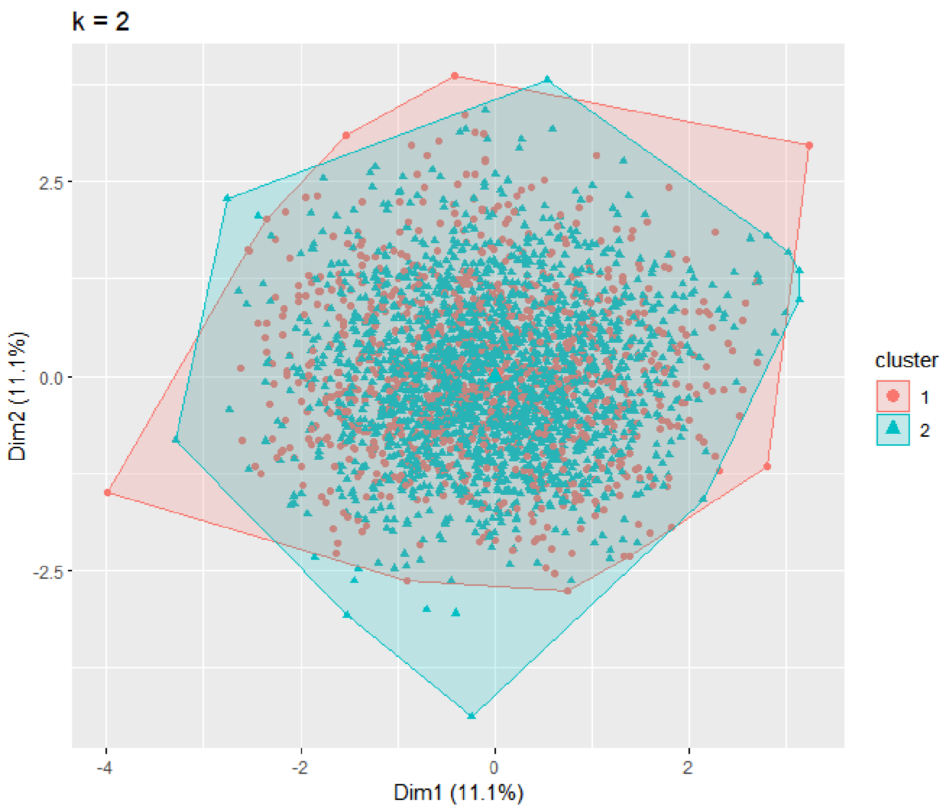
```
#BSS value
kmnew$betweenss
#TSS Value
kmnew$totss
#WSS Value
kmnew$withinss
#Value of BSS/TSS
kmnew$betweenss/kmnew$totss
```

```
> #BSS value
> kmnew$betweenss
[1] 9968.575
> #TSS Value
> kmnew$totss
[1] 41888.75
> #WSS Value
> kmnew$withinss
[1] 19567.65 12352.53
> #Value of BSS/TSS
> kmnew$betweenss/kmnew$totss
[1] 0.2379774
```

New PCA Data Clustering centers

```
> kmnew$centers
        PC1         PC2         PC3         PC4          PC5           PC6         PC7         PC8
1  1.301200  0.07560363 -0.04888849 -0.02304394 -0.005248234  0.0007483407  0.04117806 -0.01404609
2 -1.946402 -0.11309184  0.07312994  0.03447033  0.007850581 -0.0011194069 -0.06159629  0.02101088
          PC9
1 -0.01644589
2  0.02460062
```

Cluster Plot for PCA applied Clustering



Discuss the performance for this "PCA-based" dataset

PCA simplifies high-dimensional data while keeping trends and patterns. This is achieved by compressing the data into fewer dimensions that act as feature summaries. As demonstrated by the preceding instances, we employed k-mean clustering with 11 attributes in the km2 and PCA to decrease dimensionality to 9 attributes. However, the results of both procedures were almost similar

The sum of squares inside each cluster represents the observed variability. In general, a cluster with a small sum of squares is more compact than one with a big total of squares. When the WSS values of the two models are compared, we can see that the data in kmnew, which was done using PCA, has more compact data in its clusters than the data in Km2 since the WSS values of kmnew are less than the WSS values of km2. Consequently, there have been minor changes in the ratio of BSS/TSS and BSS values favorable to the k-mean model utilizing the PC values dataset.

Comparison Table

	km2	**kmnew**
BSS	9968.607	9968.575
Ratio BSS/TSS	23.2%	23.8%
WSS	20187.05 12854.34	19567.65 12352.53

2nd Objective (MLP)

Various methods used for defining the input vector in electricity load forecasting problems

Although numerous forecasting methods and systems for reliable load forecasting have been created, selecting an acceptable forecasting model for a given electrical network is challenging, and none of them can be extended to all demand patterns. Electric load forecasting methods can be classified into two main types:

1. Time series forecasting methods - This will be determined mostly by the historical series.

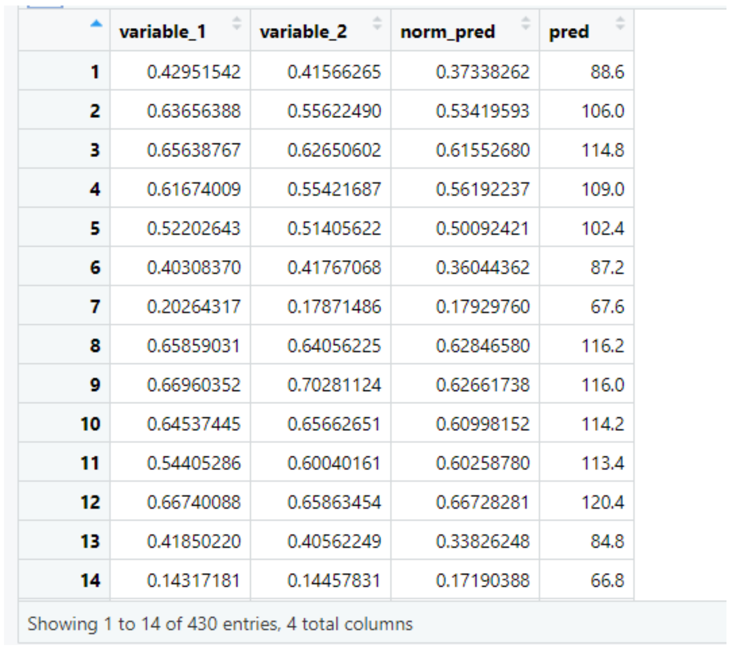
2. Multi factor forecasting methods - focuses on the relationship between different influencing elements and predicting values.

The multi-factor/cross-sectional forecasting technique focuses on the search for causal links between various influencing factors and forecasted values. In contrast, the time series forecasting approach is more dependent on previous data. As a result, many academics are using time series forecasting to anticipate electric demand to avoid the complex and non-objective aspects that might impair the accuracy of a multi-factor forecasting model. As a result, predicting time series is easier and faster. Statistical models, machine learning models, and hybrid models are the three most prevalent and commonly utilized time series forecasting models (Hammad et al., 2020).

<u>Evidence of various adopted input vectors and the related input/output matrices for both AR and NARX based approaches</u>

<u>The AR Based Approach</u>

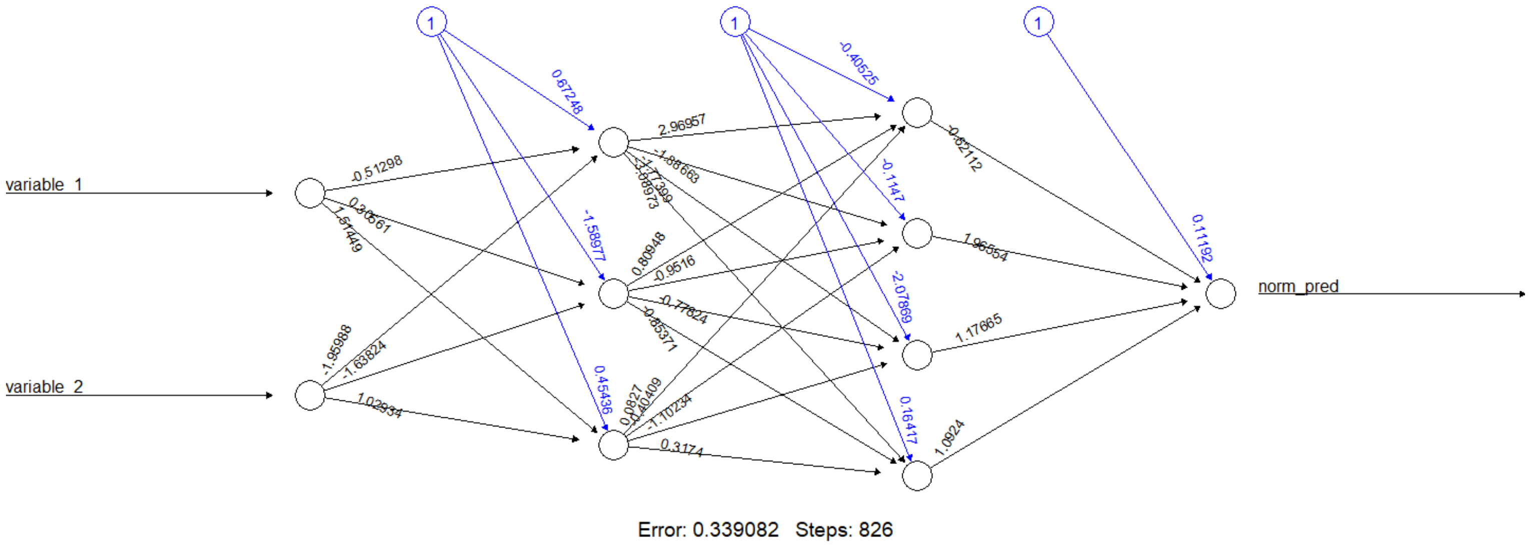
The below table will display all the input vectors variable_1 and variable_2 will be the input figures, and the norm_pred will be the output figure.



	variable_1	variable_2	norm_pred	pred
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

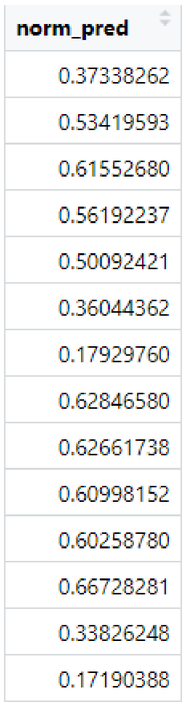
Showing 1 to 14 of 430 entries, 4 total columns

The chart below shows the model one that was trained using the AR method. It clearly visualizes the variable_1 and variable_2 which are the input figures, while the norm_pred is the relevant output.



NARX Approach

The input vector used to create the NARX-based model is shown below, with the normalized value norm_pred as the input variables.



norm_pred
0.37338262
0.53419593
0.61552680
0.56192237
0.50092421
0.36044362
0.17929760
0.62846580
0.62661738
0.60998152
0.60258780
0.66728281
0.33826248
0.17190388

Evidence of correct normalization and brief discussion of its necessity

Normalization

Normalization is a data preparation process that changes the values of numerical columns in a dataset to use a similar scale. This is especially important when the characteristics used by your Machine Learning model have varying ranges. Such a circumstance occurs frequently in the actual world, when one property is fractional and ranges from zero and one, whereas other ranges from zero and a thousand. If our aim is to forecast using regression, this characteristic will have a greater effect on the outcome due to its higher value, while not always being the more essential predictor.

A Min-Max Normalization was used in this project

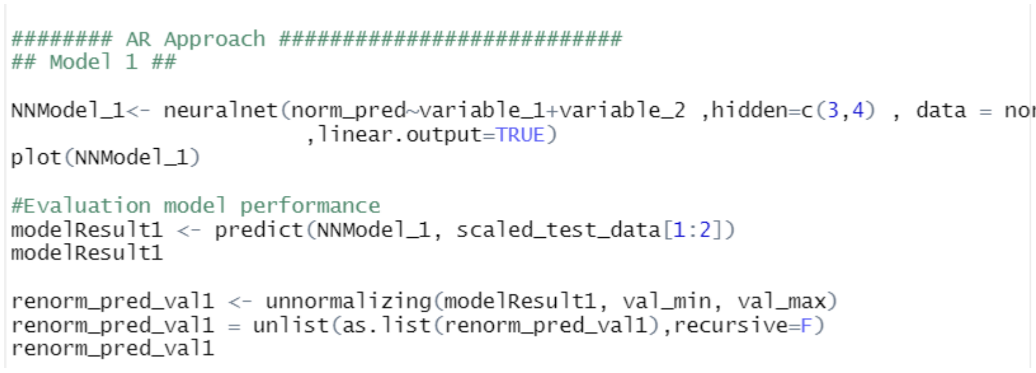
```
# Normalizing Data
normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))}

# scaling using the normalization function

normalized_data <- as.data.frame(lapply(UoW_load[,2:4], normalize))
normalized_data <- cbind(normalized_data, UoW_load[c(4)])
```

Why Normalization is Important?

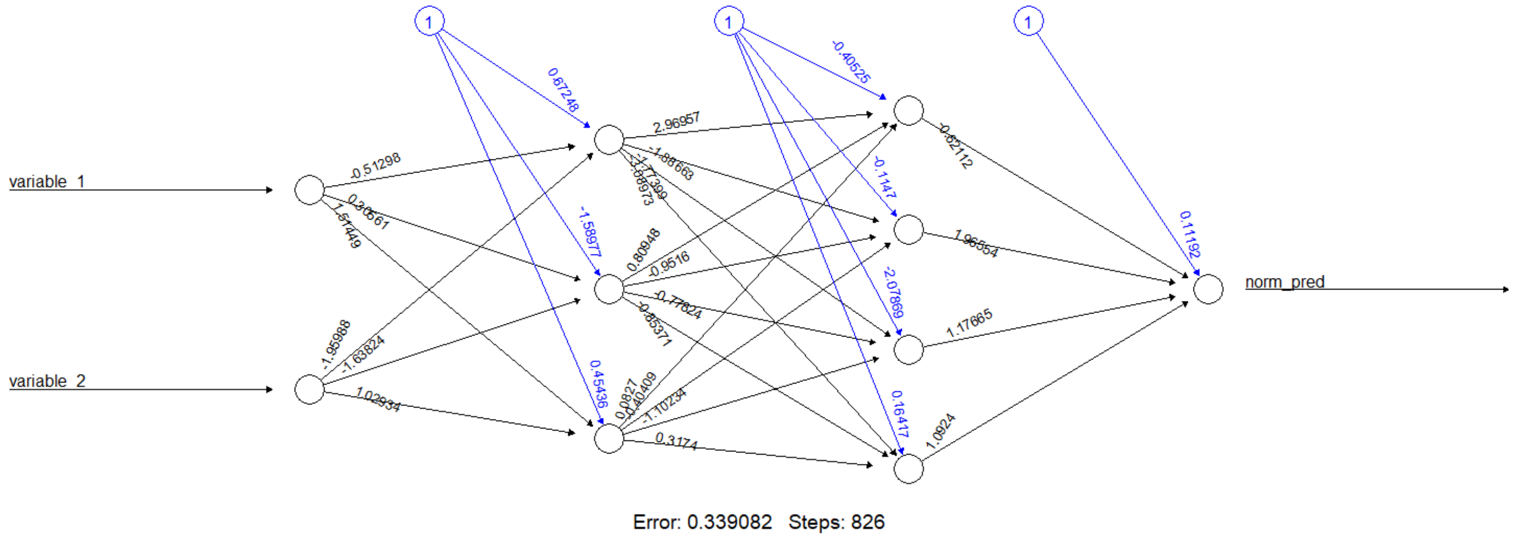
Normalization is necessary to guarantee that the table contains only data that is directly connected to the primary key, that each data field has only one data element, and that redundant data is deleted. Normalizing data to a mean around 0 is one of the recommended strategies for neural network training. Data normalization, in general, speeds up learning and leads to faster convergence. Because the tanh function (among others) looks to be significantly increased, the (logistic) sigmoid function is never employed as an activation function in the neural network's buried layers. While this may not appear to be the case at first glance, there are a variety of reasons behind this in appearance, the tan function resembles the 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.

Implement several MLPs for the AR approach and NARX, using various structures

AR Approach - Model 1
Below there are 3 models with different number of hidden layers with different number of nodes.



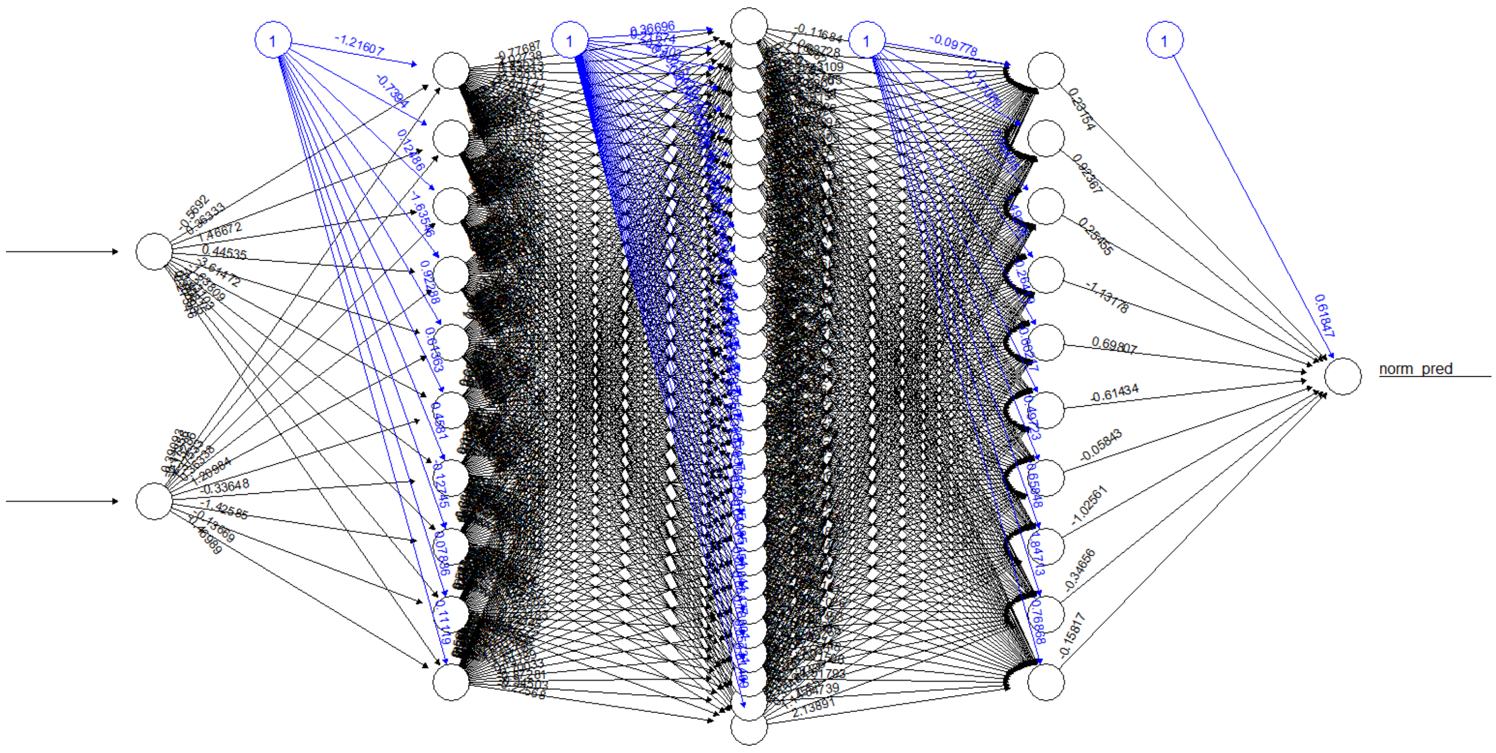
```
######## AR Approach ###########################
## Model 1 ##

NNModel_1<- neuralnet(norm_pred~variable_1+variable_2 ,hidden=c(3,4) , data = norm_traindata
                       ,linear.output=TRUE)
plot(NNModel_1)

#Evaluation model performance
modelResult1 <- predict(NNModel_1, scaled_test_data[1:2])
modelResult1

renorm_pred_val1 <- unnormalizing(modelResult1, val_min, val_max)
renorm_pred_val1 = unlist(as.list(renorm_pred_val1),recursive=F)
renorm_pred_val1
```



AR Approach - Model 2

```
## Model 2 ##

NNModel_2<- neuralnet(norm_pred~variable_1+variable_2 ,hidden=c(10,30,10) , data = norm_traindata
                      ,linear.output=TRUE)
plot(NNModel_2)

#Evaluation model performance
modelResult2 <- predict(NNModel_2, norm_testdata[1:2])
modelResult2

renorm_pred_val2 <- unnormalizing(modelResult2, val_min, val_max)
renorm_pred_val2 = unlist(as.list(renorm_pred_val2),recursive=F)
renorm_pred_val2
```

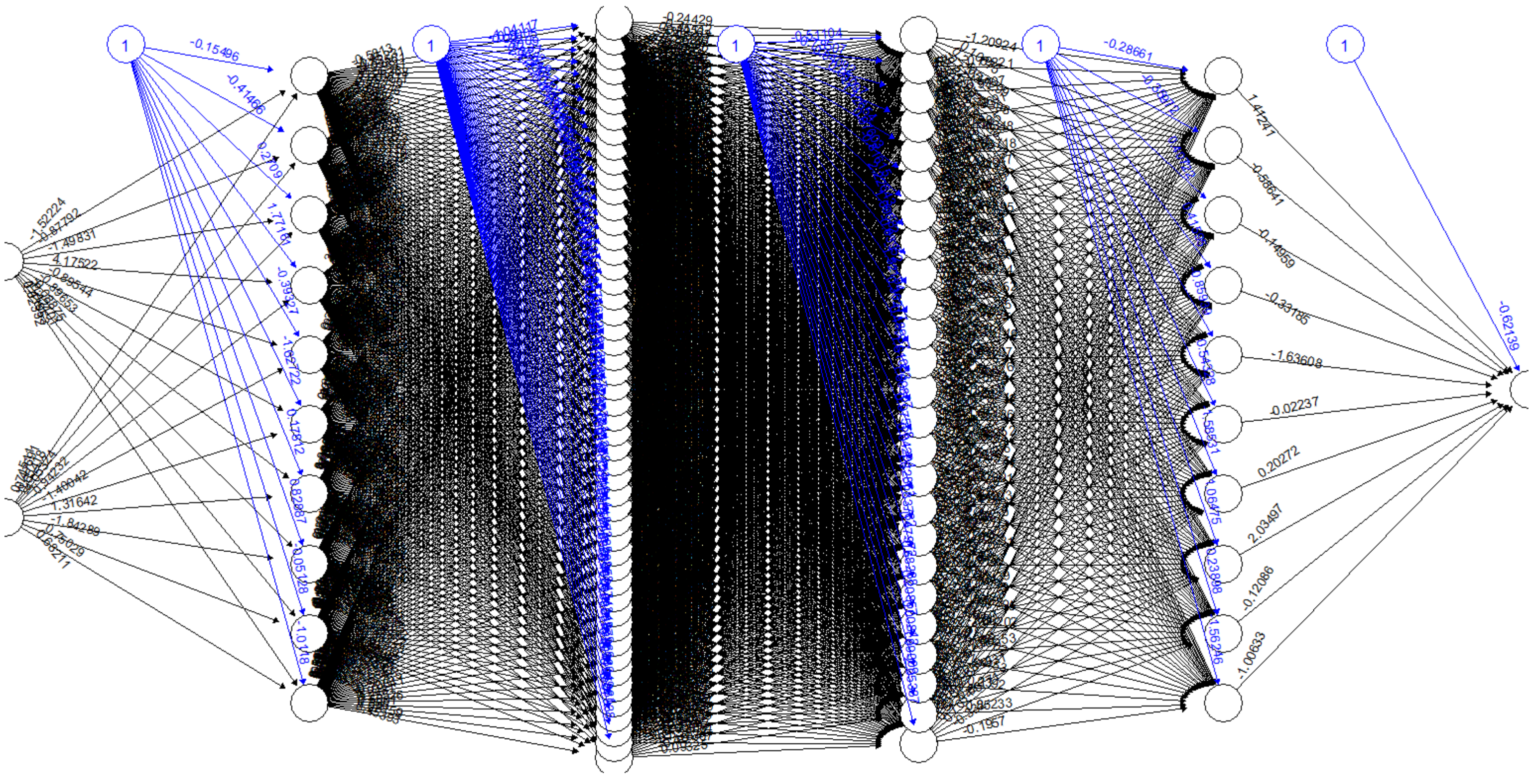
Output:



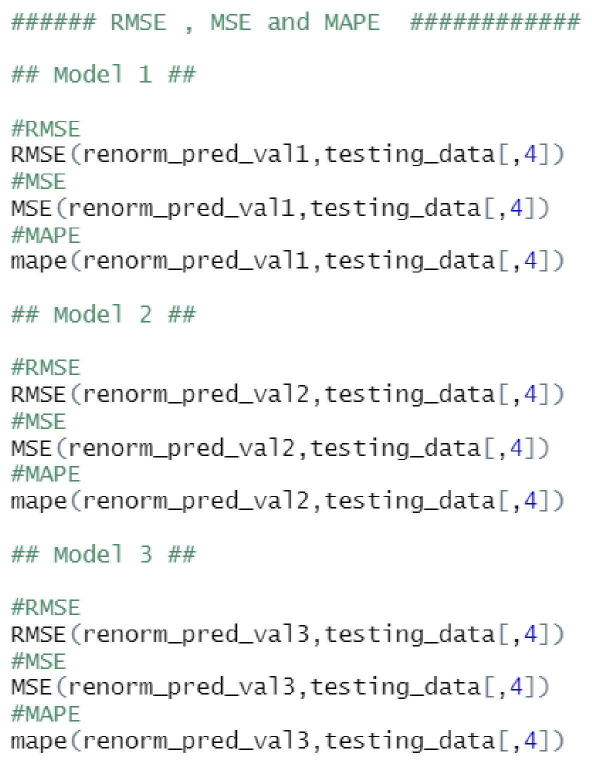
AR Approach - Model 3

```
## Model 3 ##

NNModel_3<- neuralnet(norm_pred~variable_1+variable_2 ,hidden=c(10,50,25,10) , data = norm_traindata
                      ,linear.output=TRUE)
plot(NNModel_3)

#Evaluation model performance
modelResult3 <- predict(NNModel_3, norm_testdata[1:2])
modelResult3

renorm_pred_val3 <- unnormalizing(modelResult3, val_min, val_max)
renorm_pred_val3 = unlist(as.list(renorm_pred_val3),recursive=F)
renorm_pred_val3
```

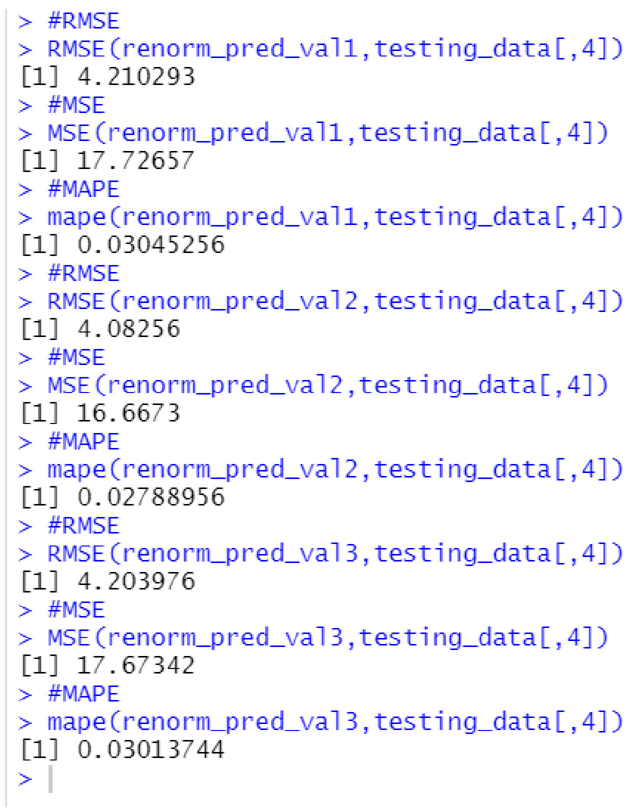
Output:



<u>Evidence of RMSE MAE and MAPE values and comparison of these values in all three models</u>



```
###### RMSE , MSE and MAPE  ############

## Model 1 ##

#RMSE
RMSE(renorm_pred_val1,testing_data[,4])
#MSE
MSE(renorm_pred_val1,testing_data[,4])
#MAPE
mape(renorm_pred_val1,testing_data[,4])

## Model 2 ##

#RMSE
RMSE(renorm_pred_val2,testing_data[,4])
#MSE
MSE(renorm_pred_val2,testing_data[,4])
#MAPE
mape(renorm_pred_val2,testing_data[,4])

## Model 3 ##

#RMSE
RMSE(renorm_pred_val3,testing_data[,4])
#MSE
MSE(renorm_pred_val3,testing_data[,4])
#MAPE
mape(renorm_pred_val3,testing_data[,4])
```



```
> #RMSE
> RMSE(renorm_pred_val1,testing_data[,4])
[1] 4.210293
> #MSE
> MSE(renorm_pred_val1,testing_data[,4])
[1] 17.72657
> #MAPE
> mape(renorm_pred_val1,testing_data[,4])
[1] 0.03045256
> #RMSE
> RMSE(renorm_pred_val2,testing_data[,4])
[1] 4.08256
> #MSE
> MSE(renorm_pred_val2,testing_data[,4])
[1] 16.6673
> #MAPE
> mape(renorm_pred_val2,testing_data[,4])
[1] 0.02788956
> #RMSE
> RMSE(renorm_pred_val3,testing_data[,4])
[1] 4.203976
> #MSE
> MSE(renorm_pred_val3,testing_data[,4])
[1] 17.67342
> #MAPE
> mape(renorm_pred_val3,testing_data[,4])
[1] 0.03013744
>
```

Comparison Table

Model	**RMSE**	**MSE**	**MAPE**
Model 1	4.210293	17.72657	0.03045256
Model 2	4.08256	16.6673	0.02788956
Model 3	4.203976	17.67372	0.03013744

By varying the inputs, we created 3 models using NARX approach which is given below:

Model 1

```
########### NARX Approach  ###########

## Model 1 ##

narx.model1<- nnetTs(norm_traindata[c(3)],m=5, size=3,steps=30)
narx.model1
renorm_pred_val4 <- unnormalizing(predict(narx.model1,steps=5,n.ahead=20), min, max)
renorm_pred_val4 = unlist(as.list(renorm_pred_val4),recursive=F)
renorm_pred_val4

plot.ts(renorm_pred_val4)
plot.ts(normalized_data[c(2)])
```

```
> narx.model1

Non linear autoregressive model

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

Model 2

```
## Model 2 ##

narx.model2<- nnetTs(norm_traindata[c(3)], m = 4, size=3,steps=20)
narx.model2
renorm_pred_val5 <- unnormalizing(predict(narx.model2,steps=5,n.ahead=20), min,  max)

renorm_pred_val5 = unlist(as.list(renorm_pred_val5),recursive=F)
renorm_pred_val5
plot.ts(renorm_pred_val5)
```

```
> narx.model2

Non linear autoregressive model

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

Model 3

```
## Model 3 ##

narx.model3<- nnetTs(norm_traindata[c(3)], m = 5, size=8,steps=10)
narx.model3
renorm_pred_val6 <- unnormalizing(predict(narx.model3,steps=5,n.ahead=20), min, max)

renorm_pred_val6 = unlist(as.list(renorm_pred_val6),recursive=F)
renorm_pred_val6
plot.ts(renorm_pred_val6)
```

```
> narx.model3

Non linear autoregressive model

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

Comparison Table

NARX MODEL	Output with network and weight
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

Discussion of the meaning of these stat. indices

Statistical Indices

Statistical indices the distribution of elements in a set of elements is described by these values. These indices can be used to investigate the relationships between subsets of elements, as well as to confirm or refute some of the rules and correlations that govern the behavior of those subsets.

- **Root mean square error (RMSE)**
  The Root Mean Square Error represents the residuals' standard deviation (i.e., the difference between the model predictions and the real values (training data)) is represented by RMSE
  The RMSE provides an approximate measure of the dispersion of the residuals.

- **Mean Absolute error (MAE)**
  The MAE is calculated by taking the absolute difference between the expected and actual outcomes. The MAE is a measure of the averaged degree of error in the regression model. If MAE is equal to zero, the model's predictions are flawless.

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

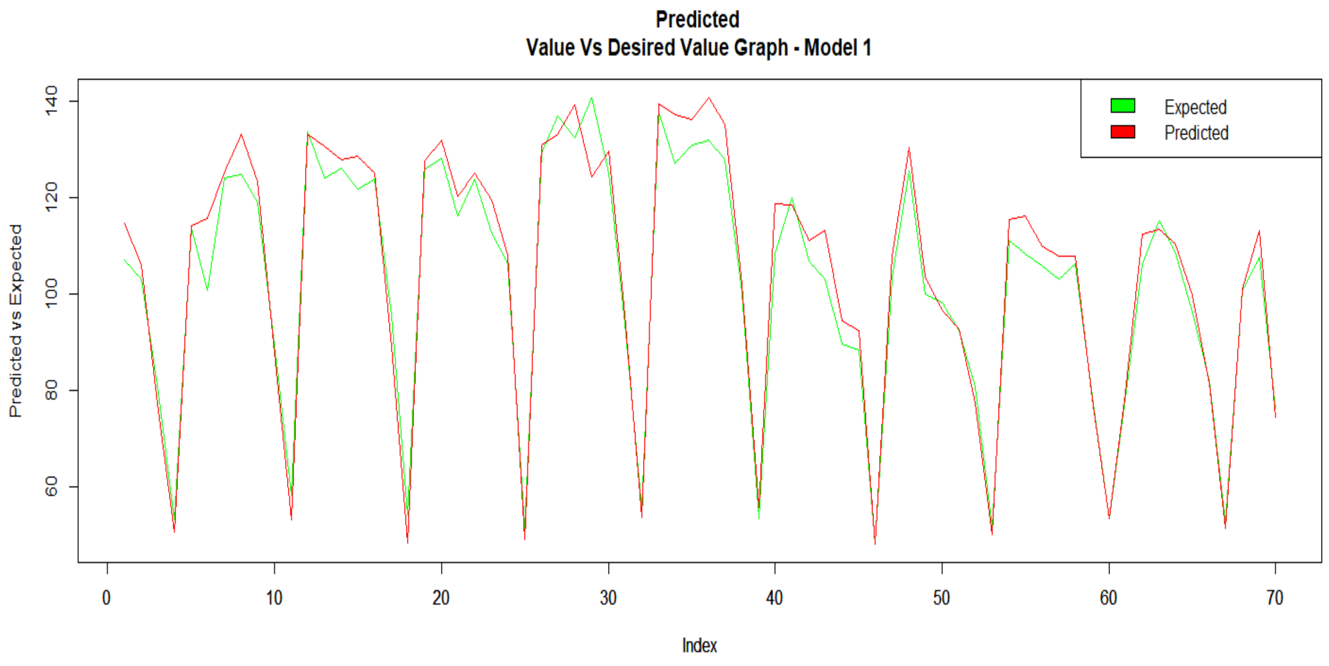
Discuss the issue of “efficiency” with your two best NN structures

The multi-factor/cross-sectional forecasting technique focuses on the search for causal links between various influencing factors and forecasted values. In contrast, the time series forecasting approach is more dependent on previous data. As a result, predicting time series is easier and faster.

Best Results Graph (Predicted output vs Desired Output)

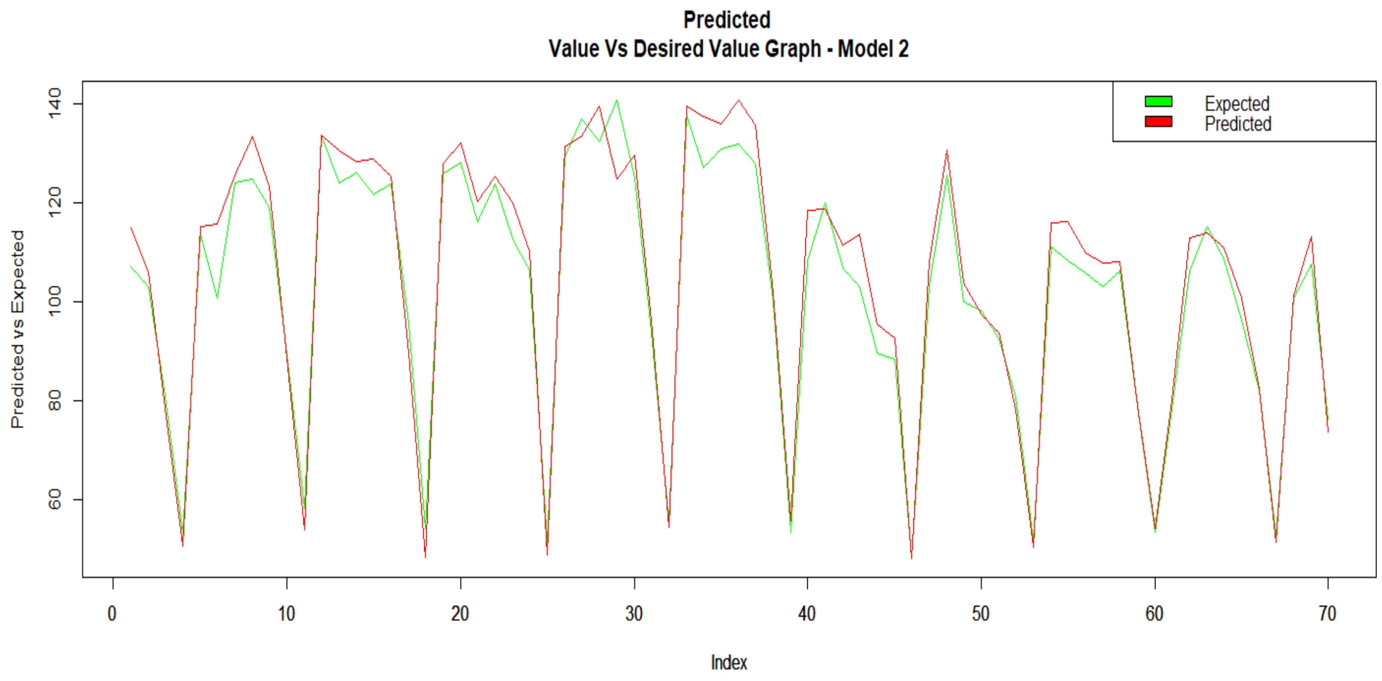
Model 1

```
# Plotting Prediction Graph #

plot(testing_data[,4] , ylab = "Predicted vs Expected", type="l", col="green" )
par(new=TRUE)
plot(renorm_pred_val1, ylab = " ", yaxt="n", type="l", col="red" ,main='Predicted
Value Vs Desired Value Graph - Model 1')
legend("topright",
       c("Expected","Predicted"),
       fill=c("green","red")
)
```



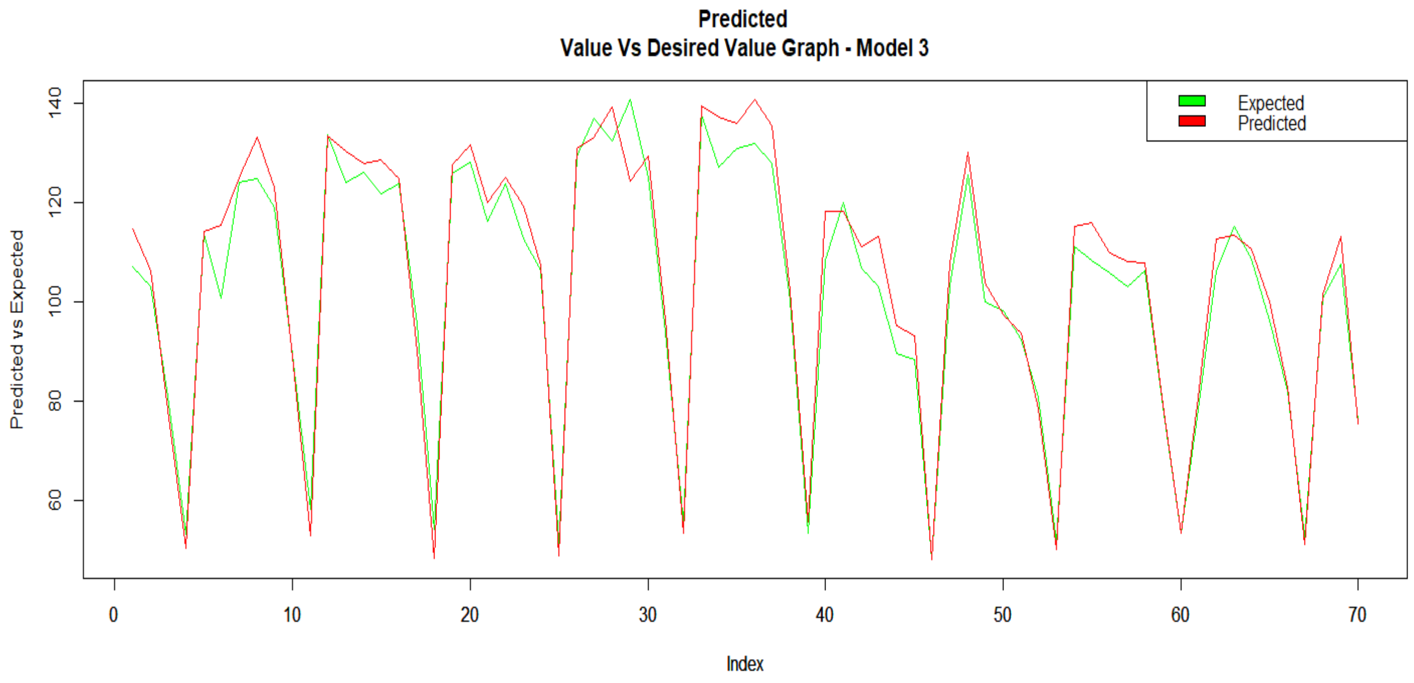
Model 2

```
plot(testing_data[,4] , ylab = "Predicted vs Expected", type="l", col="green" )
par(new=TRUE)
plot(renorm_pred_val2, ylab = " ", yaxt="n", type="l", col="red" ,main='Predicted
Value Vs Desired Value Graph - Model 2')
legend("topright",
       c("Expected","Predicted"),
       fill=c("green","red")
)
```



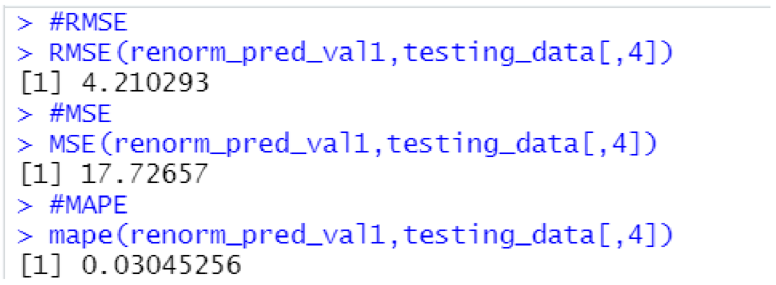
Model 3

```
plot(testing_data[,4] , ylab = "Predicted vs Expected", type="l", col="green" )
par(new=TRUE)
plot(renorm_pred_val3, ylab = " ", yaxt="n", type="l", col="red" ,main='Predicted
Value Vs Desired Value Graph - Model 3')
legend("topright",
       c("Expected","Predicted"),
       fill=c("green","red")
)
```



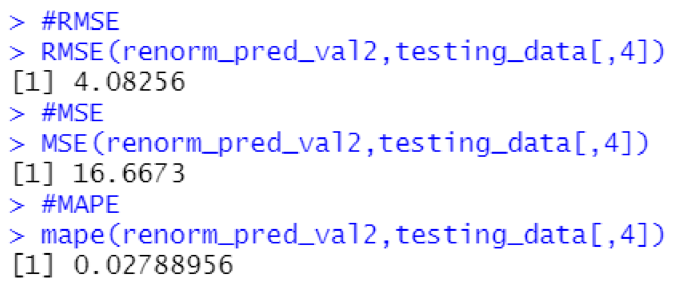
<u>Requested Statistical Indices</u>

Model 1



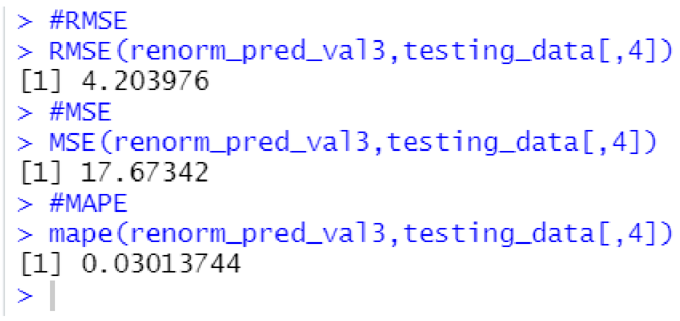
```
> #RMSE
> RMSE(renorm_pred_val1,testing_data[,4])
[1] 4.210293
> #MSE
> MSE(renorm_pred_val1,testing_data[,4])
[1] 17.72657
> #MAPE
> mape(renorm_pred_val1,testing_data[,4])
[1] 0.03045256
```

<u>Model 2</u>



```
> #RMSE
> RMSE(renorm_pred_val2,testing_data[,4])
[1] 4.08256
> #MSE
> MSE(renorm_pred_val2,testing_data[,4])
[1] 16.6673
> #MAPE
> mape(renorm_pred_val2,testing_data[,4])
[1] 0.02788956
```

<u>Model 3</u>



```
> #RMSE
> RMSE(renorm_pred_val3,testing_data[,4])
[1] 4.203976
> #MSE
> MSE(renorm_pred_val3,testing_data[,4])
[1] 17.67342
> #MAPE
> mape(renorm_pred_val3,testing_data[,4])
[1] 0.03013744
> 
```

Appendix

Objective 1

```
library(caret)

library(tidyverse)

library(leaps)

library(ggplot2)

library(lattice)

library(reshape2)

library(MASS)

library(ggcorrplot)

library(corrplot)

library(plotmo)

library(keras)

library(kableExtra)

library(modelr)

library(psych)

library(Rmisc)

library(plyr)

library(dplyr)

library(gridExtra)

library(scales)

library(rpart)

library(yardstick)

library(cluster)

library(NbClust)

library(factoextra)

library(dplyr)


library(readxl)

Whitewine_v2 <- read_excel("C:/Users/Ajeevan Sivanandhan/Desktop/DM & ML/Final CW/Whitewine_v2.xlsx")
```

```
View(Whitewine_v2)


boxplot(Whitewine_v2) ### boxplot before removing outliers

summary(Whitewine_v2)


##### Removing Outliers ####

outliers = c()

for ( i in 1:11 ) {

  stats = boxplot.stats(Whitewine_v2[[i]])$stats

  bottom_outlier_rows = which(Whitewine_v2[[i]] < stats[1])

  top_outlier_rows = which(Whitewine_v2[[i]] > stats[5])

  outliers = c(outliers , top_outlier_rows[ !top_outlier_rows %in% outliers ] )

  outliers = c(outliers , bottom_outlier_rows[ !bottom_outlier_rows %in% outliers ] )

}


newdata = Whitewine_v2[-outliers, ] ## New data set after removing outliers.


boxplot(newdata) ### boxplot after removing outliers


#removing the 12th column of the data set because it is the output class.

finalDataset <- newdata[,-12]

#normalizing the data(Scaling data).

wineDataset_final <- scale(finalDataset)

#displaying the final scaled and cleaned dataset.

view(wineDataset_final)

#boxploting the final data set.

boxplot(wineDataset_final)

#checking the dimensions of the dataset.

dim(wineDataset_final)
```

```
## Defining Cluster centers
### NBClust method #####

numClusters <- NbClust(wineDataset_final, distance = "euclidean", min.nc = 2, max.nc = 10,
            method = "kmeans", index = "all", alphaBeale = 0.1)
numClusters

########################### Elbow Method ########

#finding the number of clusters(k) automatically by the elbow method.
fviz_nbclust(wineDataset_final, kmeans, method = "wss")+
  geom_vline(xintercept = 4, linetype = 2)+
  labs(subtitle = "Elbow Method")



########################### Silhoutte Method ########

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

########### Finding Number of Clusters Manually#####

# Initialize total within sum of squares error: wss
wss <- 0

for (i in 1:10) {
  km.out <- kmeans(wineDataset_final, centers = i)
  # Save total within sum of squares to wss variable
  wss[i] <- km.out$tot.withinss
}
wss
```

```
# Plot total within sum of squares vs. number of clusters

plot(1:10, wss, type = "b",
    xlab = "Number of Clusters",
    ylab = "Within groups sum of squares")

######## K-Means Clustering ##########

# When k=2

km2 <- kmeans(wineDataset_final, centers =2, nstart =25)
km2

km2.clusters <- km2$cluster

fviz_cluster(km2, geom = "point", data = wineDataset_final)+ ggtitle("k = 2")

#BSS value
km2$betweenss
#TSS Value
km2$totss
#WSS Value
km2$withinss
#Value of BSS/TSS
km2$betweenss/km2$totss



# When k=3

km3 <- kmeans(wineDataset_final, centers =3, nstart =25)
km3
```

```
km3.clusters <- km3$cluster

fviz_cluster(km3, geom = "point", data = wineDataset_final)+ ggtitle("k = 3")

#BSS value
km3$betweenss
#TSS Value
km3$totss
#WSS Value
km3$withinss
#Value of BSS/TSS
km3$betweenss/km3$totss



# When k=4

km4 <- kmeans(wineDataset_final, centers =4, nstart =25)
km4

km4.clusters <- km4$cluster

fviz_cluster(km4, geom = "point", data = wineDataset_final)+ ggtitle("k = 4")

#BSS value
km4$betweenss
#TSS Value
km4$totss
#WSS Value
km4$withinss
#Value of BSS/TSS
```

```
km4$betweenss/km4$totss


#getting the 12 column of the data set


qualityColumn <-factor(newdata$quality)
wineQuality <-as.numeric(qualityColumn)


cm2 <- confusionMatrix(as.factor(km2$cluster), as.factor(wineQuality))
cm2


cm3 <- confusionMatrix(as.factor(km3$cluster), as.factor(wineQuality))
cm3


cm4 <- confusionMatrix(as.factor(km4$cluster), as.factor(wineQuality))
cm4


################## Applying PCA ############################


#getting the cleaned data set to check the principal components.
pca_Data <- wineDataset_final
View(pca_Data)


# Proceeding with principal components


PCA <- prcomp(finalDataset, center = T, scale. = T)
plot(PCA)
plot(PCA, type='l')
summary(PCA)


# creating a new dataset using the cumulative proportion greater than 96%
```

```
newDataset <- as.data.frame(PCA$x[,1:9])

summary(newDataset)


## Apply K-Means to new PCA Data ###


# When k=2(Winning Clusters method) ###


kmnew <- kmeans(newDataset, centers =2, nstart =25)

kmnew


kmnew.clusters <- kmnew$cluster


fviz_cluster(kmnew, geom = "point", data = newDataset)+ ggtitle("k = 2")


#BSS value

kmnew$betweenss

#TSS Value

kmnew$totss

#WSS Value

kmnew$withinss

#Value of BSS/TSS

kmnew$betweenss/kmnew$totss

#centers

kmnew$centers
```

Objective 2

```
install.packages("Metrics")
install.packages("MLmetrics")
install.packages("neuralnet")
install.packages("tsDyn")
install.packages("tidypredict")

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

# Reading the Csv file #

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

# Extracting data for testing and training
training_data = head(UoW_load, n =430)
testing_data = tail(UoW_load, n =70)

# Normalizing Data
normalize <- function(x) {
  (x - min(x)) / (max(x) - min(x))}

# scaling using the normalization function

normalized_data <- as.data.frame(lapply(UoW_load[,2:4], normalize))
normalized_data <- cbind(normalized_data, UoW_load[c(4)])
```

```
# Changing column names

names(normalized_data)[1] <- "variable_1"

names(normalized_data)[2] <- "variable_2"

names(normalized_data)[3] <- "norm_pred"

names(normalized_data)[4] <- "pred"


# taking scaled values for testing and training

norm_testdata = tail(normalized_data, n =70)

norm_traindata = head(normalized_data, n =430)


# un normalize function

unnormalizing <- function(x, min, max) {

  return( (max - min)*x + min )

}

View(norm_traindata)



# Taking the maximum and minimum value

min <- min(normalized_data[4])

max <- max(normalized_data[4])


######## AR Approach ###########################


## Model 1 ##


NNModel_1<- neuralnet(norm_pred~variable_1+variable_2 ,hidden=c(3,4) , data = norm_traindata

                     ,linear.output=TRUE)
```

```
plot(NNModel_1)


#Evaluation model performance

modelResult1 <- predict(NNModel_1, norm_testdata[1:2])

modelResult1


renorm_pred_val1 <- unnormalizing(modelResult1, min, max)

renorm_pred_val1 = unlist(as.list(renorm_pred_val1),recursive=F)

renorm_pred_val1


## Model 2 ##


NNModel_2<- neuralnet(norm_pred~variable_1+variable_2 ,hidden=c(10,30,10) , data =
norm_traindata

              ,linear.output=TRUE)

plot(NNModel_2)


#Evaluation model performance

modelResult2 <- predict(NNModel_2, norm_testdata[1:2])

modelResult2


renorm_pred_val2 <- unnormalizing(modelResult2, min, max)

renorm_pred_val2 = unlist(as.list(renorm_pred_val2),recursive=F)

renorm_pred_val2



## Model 3 ##
```

```
NNModel_3<- neuralnet(norm_pred~variable_1+variable_2 ,hidden=c(10,50,25,10) , data =
norm_traindata

                    ,linear.output=TRUE)

plot(NNModel_3)


#Evaluation model performance

modelResult3 <- predict(NNModel_3, norm_testdata[1:2])

modelResult3


renorm_pred_val3 <- unnormalizing(modelResult3, min, max)

renorm_pred_val3 = unlist(as.list(renorm_pred_val3),recursive=F)

renorm_pred_val3


########### NARX Approach  ###########


## Model 1 ##


narx.model1<- nnetTs(norm_traindata[c(3)],m=5, size=3,steps=30)

narx.model1

renorm_pred_val4 <- unnormalizing(predict(narx.model1,steps=5,n.ahead=20), min, max)

renorm_pred_val4 = unlist(as.list(renorm_pred_val4),recursive=F)

renorm_pred_val4


plot.ts(renorm_pred_val4)

plot.ts(normalized_data[c(2)])


## Model 2 ##


narx.model2<- nnetTs(norm_traindata[c(3)], m = 4, size=3,steps=20)
narx.model2

renorm_pred_val5 <- unnormalizing(predict(narx.model2,steps=5,n.ahead=20), min,  max)


renorm_pred_val5 = unlist(as.list(renorm_pred_val5),recursive=F)

renorm_pred_val5

plot.ts(renorm_pred_val5)


## Model 3 ##


narx.model3<- nnetTs(norm_traindata[c(3)], m = 5, size=8,steps=10)

narx.model3

renorm_pred_val6 <- unnormalizing(predict(narx.model3,steps=5,n.ahead=20), min, max)


renorm_pred_val6 = unlist(as.list(renorm_pred_val6),recursive=F)

renorm_pred_val6

plot.ts(renorm_pred_val6)



###### RMSE , MSE and MAPE  ############


## Model 1 ##


#RMSE

RMSE(renorm_pred_val1,testing_data[,4])

#MSE

MSE(renorm_pred_val1,testing_data[,4])

#MAPE

mape(renorm_pred_val1,testing_data[,4])
```

```
## Model 2 ##


#RMSE

RMSE(renorm_pred_val2,testing_data[,4])

#MSE

MSE(renorm_pred_val2,testing_data[,4])

#MAPE

mape(renorm_pred_val2,testing_data[,4])


## Model 3 ##


#RMSE

RMSE(renorm_pred_val3,testing_data[,4])

#MSE

MSE(renorm_pred_val3,testing_data[,4])

#MAPE

mape(renorm_pred_val3,testing_data[,4])


### Plotting ###


## Model 1 ##

# regression line #


plot(x=testing_data[,4], y = renorm_pred_val1, col = "red",

    main = 'Real vs Predicted')

abline(0, 1, lwd = 2)

plot.ts(x = testing_data[,4] ,y = renorm_pred_val1)

abline(0, 1, lwd = 2,col = "red")
```

```
## Model 2 ##
# regression line #

plot(x=testing_data[,4], y = renorm_pred_val2, col = "red",
    main = 'Real vs Predicted')
abline(0, 1, lwd = 2)
plot.ts(x = testing_data[,4] ,y = renorm_pred_val2)
abline(0, 1, lwd = 2,col = "red")

## Model 3 ##
# regression line #

plot(x=testing_data[,4], y = renorm_pred_val3, col = "red",
    main = 'Real vs Predicted')
abline(0, 1, lwd = 2)
plot.ts(x = testing_data[,4] ,y = renorm_pred_val3)
abline(0, 1, lwd = 2,col = "red")

# Plotting Prediction Graph #

plot(testing_data[,4] , ylab = "Predicted vs Expected", type="l", col="green" )
par(new=TRUE)
plot(renorm_pred_val1, ylab = " ", yaxt="n", type="l", col="red" ,main='Predicted
Value Vs Desired Value Graph - Model 1')
legend("topright",
      c("Expected","Predicted"),
      fill=c("green","red")
)
```

```
plot(testing_data[,4] , ylab = "Predicted vs Expected", type="l", col="green" )
par(new=TRUE)
plot(renorm_pred_val2, ylab = " ", yaxt="n", type="l", col="red" ,main='Predicted
Value Vs Desired Value Graph - Model 2')
legend("topright",
      c("Expected","Predicted"),
      fill=c("green","red")
)

plot(testing_data[,4] , ylab = "Predicted vs Expected", type="l", col="green" )
par(new=TRUE)
plot(renorm_pred_val3, ylab = " ", yaxt="n", type="l", col="red" ,main='Predicted
Value Vs Desired Value Graph - Model 3')
legend("topright",
      c("Expected","Predicted"),
      fill=c("green","red")
)
```