Informatics Institute of Technology

University of Westminster

Machine Learning and data mining

5DATA002W

Student name: K W A Ishini S. Ranasinghe

UoW Number: W1953836

IIT Number: 20221153

Table of Contents

Table of Figures

1. Partitioning Clustering Part

Before diving into data processing and techniques in RStudio, laying the framework by importing and initializing specific libraries is critical. These libraries contain functions required for various tasks and can be called depending on the process needs. In RStudio, these libraries are imported and initialized using the library keyword. This process guarantees that the required tools and functions are available for the upcoming data processing tasks.

SubTask1

a) scaling and outliers detection

- Firstly, we should import the data set using “readxl” and “read_excel”

```
#install the packages
library(readxl)
#import the dataset
dataSet <- read_excel("Whitewine_v6.xlsx")
```

Figure 1:import dataset

```
> summary(data)
 fixed acidity     volatile acidity
 Min.   :-2.603   Min.   :-2.070
 1st Qu.:-0.652   1st Qu.:-0.677
 Median :-0.043   Median :-0.141
 Mean   :-0.038   Mean   :-0.003
 3rd Qu.: 0.567   3rd Qu.: 0.502
 Max.   : 2.518   Max.   : 6.183
 NA's   :29301    NA's   :29301
  citric acid      residual sugar
 Min.   :-2.897   Min.   :-1.135
 1st Qu.:-0.578   1st Qu.:-0.914
 Median :-0.149   Median :-0.251
 Mean   :-0.006   Mean   : 0.007
 3rd Qu.: 0.452   3rd Qu.: 0.679
 Max.   :11.358   Max.   : 3.469
 NA's   :29301    NA's   :29301
   chlorides        free sulfur dioxide
 Min.   :-1.792   Min.   :-2.137
 1st Qu.:-0.475   1st Qu.:-0.729
 Median :-0.134   Median :-0.088
 Mean   : 0.002   Mean   : 0.005
 3rd Qu.: 0.208   3rd Qu.: 0.680
 Max.   :10.205   Max.   : 5.930
 NA's   :29301    NA's   :29301
 total sulfur dioxide    density
 Min.   :-2.905       Min.   :-2.412
 1st Qu.:-0.711       1st Qu.:-0.805
 Median :-0.084       Median :-0.120
 Mean   : 0.000       Mean   :-0.004
 3rd Qu.: 0.663       3rd Qu.: 0.736
 Max.   : 3.894       Max.   : 2.830
 NA's   :29301        NA's   :29301
       pH            sulphates
 Min.   :-2.634   Min.   :-2.372
 1st Qu.:-0.643   1st Qu.:-0.735
 Median :-0.065   Median :-0.132
 Mean   : 0.014   Mean   : 0.001
 3rd Qu.: 0.641   3rd Qu.: 0.471
 Max.   : 3.980   Max.   : 5.038
 NA's   :29301    NA's   :29301
    alcohol          quality
 Min.   :-1.677   Min.   :-1.207
 1st Qu.:-0.864   1st Qu.:-1.207
 Median :-0.172   Median : 0.018
 Mean   : 0.003   Mean   : 0.007
 3rd Qu.: 0.682   3rd Qu.: 1.243
 Max.   : 3.041   Max.   : 1.243
 NA's   :29301    NA's   :29301
```

Figure 2:Output of summery(data)

- Then Remove the 12th column from the data set because it is the Out-put column.

```
#Remove 12th column from data set(Output column)
data_excluded_clomn12 <- dataSet[, -12]
```

Figure 3:code of removing output column

i) Step 1: Outlier detection

Outliers must be removed from data analyses to improve data quality and ensure that statistical assumptions are met. Outliers may bias results, resulting in skewed conclusions and erroneous models. Outliers are removed to improve data reliability and statistical analysis. However, it is critical to thoroughly analyse outliers' context and potential impact before eliminating them, as they may include useful information or insights.

The image below shows the wine dataset's format with outliers that we should use.

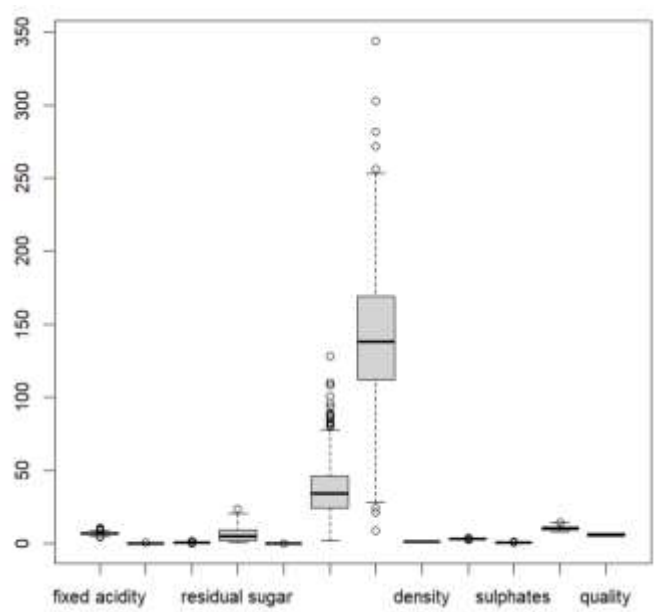


Figure 4:dataset format with outliers

i) Define a function named ‘find_outliers’ to detect outliers using the IQR method. (Interquartile Range)

```r
#Function for find outliers using IQR
find_outliers <- function(x) {
  q1 <- quantile(x, 0.25)
  q3 <- quantile(x, 0.75)
  iqr <- q3 - q1
  lower_bound <- q1 - 1.5 * iqr
  upper_bound <- q3 + 1.5 * iqr
  outliers <- x < lower_bound | x > upper_bound
  return(outliers)
}
```

Figure 5:findoutliers function

The code iterates through each variable in the dataset, calculating the first and third quartiles (Q1 and Q3), IQR, and upper and lower bounds for the feature.

1) use the “find_outliers’’ function for each column and row of the data set.

```r
# use find_outliers function to each columns of the data set
outlier_index <- apply(data_excluded_clomn12, 2, find_outliers)

outlier_rows <- rowSums(outlier_index) > 0
```

2) **After scaling,** check the boxplot and find outliers in columns 2,3,4,6,9. (about scaling in the next step)

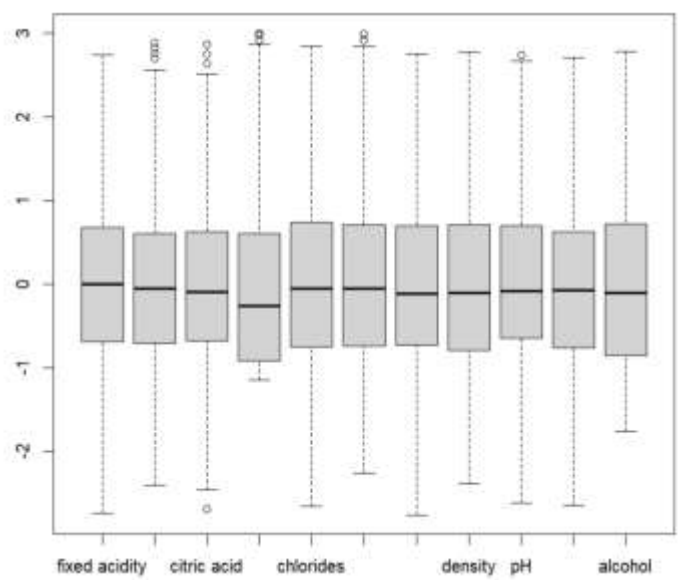


Figure 6:boxplot with remaining outliers

Next, apply and remove the outliers and find a function for those columns of the scaled and uncleaned data.

```
scaled_cleaned_data_step1 <- as.data.frame(scale(cleaned_data_step1))
#check the boxplot in step1
boxplot (scaled_cleaned_data_step1)
# Apply the outliers find function to the 2,3,4,6,9 column of the scaled_cleaned_data
outlier_column_2 <- find_outliers(scaled_cleaned_data_step1[[2]])

outlier_column_3 <- find_outliers(scaled_cleaned_data_step1[[3]])

outlier_column_4 <- find_outliers(scaled_cleaned_data_step1[[4]])

outlier_column_6 <- find_outliers(scaled_cleaned_data_step1[[6]])

outlier_column_9 <- find_outliers(scaled_cleaned_data_step1[[9]])
```

Figure 7:steps for removing outliers 1

```
# Remove outliers from the scaled_cleaned_data dataframe for the specified columns
scaled_cleaned_data_step2 <- scaled_cleaned_data_step1[
  !outlier_column_2 &
    !outlier_column_3 &
    !outlier_column_4 &
    !outlier_column_6 &
    !outlier_column_9,
]
```

Figure 8:steps for remove outliers

ii) Apply the "find_outliers" function again to 3rd column to remove the remaining outliers of the third column.

```
# Apply the outliers detection function to the 3rd column of the cleaned_scaled_data
outlier_column_3_again <- find_outliers(scaled_cleaned_data_step2[[3]])
# Remove outliers from the cleaned_scaled_data dataframe for the third column
cleaned_scaled_data_last <- scaled_cleaned_data_step2[!outlier_column_3_again, ]
```

1) After this, the boxplot looks like the below image,

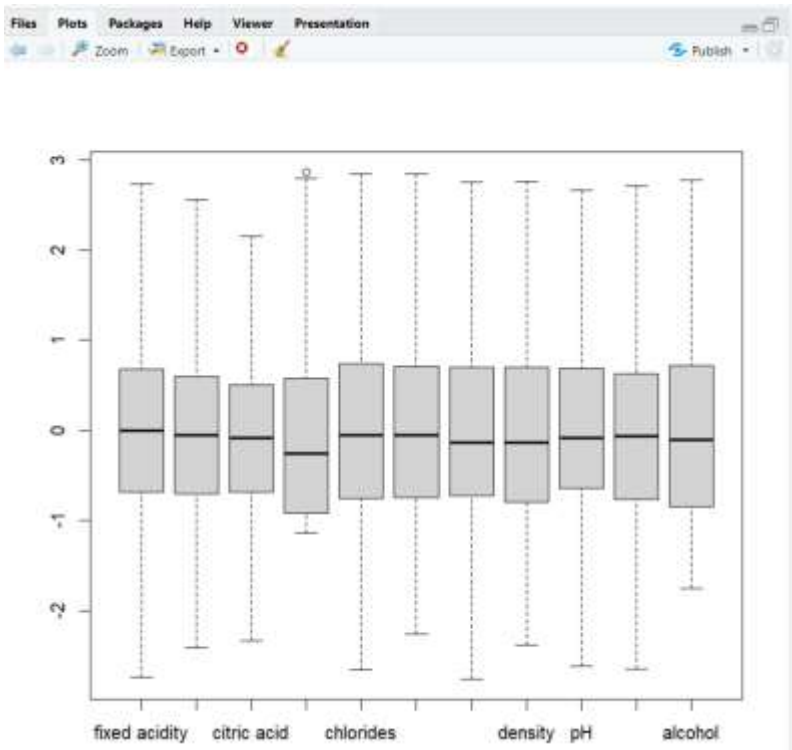


Figure 9:boxplot of after removing outliers

iii) Scaling

Scaling the dataset ensures that features of different scales or units are on the same scale. There are various scaling methods are available, such as min-max, normalization, and standardization.

The algorithm scales the data using Z-score standardization. Using this technique, the data are adjusted so that each characteristic's mean and standard deviation are both.

This is accomplished by subtracting the mean of each feature and dividing by its standard deviation. Scale the data set,

```r
#scale the data set that cleaned
cleaned_data_step1 <- data_excluded_clomn12[!outlier_rows, ]

scaled_cleaned_data_step1 <- as.data.frame(scale(cleaned_data_step1))
```

Figure 10:steps for scaling cleaned data set

Before scaling:

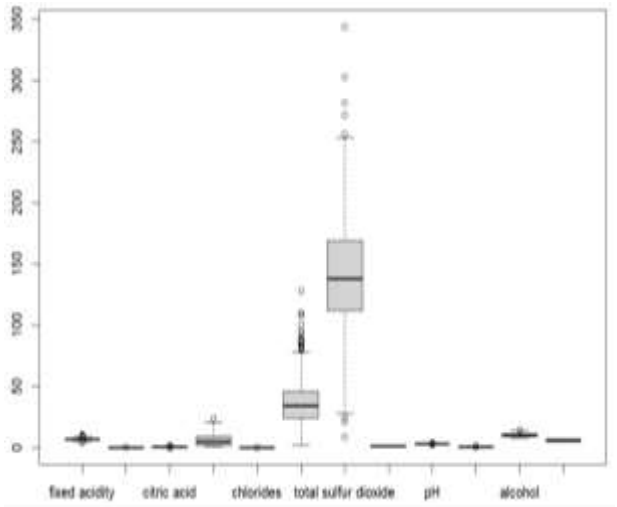


Figure 11:boxplot of before scaling

After scaling:

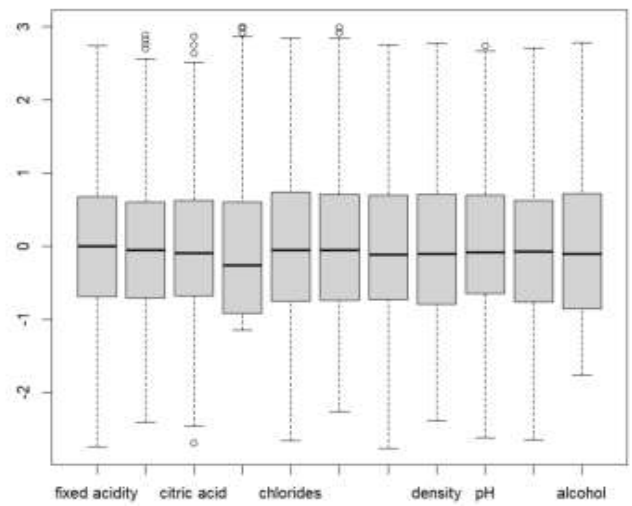


Figure 12: boxplot of after scaling

The below image shows the data representation of data after scaling

```
> head(cleaned_scaled_data_last)
  fixed acidity volatile acidity citric acid residual sugar   chlorides free sulfur dioxide
1   0.954047267        0.5986629  -0.4433167      0.8856225 -0.25465331          -0.9470545
2   0.954047267        0.5986629  -0.4433167      0.8856225 -0.25465331          -0.9470545
3   1.227704100       -0.4468777  -0.5614537     -0.2924049 -0.05497235           0.7070453
4  -0.003751649       -0.8389555   0.8561905      0.3119744 -1.05337715           1.1205703
5   0.133076768       -1.4924184   0.1473684     -0.9787340 -0.35449379          -0.9470545
7   2.048674598       -0.9696481   1.2106016     -0.8353219  0.94343244          -1.0159753
  total sulfur dioxide      density         pH  sulphates    alcohol
1            0.1074955  0.906831998 -1.1355770 -1.3558876 -0.9317326
2            0.1074955  0.906831998 -1.1355770 -1.3558876 -0.9317326
3            0.6214899 -0.410756756 -1.6982854 -0.8603915  0.7977315
4            0.8662491 -0.029349485  0.1305167 -2.1486813  0.7153761
5            0.8907250 -0.445430144  0.8339021  1.3197913  0.2212435
7           -0.8960174  0.005323903 -2.4720093 -0.2657962 -0.8493772
```

Figure 13:output of dataset after scaling

b) determine the number of cluster centers via four “automated tools”

The number of cluster centers must be determined using "automated tools". This project uses the NbClust, Elbow, Silhouette, and Gap Statistic methods as automated tools.

i) NBClust method

Determine the appropriate number of clusters in a dataset. It comprises numerous indices that determine the number of clusters in a data set and provide the user with the best clustering strategy based on various outcomes.

1) Install and apply the NBClust method.

```
#install the "NbClust" package [NBclust Method]
library(NbClust)
#Number of cluster centers
clusterNo=NbClust(data,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",index="all")
```

Figure 14:steps for NbClust method

According to the NbClust method, 2 clusters are recommended. we can ensure that from the outputs below.

(a) NbClust “Euclidean” distance Output

```
> #Distance= euclidean
> clusterNo=NbClust(cleaned_scaled_data_last,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",in
dex="all")
*** : 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
* 5 proposed 4 as the best number of clusters
* 1 proposed 9 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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


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

Figure 15:Steps for NbClust euclidean distance

NbClust "Euclidean" distance Output's boxplot is in Appendix A.1.1

- The majority rule and a number of indices, including the Hubert and D indices, suggest that there should be two clusters for the dataset in question.
- The distances between data points were determined using the Euclidean distance metric in order to carry out cluster analysis. The majority rule recommended two clusters as the ideal number, despite other indices and methodologies (Hubert index, D index) suggesting different optimal numbers of clusters (2, 3, 4, 6, and 14).

(a) NbClust “Manhattan” distance Output

```
#Distance= manhattan
clusterNo=NbClust(data, distance="manhattan", min.nc=2,max.nc=15,method="kmeans",index="all")
```

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

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

*******************************************************************
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 5 proposed 3 as the best number of clusters
* 5 proposed 4 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 1 proposed 9 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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


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

Figure 16:Code for find manhattn distance

(b) NbClust “Maximum” distance Output

```
#Distance= maximum
clusterNo=NbClust(data, distance="maximum", min.nc=2,max.nc=15,method="kmeans",index="all")
```

Figure 17: Code for finding maximum distance 1

```
*******************************************************************
* Among all indices:
* 8 proposed 2 as the best number of clusters
* 5 proposed 3 as the best number of clusters
* 5 proposed 4 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 2 proposed 12 as the best number of clusters
* 2 proposed 15 as the best number of clusters

                   ***** Conclusion *****

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


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

Figure 18:2 Code for maximum distance

ii) Elbow method

The elbow method is a methodology for determining the most appropriate number of clusters in a dataset for clustering analysis, such as K-means clustering. It works by charting the within-cluster sum of squares (WCSS) against the **number of clusters and detecting the "elbow" point**, which is where the rate of reduction in

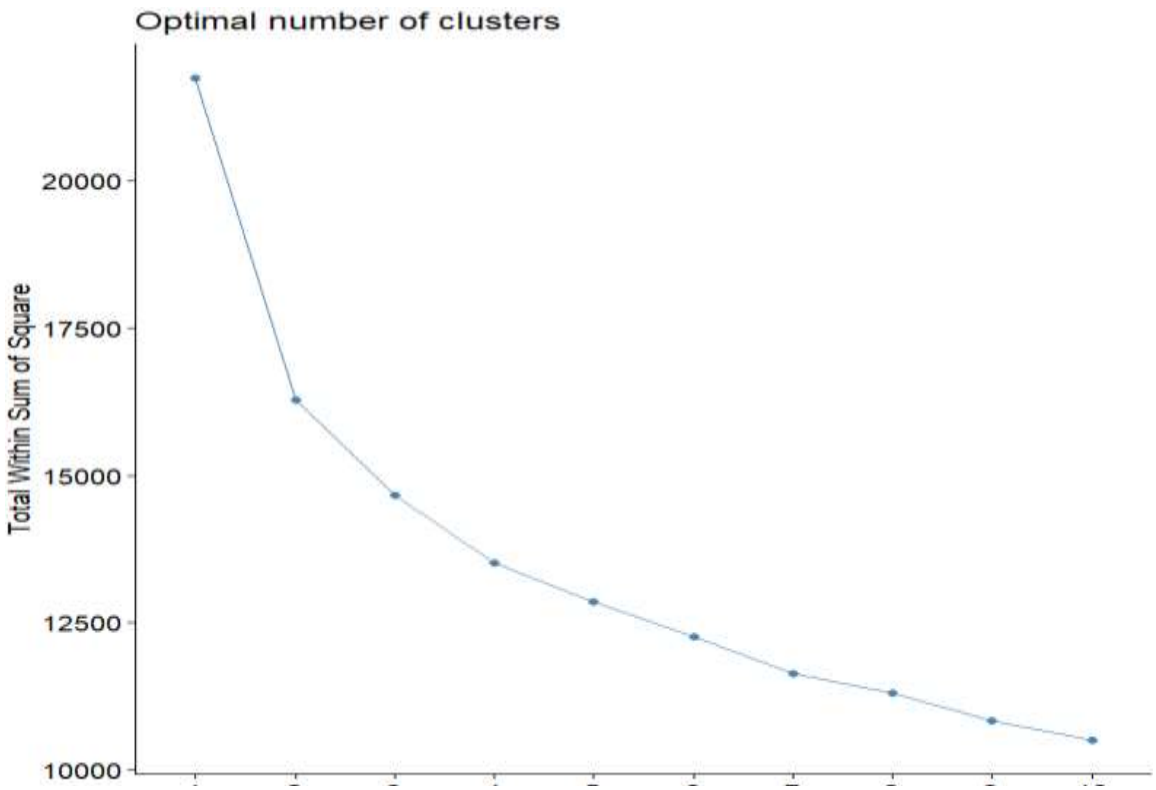


Figure 19:boxplot for elbow method

WCSS changes dramatically.

Because, the change in the within-cluster sum of squares (WCSS) is lowest at k=2 when compared to other k values, the elbow technique proposes using that number of clusters. This choice strikes a balance between decreasing the WCSS and avoiding unnecessary complexity in the clustering model.

iii) Gap Statistics method

Gap statistics is a method for determining the appropriate number of clusters in a dataset. It compares the data's within-cluster dispersion to a reference distribution to assess whether the clustering structure is statistically significant. The appropriate number of clusters is determined by the biggest difference between the data and reference dispersion.

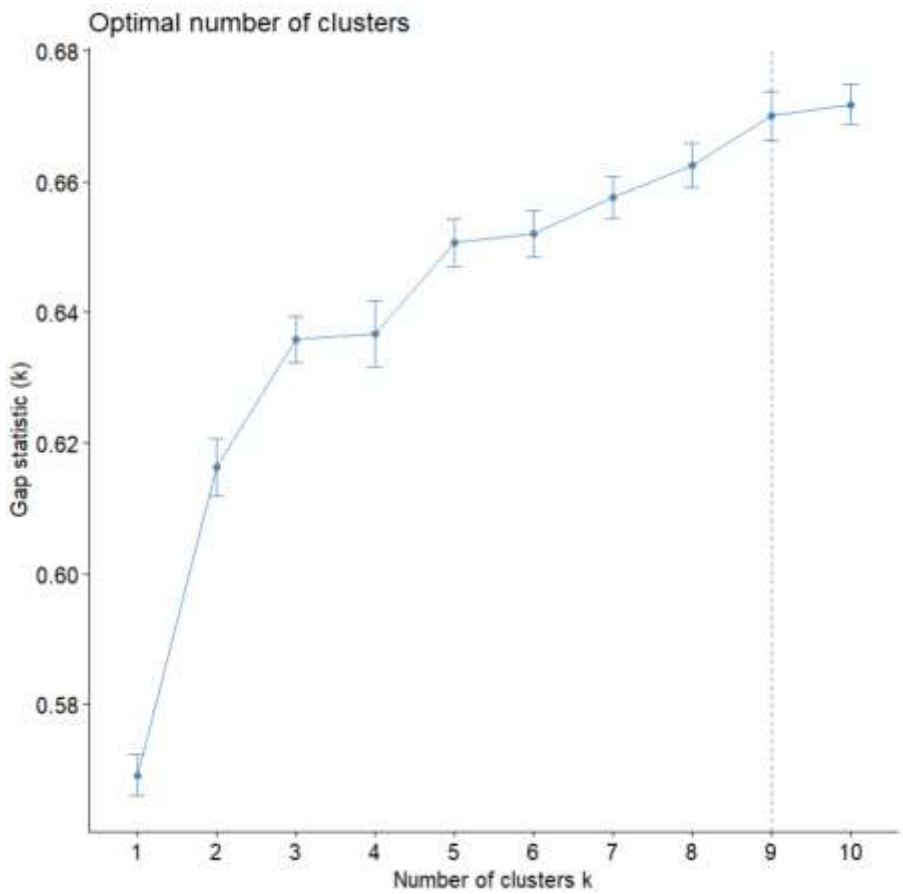


Figure 20:boxplot for gap satistics method

- Gap statistics method suggests using 3 clusters.

iv) silhouette method

The silhouette approach assesses cluster quality by comparing each point's similarity to its own cluster to that of other clusters. A higher average silhouette score suggests greater clustering.

- Silhouette's method suggests using 2 clusters.

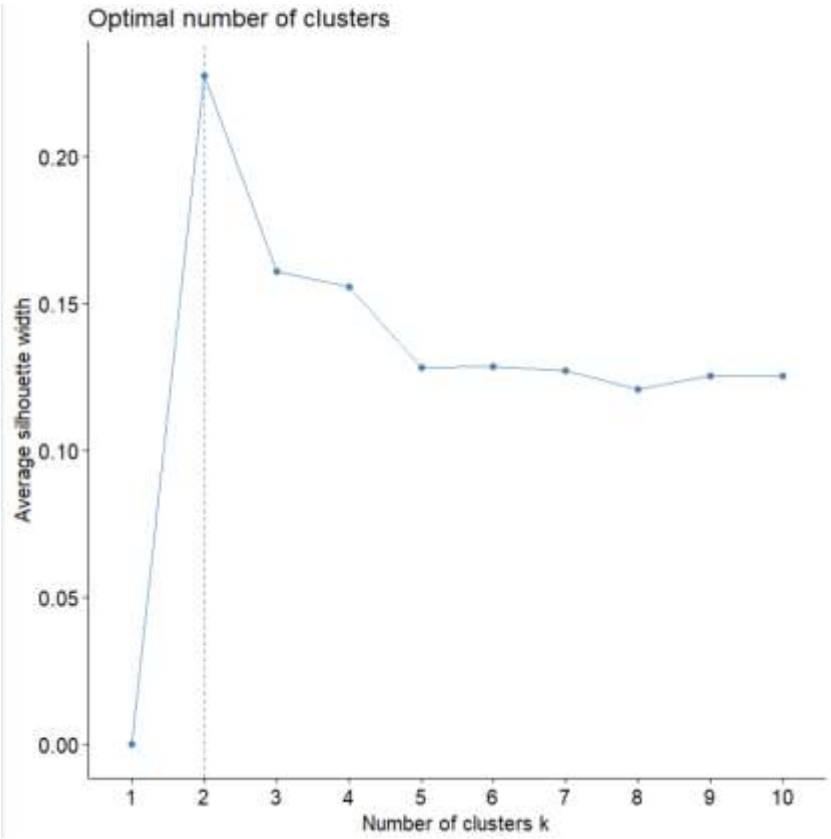


Figure 21:boxplot of silhouette's method

c) K-means clustering investigation

- The data is divided into a predetermined number of groups (K), which, in this example, are two clusters, using the K-means clustering method.
- Furthermore, it gives the sizes of each cluster—the number of data points that are part of each cluster.
- **Cluster Means**: Each row corresponds to a cluster, and each column represents a feature in the dataset. The output shows the mean values of each cluster's features (columns) in the dataset. These indicate the centroids or centres of the clusters in the feature space.
- **Clustering Vector**: The clustering vector designates a distinct cluster for every data point. It displays the cluster to which every observation belongs.
- **Within Cluster Sum of Squares** (WCSS): This type of sum of squares gauges how compact a cluster is within another cluster. It shows the total squared distances between every data point and the cluster centroid that corresponds to it. Lower WCSS values show tighter clusters and improved clustering ability.

```
> wss = kc$withinss
> wss
[1] 10657.264  7024.343
> #total of wss
> wss=kc$tot.withinss
> wss
[1] 17681.61
```

Figure 22:outputs for wss

- **The Between Cluster Sum of Squares (BCSS)** is a statistical tool that quantifies the distance between clusters. The sum of squared distances between cluster centroids is what it represents. Higher BCSS scores indicate greater separation between clusters and improved discrimination.

```
> bss = kc$betweenss
> bss
[1] 5430.108
```

Figure 23:output for bss

- Total **Sum of Squares (TSS)**: The sum of squared distances between each data point and the dataset's mean is known as the total sum of squares. It is a

```
> TSS = kc$tot.withinss + kc$betweenss
> TSS
[1] 23140.09
```

Figure 24:output for TSS

representation of the entire data variance.

- **Iter**: The number of iterations the K-means algorithm went through in order to converge is shown by this component.
- **Ifault**: The ifault component shows whether there were any problems when the algorithm was being executed. Successful convergence is usually indicated by a value of 0.

```
K-means clustering with 2 clusters of sizes 927, 1367

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide
1     0.1656687       0.02729504 -0.03747606      0.8367978  0.1538859           0.4858739
2    -0.1130669      -0.14845742 -0.09369382     -0.6139282 -0.3150296          -0.4174604
  total sulfur dioxide    density          pH    sulphates    alcohol    quality
1            0.7082726  0.9128556 -0.17147750 -0.006690078 -0.7470771 -0.4247125
2           -0.5469335 -0.6981345  0.09417411 -0.092551520  0.5653794  0.3540323

Clustering vector:
   [1] 1 2 2 1 2 1 2 1 2 2 2 1 2 2 1 2 2 2 2 2 1 1 1 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 2 2 2 1 2 1 2
  [47] 1 2 1 2 2 1 1 2 1 2 1 1 1 1 1 1 1 2 2 2 1 2 2 1 1 1 1 1 1 2 1 2 2 1 2 2 1 2 1 2 2 1 2 1 2
  [93] 1 2 1 1 1 2 1 1 1 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
 [139] 2 2 2 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 2 2 1 2 2 2 2 1 1 1 1 1 2 2 1 1 1 2 2 2 2 1 2 2 1 2 1
 [185] 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1
 [231] 1 2 1 1 2 2 2 1 1 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 1 2 1 1 1 2
 [277] 1 1 2 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 1 1 1 2 1 1 1 1 1 1 2 1 1
 [323] 2 1 1 1 2 1 1 1 1 1 1 1 2 2 2 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 2 1 2 1
 [369] 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 2 2 2 1 1 1 1 1 2 2 2 1 1 2 2 1 2 1 2 2 2 1 1
 [415] 1 1 2 1 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
 [461] 2 2 2 1 2 1 2 2 1 1 1 2 2 2 2 1 2 1 1 2 2 1 2 2 2 1 2 1 2 2 1 2 1 2 1 1 1 1 2 2 1 1 2 2 2 1
 [507] 1 1 2 2 1 1 1 2 1 1 1 2 2 1 1 2 1 1 2 1 1 2 1 1 1 1 1 2 2 1 2 2 2 1 1 1 1 2 1 2 2 2 2 1 1 2
 [553] 1 1 2 1 1 2 1 1 2 1 1 1 1 2 2 1 2 1 2 2 2 2 2 2 2 2 1 1 1 1 2 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
 [599] 2 1 1 2 1 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 2 1 2 2 1 2 1 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 1 1 2
 [645] 2 1 1 1 2 2 2 2 2 1 2 2 1 2 2 2 1 1 1 2 1 1 1 1 2 2 1 1 2 1 2 2 2 1 1 2 2 1 1 1 1 1 1 1 1
 [691] 1 1 1 1 1 2 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 2 1 1 1 2 1 2 1 1 1 1 1
 [737] 1 1 1 2 1 1 2 1 1 1 2 2 1 1 2 1 2 2 2 2 2 2 1 2 1 2 2 2 2 1 2 2 1 2 2 2 1 1 2 2 1 1 2 2 1 2
 [783] 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 2 2 1 1 1 1 1 2 2 2 2 2 2 1
 [829] 1 1 1 1 1 1 1 1 1 2 2 2 1 2 1 1 1 1 1 2 2 1 2 2 2 1 1 1 2 1 1 2 2 2 1 1 1 1 2 2 2 1 2 1 2 2
 [875] 1 2 1 1 2 2 1 2 1 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 1 1 2 1 2 2 2 2 2 2
 [921] 2 2 2 2 2 1 2 1 2 2 2 2 1 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 1 2 1 2 2 2 1 1 2 1 1
 [967] 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 1 1 1 2 1 2 1 2 1 2
 [ reached getOption("max.print") -- omitted 1294 entries ]

Within cluster sum of squares by cluster:
[1]  6738.729 10481.672
 (between_SS / total_SS =  24.0 %)
```

Figure 25:outputs of Kmeans

```
Available components:

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

Figure 26: 2 Outputs of Kmeans

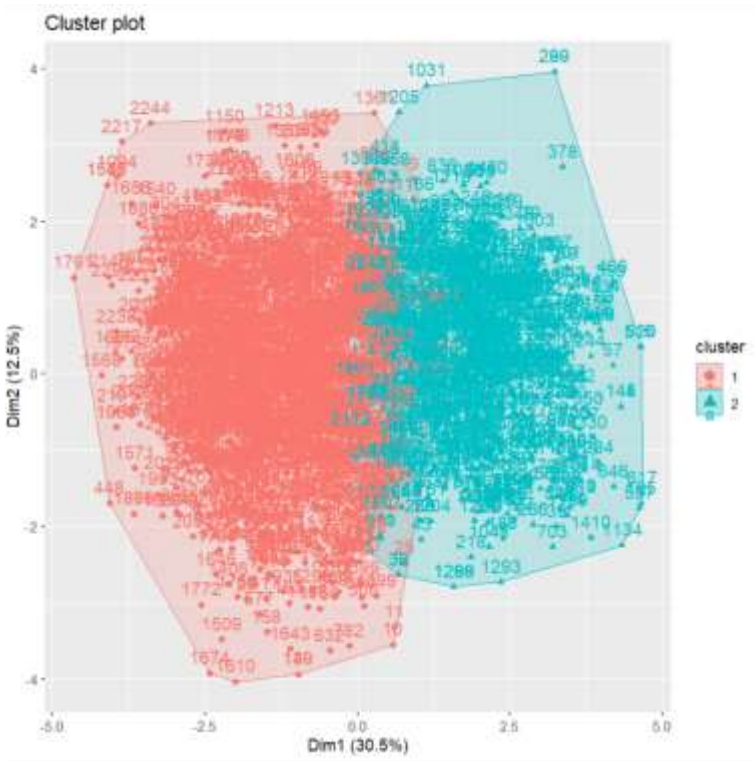


Figure 27:cluster plot for 2 clusters

d) About silhouette plot

```
> fviz_silhouette(sil)
  cluster size ave.sil.width
1       1  883          0.23
2       2 1276          0.21
```

Figure 28: output about cluster size

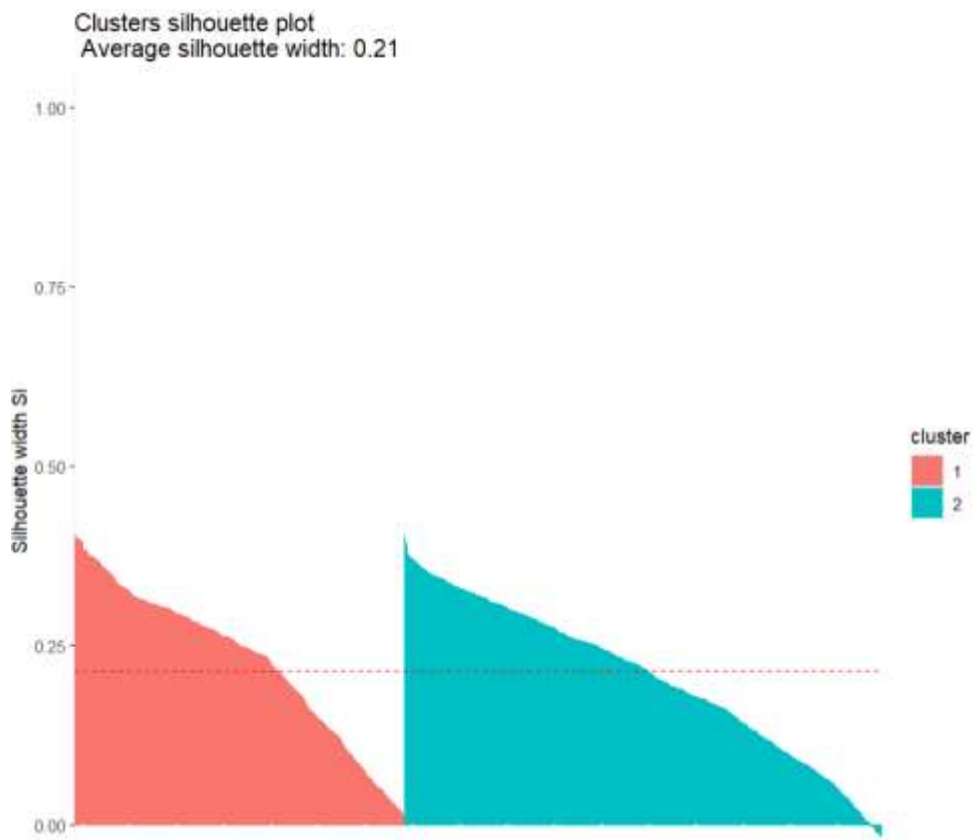


Figure 29: silhouette plot

The image displays a clustering analysis silhouette plot. Each cluster's average silhouette width and size are listed in the table above the graphic. With an average silhouette width of 0.23, 883 data points make up Cluster 1. A mean silhouette width of 0.21 is found in 1276 data points in Cluster 2. It is mentioned below the plot that the average silhouette width for both clusters is 0.21. The average silhouette width serves as a quality measure of the clustering; a greater number indicates a better fit between the clustering and the data. Strong clustering is often represented by a number more than 0.5, and weak clustering is typically represented by a value less than 0.2.

Subtask 2 - Principal Component Analysis (PCA)

Principal component analysis, or PCA, is a method of decreasing the dimensionality of a dataset by converting its original variables into a new set of uncorrelated variables. It helps to simplify complicated datasets while preserving the highest level of volatility.

Step 01: Calculate the covariance matrix

```r
# Compute the Covariance Matrix
wine_cov <- cov(cleaned_scaled_data_last)
wine_cov
```

- Calculate the covariance matrix after getting the cleaned and scaled dataset.

Figure 30:code for covariance matrix

```
> wine_cov
                      fixed acidity volatile acidity citric acid residual sugar    chlorides
fixed acidity            1.00000000     -0.033000996  0.26676666     0.12710347  0.034275668
volatile acidity        -0.03300100      1.000000000 -0.17615262     0.09208221  0.059305744
citric acid              0.26676666     -0.176152622  1.00000000     0.07169087  0.139136435
residual sugar           0.12710347      0.092082209  0.07169087     1.00000000  0.102422622
chlorides                0.03427567      0.059305744  0.13913644     0.10242262  1.000000000
free sulfur dioxide     -0.04282450     -0.059373770  0.08611768     0.33359604  0.095254387
total sulfur dioxide     0.09887031      0.096586399  0.09212444     0.43704804  0.198094931
density                  0.30050473      0.008584229  0.13412873     0.83639210  0.285038133
pH                      -0.44934273     -0.082156366 -0.17427225    -0.21253598 -0.102332797
sulphates               -0.01995607     -0.034915521  0.04057284    -0.03244134  0.001388961
alcohol                 -0.14506541      0.084367697 -0.08673683    -0.46608129 -0.391545201
quality                 -0.13530209     -0.144798914 -0.04958685    -0.16599653 -0.261456085
                      free sulfur dioxide total sulfur dioxide      density          pH
fixed acidity               -0.04282450           0.09887031  0.300504726 -0.44934273
volatile acidity            -0.05937377           0.09658640  0.008584229 -0.08215637
citric acid                  0.08611768           0.09212444  0.134128732 -0.17427225
residual sugar               0.33359604           0.43704804  0.836392096 -0.21253598
chlorides                    0.09525439           0.19809493  0.285038133 -0.10233280
free sulfur dioxide          1.00000000           0.60922356  0.327237672 -0.01437526
total sulfur dioxide         0.60922356           1.00000000  0.563916908 -0.03303686
density                      0.32723767           0.56391691  1.000000000 -0.11413475
pH                          -0.01437526          -0.03303686 -0.114134755  1.00000000
sulphates                    0.06133382           0.11417070  0.066361194  0.17852362
alcohol                     -0.26029200          -0.46868612 -0.804387458  0.13851073
quality                     -0.04678736          -0.25007836 -0.390189151  0.12190841
                        sulphates      alcohol     quality
fixed acidity        -0.019956071 -0.145065406 -0.13530209
volatile acidity     -0.034915521  0.084367697 -0.14479891
citric acid           0.040572839 -0.086736830 -0.04958685
residual sugar       -0.032441345 -0.466081292 -0.16599653
chlorides             0.001388961 -0.391545201 -0.26145609
free sulfur dioxide   0.061333823 -0.260292002 -0.04678736
total sulfur dioxide  0.114170697 -0.468686120 -0.25007836
density               0.066361194 -0.804387458 -0.39018915
pH                    0.178523619  0.138510730  0.12190841
sulphates             1.000000000 -0.002283024  0.07355058
alcohol              -0.002283024  1.000000000  0.50613553
quality               0.073550580  0.506135525  1.00000000
```

Figure 31:covariance matrix output

Step 02: Get the covariance matrix's eigenvalues and eigenvectors**.**

```r
# Compute the Eigenvalues
wine_eigen <- eigen(wine_cov)
# Access the Eigenvalues
wine_eigen$values
#To access eigenvalues and eigenvectors separately
wine_eigen$vectors
```

Figure 32: code for covariance matrix's eigenvalues and eigenvectors.

```
> wine_eigen <- eigen(wine_cov)
> # Access the Eigenvalues
> wine_eigen$values
 [1] 3.5306040 1.5891063 1.2997647 1.1326832 0.9691715 0.9163063 0.7260270 0.6849826 0.5241535 0.3320978 0.2817914 0.0133117
>
> #To access eigenvalues and eigenvectors separately
> wine_eigen$vectors
            [,1]        [,2]          [,3]        [,4]        [,5]       [,6]        [,7]        [,8]        [,9]
 [1,]  0.16968707  0.54708734 -0.2061442512  0.12796408 -0.23025115  0.1789177 -0.23424752  0.17718127  0.60447110
 [2,]  0.02478260 -0.01147674  0.5667234193  0.34797567 -0.55392905 -0.2122031  0.11657795 -0.30762303  0.09538287
 [3,]  0.11949027  0.30875376 -0.4842658565 -0.23850571 -0.10844521 -0.3085917  0.09645163 -0.67431047 -0.12415980
 [4,]  0.39660819 -0.06147155 -0.0044010012  0.35033087  0.12073975  0.2713218  0.40791800 -0.19843474 -0.15537869
 [5,]  0.22304663  0.05698565  0.1942413609 -0.52435404 -0.09984866 -0.4206675  0.49746788  0.31412016  0.19715602
 [6,]  0.26833641 -0.34181065 -0.2860463701  0.25717360  0.06547818 -0.4761586 -0.21888152  0.17747442 -0.07528402
 [7,]  0.38285290 -0.27378251 -0.1060398281  0.16347133 -0.13344804 -0.2655623 -0.25067592  0.04480992  0.20990137
 [8,]  0.49142601 -0.04179983  0.0008128278  0.02671403  0.05668091  0.3242663  0.16539226 -0.10473539  0.11755549
 [9,] -0.14641374 -0.56315787 -0.0036834103 -0.29297828  0.06778233  0.1670352 -0.05979682 -0.39469098  0.57299922
[10,]  0.01401788 -0.27151656 -0.3146560468 -0.21765108 -0.73768993  0.3365153  0.01984105  0.19350953 -0.28132899
[11,] -0.43824740  0.02484748 -0.1099424737  0.26837182 -0.16978998 -0.1805914  0.03894740 -0.09905872  0.06610358
[12,] -0.27475766 -0.11482460 -0.3998620316  0.33040551  0.03034477 -0.0315808  0.60396607  0.18362544  0.26776622
            [,10]        [,11]        [,12]
 [1,]  0.20214794 -0.069997891 -0.174901263
 [2,] -0.06066919 -0.289992283 -0.005249824
 [3,] -0.08725327 -0.069965933 -0.009168246
 [4,]  0.30811075  0.286363153 -0.470240811
 [5,]  0.19221110  0.157602853 -0.024394367
 [6,]  0.44512916 -0.385187253  0.024550797
 [7,] -0.52691333  0.513482390 -0.044282982
 [8,]  0.03292841 -0.072784345  0.767562601
 [9,]  0.17888651 -0.052392319 -0.143670100
[10,]  0.05986467 -0.002907425 -0.041170582
[11,]  0.43276809  0.570544916  0.365048254
[12,] -0.34092624 -0.231648807  0.016248470
```

Figure 33:out puts of covariance matrix's eigenvalues and eigenvectors.

- Calculating Cumulative Score,

```
> PVE <- wine_eigen$values / sum(wine_eigen$values)
> round(PVE,2)
 [1] 0.32 0.14 0.11 0.10 0.09 0.07 0.06 0.05 0.03 0.03 0.00
```

Figure 34:PVE output

The first six components of this scenario account for 83% of the variability, with the eighth accounting for 6%. The combined components account for 89% of the variability. It is frequently helpful to plot the PVE and cumulative PVE.

So data is explained by 7 components.

The PVE plot and cumulative PVE tools used to determine the contribution of each principal component to the overall variance in the dataset:

```
> components <- 7
> (wine_matrix <- wine_eigen$vectors[,1:components])
             [,1]        [,2]        [,3]        [,4]        [,5]         [,6]        [,7]
 [1,]  0.139044672  0.63815119 -0.11740606 -0.14919305 -0.19952084 -0.008244369 -0.02088784
 [2,]  0.006136882 -0.08925686  0.62764425 -0.16812210 -0.55110103  0.371683217  0.23549149
 [3,]  0.013101379  0.28227957 -0.39611508 -0.34285826  0.15944884  0.339049258  0.60783998
 [4,]  0.409210238  0.03433666  0.25967687  0.07995808  0.03053336 -0.405157165  0.40812352
 [5,]  0.342787818 -0.03463001 -0.24692379  0.27995437 -0.16286100  0.589064617 -0.23330757
 [6,]  0.283681484 -0.20013228  0.12496087 -0.53765250  0.43745635 -0.014525605 -0.20458560
 [7,]  0.403128665 -0.17315648  0.09362355 -0.35599638  0.10017175  0.242660166 -0.14532339
 [8,]  0.498693337  0.01429883 -0.01446781  0.17180298 -0.08510749 -0.210693115  0.25336846
 [9,] -0.089919262 -0.61621884 -0.26347283  0.11644215  0.01723623  0.097770564  0.43408389
[10,]  0.050611198 -0.22578665 -0.43160963 -0.43177601 -0.62711014 -0.350083250 -0.14598672
[11,] -0.439512057  0.04231366  0.16331935 -0.31913582  0.03727227  0.008709998  0.12934489
```

Figure 35:PC values

```
> # Assign row and column names to the dataframe
> phi <-- as.data.frame(wine_matrix)
> row_nam <- c("fixed.acidity", "volatile.acidity", "citric.acid", "residual.sugar", "chlorides", "free.
sulfur.dioxide", "total.sulfur.dioxide", "density", "pH", "sulphates", "alcohol")
> col_nam <- c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6", "PC7")
>
> rownames(phi) <- row_nam
> colnames(phi) <- col_nam
>
> print(phi)
                         PC1         PC2         PC3         PC4         PC5          PC6
fixed.acidity        -0.139044672 -0.63815119  0.11740606  0.14919305  0.19952084  0.008244369
volatile.acidity     -0.006136882  0.08925686 -0.62764425  0.16812210  0.55110103 -0.371683217
citric.acid          -0.013101379 -0.28227957  0.39611508  0.34285826 -0.15944884 -0.339049258
residual.sugar       -0.409210238 -0.03433666 -0.25967687 -0.07995808 -0.03053336  0.405157165
chlorides            -0.342787818  0.03463001  0.24692379 -0.27995437  0.16286100 -0.589064617
free.sulfur.dioxide  -0.283681484  0.20013228 -0.12496087  0.53765250 -0.43745635  0.014525605
total.sulfur.dioxide -0.403128665  0.17315648 -0.09362355  0.35599638 -0.10017175 -0.242660166
density              -0.498693337 -0.01429883  0.01446781 -0.17180298  0.08510749  0.210693115
pH                    0.089919262  0.61621884  0.26347283 -0.11644215 -0.01723623 -0.097770564
sulphates            -0.050611198  0.22578665  0.43160963  0.43177601  0.62711014  0.350083250
alcohol               0.439512057 -0.04231366 -0.16331935  0.31913582 -0.03727227 -0.008709998
                        PC7
fixed.acidity        0.02088784
volatile.acidity    -0.23549149
citric.acid         -0.60783998
residual.sugar      -0.40812352
chlorides            0.23330757
free.sulfur.dioxide  0.20458560
total.sulfur.dioxide 0.14532339
density             -0.25336846
pH                  -0.43408389
sulphates            0.14598672
alcohol             -0.12934489
```

Figure 36:PC values with new row names

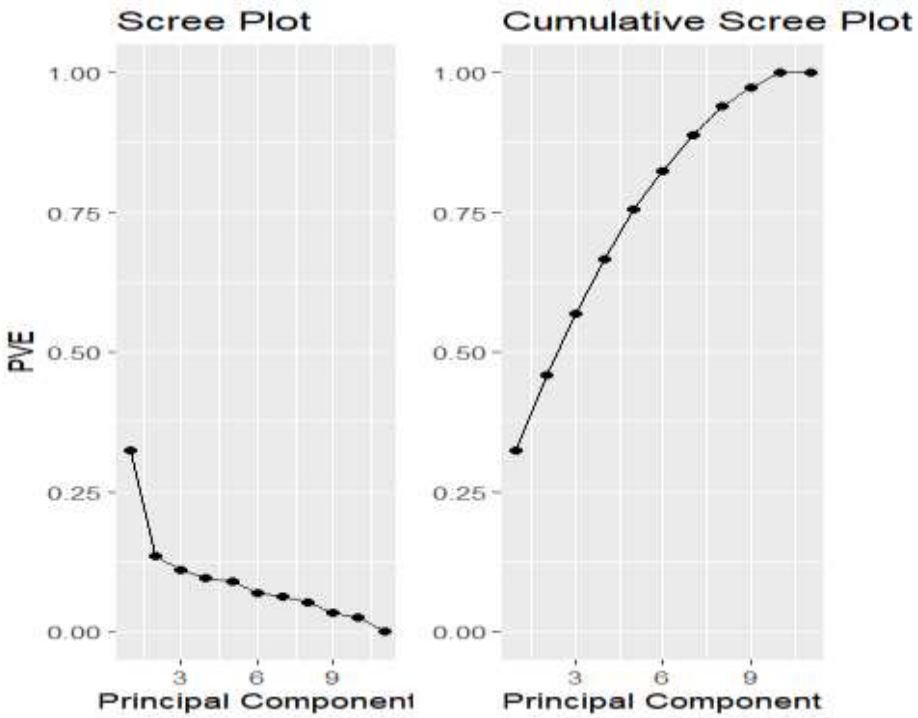


Figure 37: scree plot and culmative plot

Step 03: Defining the number of Cluster Centers for the PCA dataset

a) NbClust Method

NbClust is one of the methods that can find the ideal number of cluster centers. before using this method we have to install “” package.

```
# e part-Automated Tools on PCA-based Data set
library(NbClust)
#Distance= euclidean
clusterNo=NbClust(cleaned_scaled_data_last,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",index="all")
```

- The Nbclust method suggests 2 cluster centers. The below outputs show the results.

```
> library(NbClust)
> #Distance= euclidean
> clusterNo=NbClust(scaled_wine,distance="euclidean", min.nc=2,max.nc=10,method="kmeans",index="all")
*** : 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:
* 10 proposed 2 as the best number of clusters
* 4 proposed 3 as the best number of clusters
* 7 proposed 6 as the best number of clusters
* 1 proposed 7 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


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

Figure 38:output of PCA euclidean distance

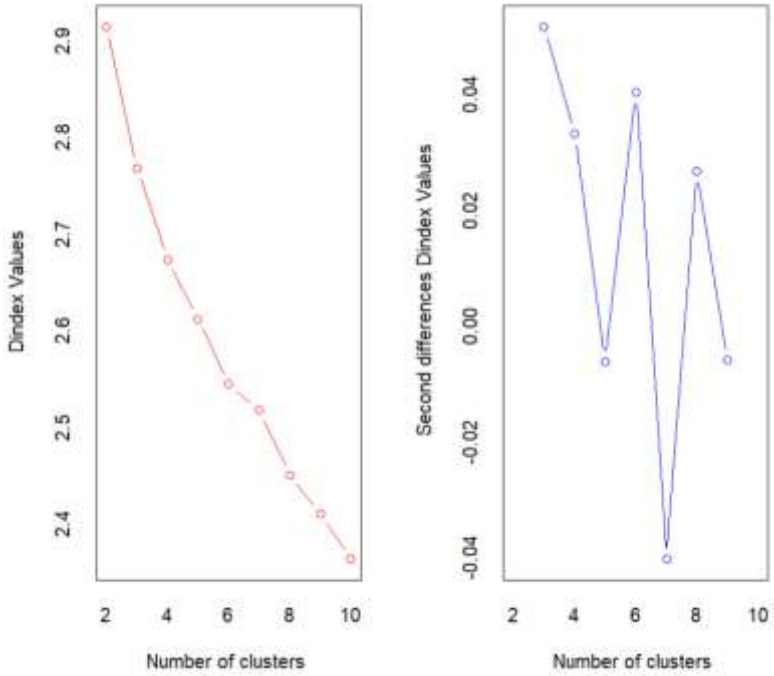


Figure 39:boxplot of clusters

b) Elbow method

In PCA, the "elbow" technique is used to choose the number of principal components to maintain based on explained variance.

```
#Elbow method
x= PC_without_state
y<- data$quality

library(factoextra)
fviz_nbclust(x, kmeans, method = 'wss')
```

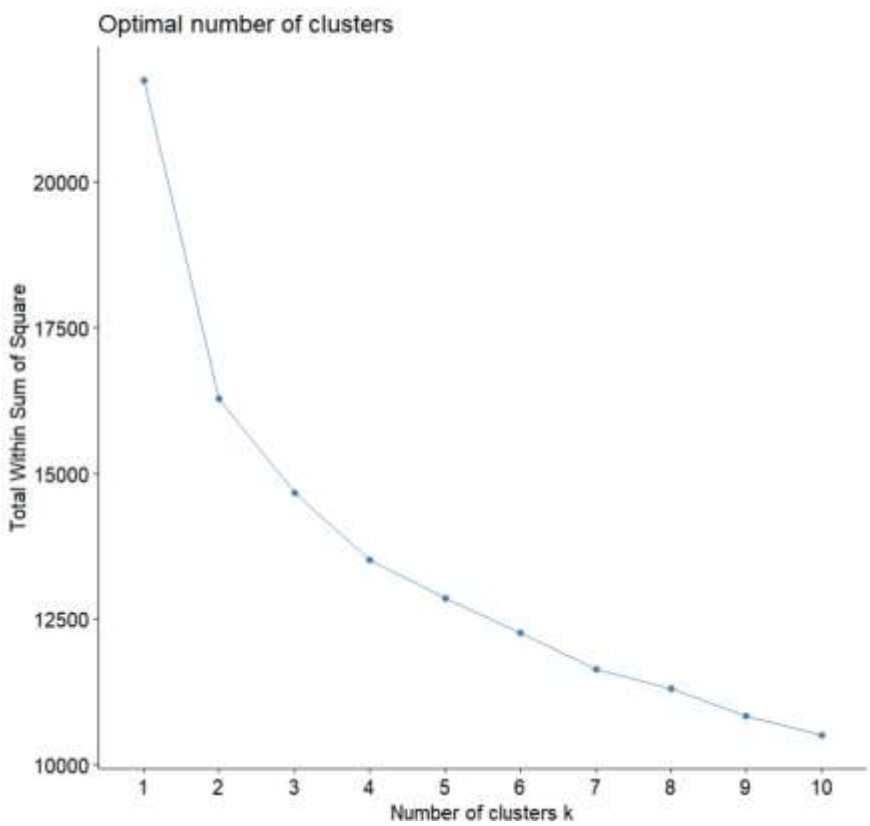


Figure 40:Elbow method graph

- This point, often known as the "elbow" point, represents a fair compromise between collecting enough variance in the data and avoiding overfitting by integrating too many components.

- around k=2 suggests that 2 clusters may be the optimal choice according to this particular clustering analysis method.

c) Silhouette Method

This method assigns a silhouette score to each data point, indicating how similar the point is to its own cluster compared to other clusters. A high average silhouette score implies well-defined clusters, whereas a low score reveals that data points may be assigned to the incorrect clusters.

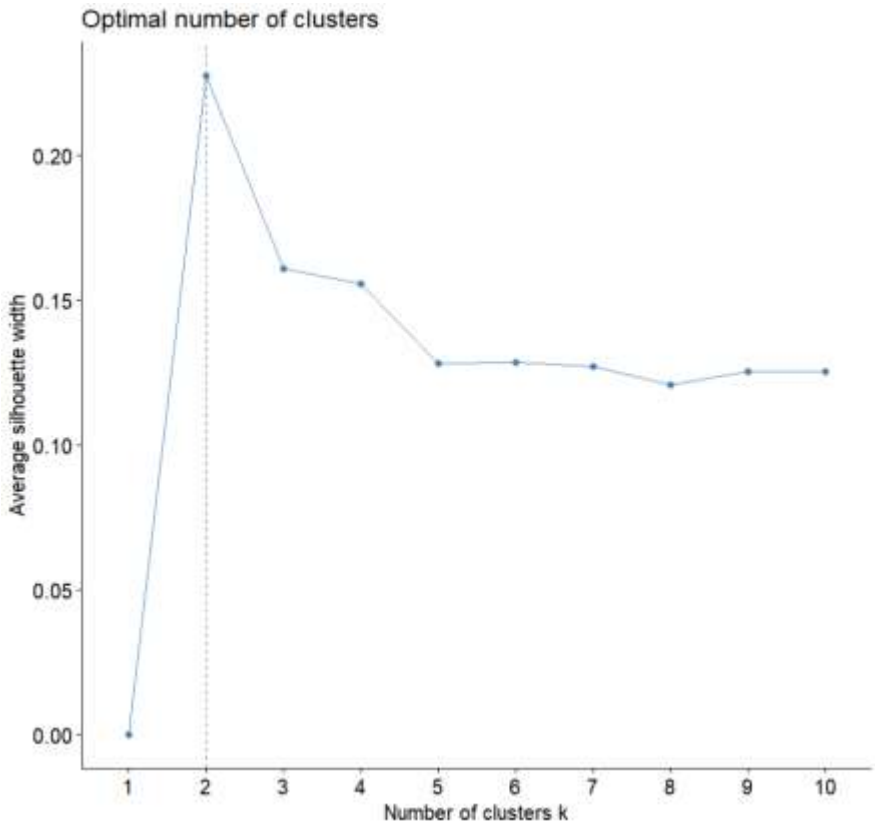


Figure 41:Silhouette method graph

- Silhouette method suggests that 2 clusters.

d) Gap Statistic method

Gap statistics is a technique for determining the appropriate number of clusters in a dataset.

It assesses clustering quality by comparing the data's within-cluster variance to a reference null distribution. The appropriate number of clusters is determined by the biggest difference between the data and reference dispersion.

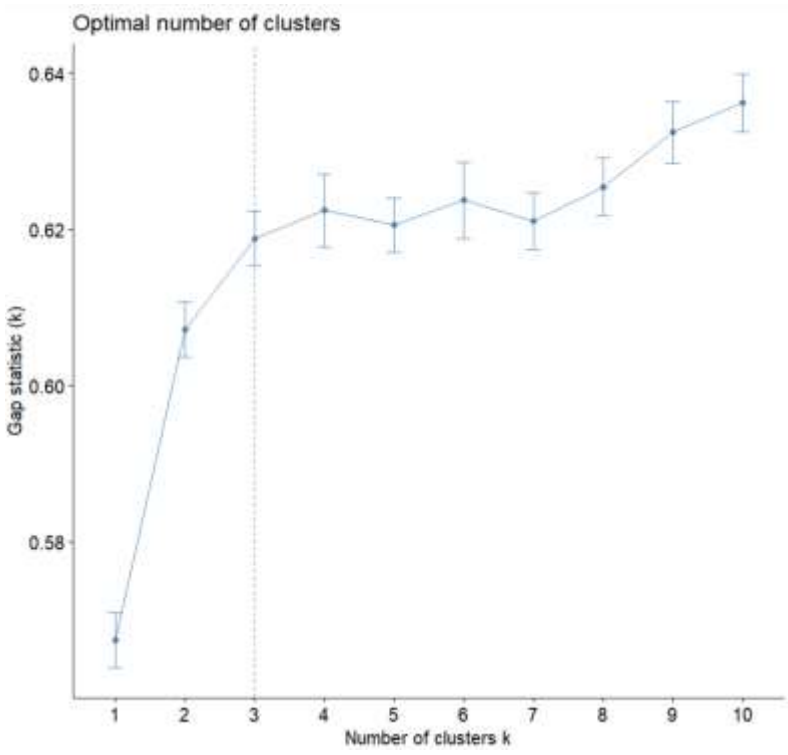


Figure 42: Gap Statistic method graph

- This method suggests that 3 clusters.

Summary of the outputs:

According to each method's outputs, most techniques recommend that two clusters be ideal. So, we get 2 clusters for the next steps.

Step 04: K Means Clustering

```
kc2 <- kmeans(x,2)#K-means clustering with 2 clusters of sizes 888, 1271
kc2
#wss1 and wss2
wss = kc2$withinss #7024.343 10657.264

#total of wss
wss_tot=kc2$tot.withinss #17681.61

#bss
bss = kc2$betweenss # 5458.482

#task h
k=2
# Assuming 'x' is your cleaned and scaled data
library(cluster)
library(factoextra)
x=PC_without_state
sil_width <- silhouette(kc2$cluster, dist(x))
# Calculate silhouette widths
sil <- silhouette(kc2$cluster, dist(x))
fviz_silhouette(sil)

avg_silhouette_width <- mean(sil[,"sil_width"])
cat("average silhouette width score: ",avg_silhouette_width,"\n")
```

Figure 43: code block for clustering

The above image shows the steps of Kmeans clustering.

```
> cluster_assignments <- kc2$cluster
>
> # Print cluster means
> print(cluster_means)
        PC1         PC2         PC3          PC4         PC5          PC6          PC7
1  1.339491 -0.08236713  0.07941596 -0.025814997 -0.01398830 -0.006123213 0.0470964424
2 -1.880731  0.13980736 -0.09627607  0.001682223  0.02097859  0.055842365 0.0009166541
> # Convert cluster_assignments to a simple vector
> cluster_assignments_simple <- unname(cluster_assignments)
>
> # Print cluster assignments without counts
> print(cluster_assignments_simple)
   [1] 2 2 1 1 1 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 2 1 1 1 1 1 1 1 2 1 1 1 2 1
  [45] 1 1 1 1 2 1 1 2 2 2 1 2 2 2 2 2 2 2 2 1 1 1 1 2 1 1 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 1
  [89] 1 1 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2 2 2 2 2 1 1 1 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2 2 2
 [133] 2 2 1 1 2 2 2 1 1 1 1 2 2 2 1 1 2 1 1 1 1 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1 2 2 2 1 2
 [177] 2 2 2 1 2 2 2 2 1 2 1 2 2 2 1 1 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 1 1 2 2 2 2 2 2 1 1 1 1 2
 [221] 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 1 2 1 2 1 2 2 2 2 2 1 1 2 2 2 2 2 2
 [265] 2 2 2 2 2 2 2 2 2 1 1 1 2 1 2 1 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2 2 2 2 1 2
 [309] 2 2 2 2 2 1 1 2 2 2 2 2 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 2 2 2 2 2
 [353] 2 1 1 1 2 2 2 2 2 1 1 2 2 1 1 2 1 1 1 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 1 2 1 2 1 1 1 1 1
 [397] 1 1 1 2 2 1 2 2 1 1 2 1 2 1 2 1 1 2 1 1 1 2 1 2 1 1 2 2 2 1 1 1 2 1 2 2 1 1 2 1 1 1 1 1
 [441] 1 2 1 1 2 1 2 1 2 2 2 2 1 1 2 1 1 1 1 2 2 2 1 1 2 2 1 2 2 2 2 1 1 2 2 1 2 2 1 2 2 1 2 2
 [485] 2 2 2 1 1 2 2 2 2 1 1 1 1 1 1 2 2 2 1 2 2 2 2 1 1 2 1 2 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2
 [529] 2 2 2 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 2 2 1 1 1 1 2 2 2
 [573] 2 1 1 2 1 2 2 1 1 2 2 1 1 2 1 2 1 1 2 2 1 2 2 2 2 2 2 1 2 2 1 2 1 1 1 2 2 1 2 2 2 2 2 2
 [617] 2 2 2 2 1 1 1 1 1 2 1 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 2 1 2
 [661] 2 2 1 1 2 1 2 2 2 1 1 2 2 1 2 1 1 1 1 2 1 2 1 2 1 1 1 1 2 2 2 2 1 1 1 2 2 1 1 2 1 1 2 1
 [705] 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 1 1 2 2 2 2 2 1 1 1 1 2 2 2 2 2 2 2 2
 [749] 2 2 2 2 1 1 2 1 2 2 2 2 2 1 1 2 1 1 1 2 2 2 1 2 2 1 1 1 1 2 2 2 1 1 1 2 1 2 1 1 2 2 1 2
 [793] 2 2 1 2 1 2 2 1 2 1 2 1 2 2 2 1 1 1 1 1 1 1 2 2 1 1 1 2 2 2 2 2 1 2 2 2 2 1 2 1 1 1 1 1
 [837] 1 1 1 1 2 2 1 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 1 2 1 1 1 2 2 1 2 2 2 2
 [881] 2 2 2 2 2 1 1 1 1 1 1 1 1 2 2 1 2 2 1 1 1 1 2 2 2 1 2 1 2 1 2 1 1 2 2 2 2 2 1 1 1 2 2 1
 [925] 1 1 2 1 2 2 1 2 1 1 1 2 1 2 1 1 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 1 1 2 2 2 2 1 2 2 1 1 1 1 1
 [969] 1 1 1 1 2 1 1 1 2 1 1 2 2 2 1 1 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 2 1
 [ reached getOption("max.print") -- omitted 1159 entries ]
```

Figure 44:Output clusters

```
> wss = kc2$withinss #6365.091 9925.699
> wss
[1] 6365.091 9925.699
> #total of wss
> wss_tot=kc2$tot.withinss #17681.61
> wss_tot
[1] 16290.79
> #bss
> bss = kc2$betweenss # 5458.482
> bss
[1] 5458.48
```

Figure 45:Outputs of WSS,BSS

```
> # Assuming 'x' is your cleaned and scaled data
> library(cluster)
> library(factoextra)
> x=PC_without_state
> sil_width <- silhouette(kc2$cluster, dist(x))
> # Calculate silhouette widths
> sil <- silhouette(kc2$cluster, dist(x))
> fviz_silhouette(sil)
  cluster size ave.sil.width
1       1  883          0.26
2       2 1276          0.23
> avg_silhouette_width <- mean(sil[,"sil_width"])
> cat("average silhouette width score: ",avg_silhouette_width,"\n")
average silhouette width score:  0.2393768
>
```

Figure 46:code of average silhouette width score

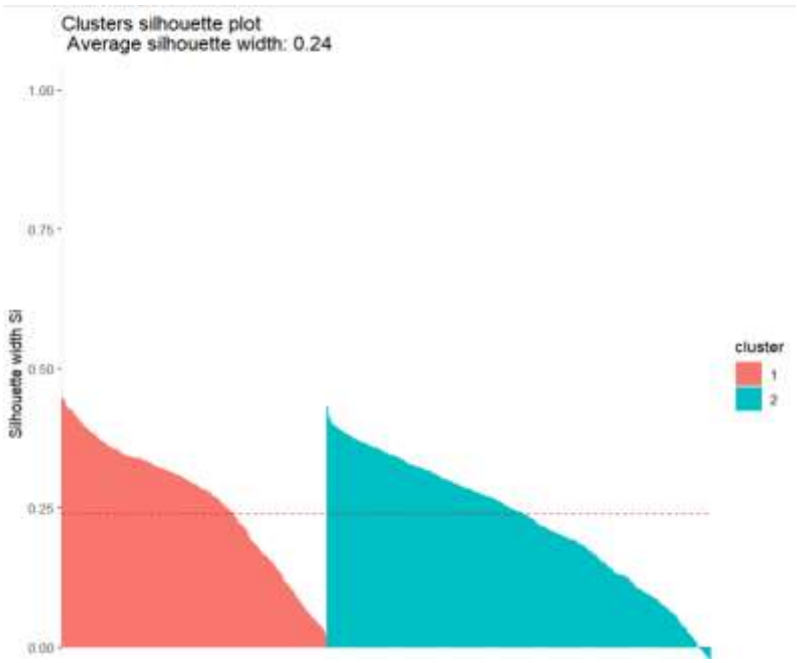


Figure 47:silhouetta method graph

Compared to other clusters, the silhouette plot is a graphic representation that shows how well each data point fits into its designated cluster. With most data points having positive silhouette values, which suggests that they are relatively well-clustered, the silhouette plots in both graphs (before and after pca) have a similar shape. In general, silhouette values above 0.5 are regarded as good, those between 0.25 and 0.5 as moderate, and those below 0.25 as perhaps indicative of problems with the data or the clustering methodology.

The above graph has greater average silhouette width; however, this raises the possibility that the grouping outcomes were improved by the use of PCA.

Step 06: illustrate the Calinski-Harabasz Index

```
> library(fpc)
>
> # Compute Calinski-Harabasz Index
> ch_index <- calinhara(x,kc2$cluster,2)
>
> # Print the index value
> cat("Calinski-Harabasz Index:", ch_index, "\n")
Calinski-Harabasz Index: 781.0734
```

Figure 48:implementation of calinski-harabasz index

This positive index value indicates that there is comparatively more separation between the clusters in the clustering solution produced by the k-means algorithm with two clusters than within-cluster variability. This effectively indicates that each cluster's data points are closely clustered, yet the clusters themselves are significantly separated. A technique for clustering that shows distinct and well-defined clusters is preferable since it may help to recognise and understand the fundamental trends in the data.

So, the Calinski-Harabasz Index results show that the k-means clustering method successfully divided the data into meaningful and well-separated groups. Alongside additional metrics like silhouette width and the within-cluster sum of squares, this evaluation measure completes the overall assessment of the clustering method.

2. Financial Forecasting Part

In this section, We address the choice of input variables for MLP-NN in exchange rate forecasting. Using time-delayed exchange rate values as inputs, we investigate the autoregressive (AR) method. Also explore several approaches for defining input vectors, taking inspiration from the literature. For MLP training and testing, we create input/output matrices and experiment with various time-delayed input configurations. The improvement of MLP performance by data normalization is explored. We want to improve input variable selection and comprehend its effect on the accuracy of exchange rate prediction through these stages.

2.1. MLP model discussion

- The Autoregressive (AR) Method:

  The AR method's simplicity, flexibility, efficiency, and application to a wide range of data patterns make it a good choice for forecasting exchange rates since it can capture temporal relationships. It's crucial to remember that the forecasting horizon and the particulars of the exchange rate data may have an impact on how well the AR approach performs. Determining the AR method's efficacy for a particular forecasting assignment requires experimentation and model review.

  Why is this AR Method most suitable for forecasting exchange rates?

  Time Dependency: Exchange rates frequently show significant temporal dependencies, which indicates a strong correlation between the present rate and previous values. By simulating the link between an observation and a linear combination of its lag values, the AR technique takes direct use of this feature.

  Easy to Understand and Simple in Idea: The AR approach is intuitive and straightforward in idea. It is predicated on the idea that the exchange rate's previous values may be used to anticipate its future values. It is a well-liked option for predicting assignments because of its simplicity, particularly where interpretability is crucial.

Flexibility: The autoregressive model's order may be changed to allow the AR approach to capture varying degrees of temporal dependency. The model may adjust to varied patterns and dynamics seen in the exchange rate data by experimenting with alternative lagged values as input variables.

Other advantages:

- Effective Modelling
- Correspondence to Nonlinear Patterns

Limitations: It is predicated on the idea that historical exchange rate values are adequate to forecast future values, which may only sometimes be the case, particularly in unstable or non-linear markets. It could ignore additional pertinent elements influencing changes in currency rates.

- Other approaches:
  - External Factors:

    Exchange rate fluctuations can be influenced by factors that are not related to the exchange rate data, such as news mood, geopolitical events, and economic statistics like GDP growth rate and inflation rate. Enhancing the model's predictive capacity can be achieved by adding these outside variables as input features.

  - Hybrid approach:-
    - Advantages:
      1. Possibility of alpha production in weather-sensitive industries.
      2. Current and useful meteorological prediction information.
      3. Seasonal tendencies are being taken advantage of for trading chances.
      4. reduction of risk from weather-related incidents.
    - Limitations
      1. Problems with data accuracy and quality in weather predictions.
      2. Model risk arising from unpredictability in the weather-market dynamics.

3. threats to market efficiency if more traders use comparable tactics.

- Technical Indicators:

  Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Bollinger Bands are a few examples of technical indicators that may be used to exchange rate data to give further insight into the dynamics of the market. An enhanced input vector can result from adding these indications as input variables.

2.2 input/output matrices

"I/O matrix" refers to a matrix that contains the neural network's inputs and outputs listings. To use a network for training and forecasting, it is essential to articulate the relationship between inputs and outputs. A neural network model that can learn from the given data and generate forecasts or predictions based on fresh input values may be trained using this matrix.

- One output variable (Output) and four input variables (Input1, Input2, Input3, Input4) are shown in the provided image. (t-4)

	Input1	Input2	Input3	Input4	Output
1	1.3730	1.3860	1.3768	1.3718	1.3774
2	1.3860	1.3768	1.3718	1.3774	1.3672
3	1.3768	1.3718	1.3774	1.3672	1.3872
4	1.3718	1.3774	1.3672	1.3872	1.3932
5	1.3774	1.3672	1.3872	1.3932	1.3911
6	1.3672	1.3872	1.3932	1.3911	1.3838
7	1.3872	1.3932	1.3911	1.3838	1.4171
8	1.3932	1.3911	1.3838	1.4171	1.4164
9	1.3911	1.3838	1.4171	1.4164	1.3947
10	1.3838	1.4171	1.4164	1.3947	1.3675
11	1.4171	1.4164	1.3947	1.3675	1.3801
12	1.4164	1.3947	1.3675	1.3801	1.3744
13	1.3947	1.3675	1.3801	1.3744	1.3759
14	1.3675	1.3801	1.3744	1.3759	1.3743
15	1.3801	1.3744	1.3759	1.3743	1.3787
16	1.3744	1.3759	1.3743	1.3787	1.3595
17	1.3759	1.3743	1.3787	1.3595	1.3599
18	1.3743	1.3787	1.3595	1.3599	1.3624

Showing 1 to 18 of 496 entries, 5 total columns

Figure 49:input and output values of t-4

- One output variable (Output) and three input variables (Input1, Input2, Input3) are shown in the provided image. (t-3)

	Input1	Input2	Input3	Output
1	1.3730	1.3860	1.3768	1.3774
2	1.3860	1.3768	1.3718	1.3672
3	1.3768	1.3718	1.3774	1.3872
4	1.3718	1.3774	1.3672	1.3932
5	1.3774	1.3672	1.3872	1.3911
6	1.3672	1.3872	1.3932	1.3838
7	1.3872	1.3932	1.3911	1.4171
8	1.3932	1.3911	1.3838	1.4164
9	1.3911	1.3838	1.4171	1.3947
10	1.3838	1.4171	1.4164	1.3675
11	1.4171	1.4164	1.3947	1.3801
12	1.4164	1.3947	1.3675	1.3744
13	1.3947	1.3675	1.3801	1.3759
14	1.3675	1.3801	1.3744	1.3743
15	1.3801	1.3744	1.3759	1.3787
16	1.3744	1.3759	1.3743	1.3595
17	1.3759	1.3743	1.3787	1.3599
18	1.3743	1.3787	1.3595	1.3624

Figure 50:Output values of t-3

- One output variable (Output) and two input variables (Input1, Input2, Input3) are shown in the provided image. (t-2)

	Input1	Input2	Output
1	1.3730	1.3860	1.3774
2	1.3860	1.3768	1.3672
3	1.3768	1.3718	1.3872
4	1.3718	1.3774	1.3932
5	1.3774	1.3672	1.3911
6	1.3672	1.3872	1.3838
7	1.3872	1.3932	1.4171
8	1.3932	1.3911	1.4164
9	1.3911	1.3838	1.3947
10	1.3838	1.4171	1.3675
11	1.4171	1.4164	1.3801
12	1.4164	1.3947	1.3744
13	1.3947	1.3675	1.3759
14	1.3675	1.3801	1.3743
15	1.3801	1.3744	1.3787
16	1.3744	1.3759	1.3595
17	1.3759	1.3743	1.3599
18	1.3743	1.3787	1.3624

Figure 51:output,input values of t-2

- One output variable (Output) and one input variables (Input1, Input2, Input3) are shown in the provided image. (t-1)

	Input1	Output
1	1.3730	1.3774
2	1.3860	1.3672
3	1.3768	1.3872
4	1.3718	1.3932
5	1.3774	1.3911
6	1.3672	1.3838
7	1.3872	1.4171
8	1.3932	1.4164
9	1.3911	1.3947
10	1.3838	1.3675
11	1.4171	1.3801
12	1.4164	1.3744
13	1.3947	1.3759
14	1.3675	1.3743
15	1.3801	1.3787
16	1.3744	1.3595
17	1.3759	1.3599
18	1.3743	1.3624

Figure 52:Output,input values of t-1

2.3 Normalization

In data analysis and machine learning, normalisation is an essential preprocessing step. Normalisation process very important for this particular kind of NN because,

- Better Convergence: MLP training is accelerated by normalizing input feature scales to a common value. The gradients of the loss function can vary greatly across dimensions when features have widely disparate scales. During training, this disparity may cause sluggish convergence or oscillation.
- Keeping Activation Functions from Saturating: When input values are too big or too little, disappearing gradients result from activation functions. To avoid saturation and preserve the gradient flow during backpropagation, normalize input features to a common range. This ensures that activations occur within the regions where gradients are non-zero.
- Improved Generalization: Because normalization has broader scales, it keeps some characteristics from taking over the learning process. Normalization makes guarantee that the model learns meaningful patterns from every feature by bringing all of the features to the same scale, which improves generalization performance on unannotated data.

Data scaling is necessary to standardize the input variables. This guarantees that every input variable makes an equal contribution to the neural network's learning process, avoiding the possibility of any one variable's larger size having an undue impact on the network's behavior. All features are treated fairly during training when the range of input values is limited to a common scale, usually between 0 and 1 or -1 and 1.

Furthermore, by decreasing the probability of running into local minima, normalization specifically, the min-max formula—speeds up the neural network's training phase. Regularization keeps gradients from inflating or disappearing, which can hinder convergence or result in less-

```
#to scale every feature in dataset
normalize=function(x){
  return((x-min(x))/(max(x)-min(x)))
}
unnormalize = function(x, min, max) {
  return( (max - min)*x + min )
}
```

than-ideal solutions. It does this by ensuring that gradient magnitudes remain constant throughout all dimensions. Faster and more stable convergence during training is made possible by the neural network's increased ability to traverse parameter space.

Figure 53:function for normalize and unnormalize

2.4 Implementation of Different Models.

In this section, alter input vectors and internal network topologies as we experiment with different MLP models in this phase. This section's goal is to maximize performance by applying common statistical indices such as sMAPE, RMSE, MAE, and MAPE and aim to find the optimal settings for precise forecasts by methodically adjusting model configurations.

1. **Model 1 (with 4 inputs)**

```
#==============1st MLP with 4 input variables and 1 hidden layer=======================

# Train your neural network model
t4_nn_1 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4,
                     data = training_Type_t4,
                     hidden = 12,
                     linear.output = TRUE)

# Plot the model if desired
plot(t4_nn_1)
# Use the model to make predictions on testing data
t4_nn_1_result <- predict(t4_nn_1, testing_Type_t4)
t4_nn_1_result

t4_1_original <- unnormalize(t4_nn_1_result,min_t4_lagged_data,max_t4_lagged_data)
#t4_1_original
t4_1_original_rounded <-as.data.frame(sapply(t4_1_original,round,digits=2))
#t4_1_original_rounded

test_data_t4_1_result <- exchangeUSD_df[401:496,]

t4_1_output <- cbind(test_data_t4_1_result,t4_1_original_rounded )
colnames(t4_1_output) = c("Actual","Predicted")
head(t4_1_output)

> RMSE_t4_nn_1 = rmse(t4_1_output$Actual, t4_1_output$Predicted)
> MAE_t4_nn_1 = mae(t4_1_output$Actual, t4_1_output$Predicted)
> MAPE_t4_nn_1 = mape(t4_1_output$Actual, t4_1_output$Predicted)
> SMAPE_t4_nn_1= smape(t4_1_output$Actual, t4_1_output$Predicted)
> RMSE_t4_nn_1
[1] 0.01029174
> MAE_t4_nn_1
[1] 0.008208333
> MAPE_t4_nn_1
[1] 0.006222391
> SMAPE_t4_nn_1
[1] 0.006221683
```

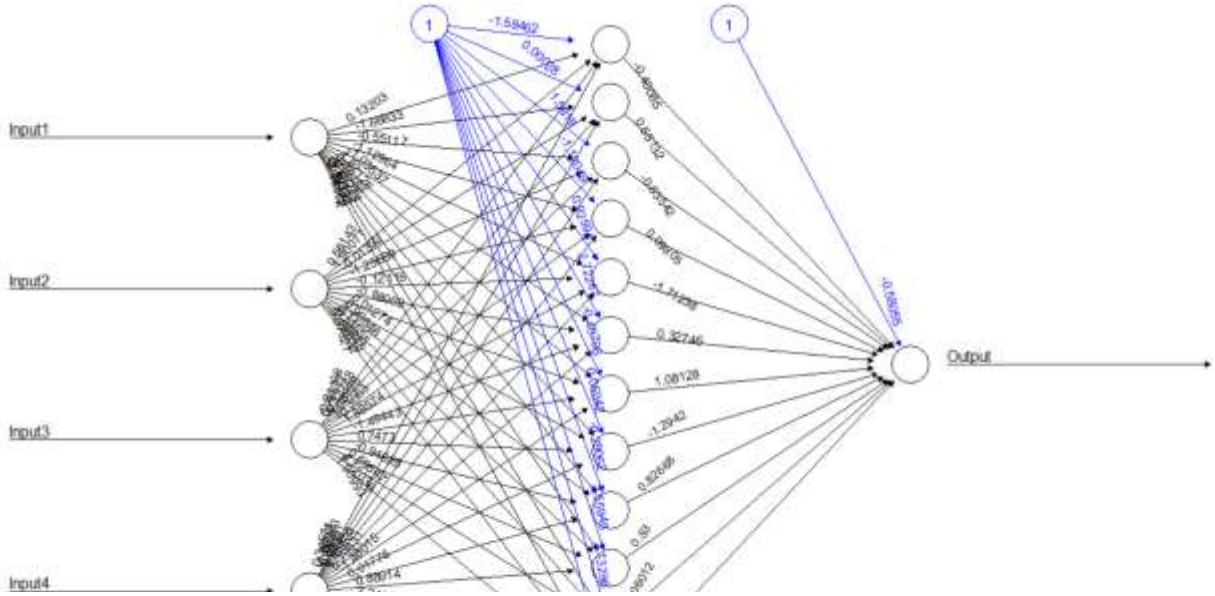


Figure 54:code block for model 1

Figure 56:plot of model 1

Likewise, two further models (Model 2, 3) were created with four inputs, changing node counts,

Figure 55:Outputs of model 1

hidden layers, and linear/nonlinear output.

2. Model 4 (with 3 inputs)

```
#---------------4th MLP with 3 input variables and 1 hidden layer----------------------

# Train your neural network model
t3_nn_1 <- neuralnet(Output ~ Input1 + Input2 + Input3 ,
                     data = training_Type_t3,
                     hidden = 12,
                     linear.output = TRUE)

# Plot the model if desired
#plot(t3_nn_1)
# Use the model to make predictions on testing data
t3_nn_1_result <- predict(t3_nn_1, testing_Type_t3)
#t3_nn_1_result

t3_1_original <- unnormalize(t3_nn_1_result,min_t3_lagged_data,max_t3_lagged_data)
#t3_1_original
t3_1_original_rounded <-as.data.frame(sapply(t3_1_original,round,digits=2))
#t3_1_original_rounded

test_data_t3_1_result <- exchangeUSD_df[401:497,]

t3_1_output <- cbind(test_data_t3_1_result, t3_1_original_rounded)
colnames(t3_1_output) <- c("Actual","Predicted")
#head(t3_1_output)

library(Metrics)
RMSE_t3_nn_1 = rmse(t3_1_output$Actual, t3_1_output$Predicted)
MAE_t3_nn_1 = mae(t3_1_output$Actual, t3_1_output$Predicted)
MAPE_t3_nn_1 = mape(t3_1_output$Actual, t3_1_output$Predicted)
SMAPE_t3_nn_1 = smape(t3_1_output$Actual, t3_1_output$Predicted)
RMSE_t3_nn_1
MAE_t3_nn_1
MAPE_t3_nn_1
SMAPE_t3_nn_1
```

Figure 58:code block for model 4

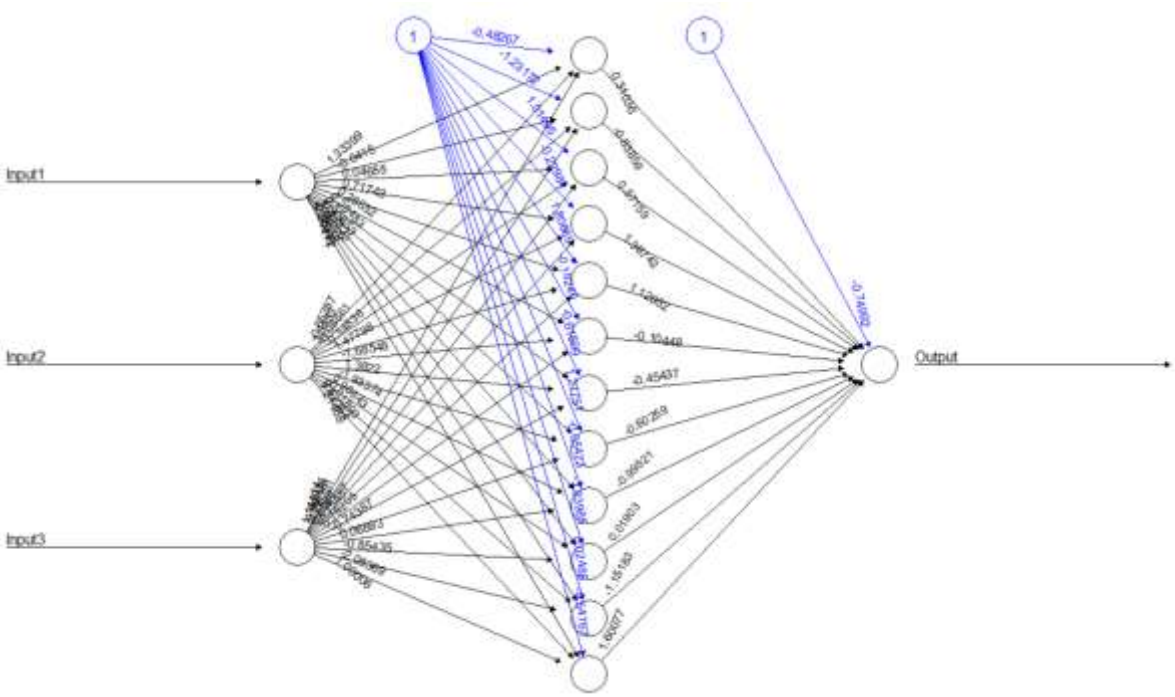
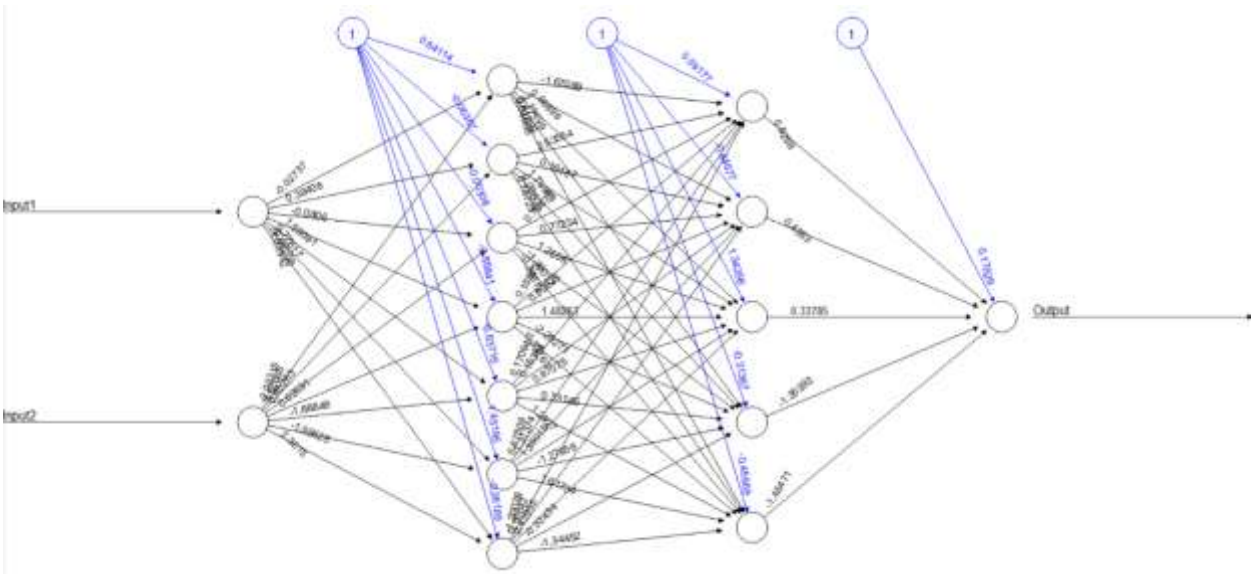


Figure 57:boxplot of model 4

3. Model 8 (with 2 inputs)

```
#==============8th MLP with 2 input variables and 2 hidden layer=======================

# Train your neural network model
t2_nn_2 <- neuralnet(Output ~ Input1 + Input2  ,
                     data = training_Type_t2,
                     hidden = c(7,5),
                     linear.output = TRUE)

# Plot the model if desired
#plot(t2_nn_2)
# Use the model to make predictions on testing data
t2_nn_2_result <- predict(t2_nn_2, testing_Type_t2)
#t2_nn_2_result

t2_2_original <- unnormalize(t2_nn_2_result,min_t2_lagged_data,max_t2_lagged_data)
#t4_1_original
t2_2_original_rounded <-as.data.frame(sapply(t2_2_original,round,digits=2))
#t4_1_original_rounded

test_data_t2_2_result <- exchangeUSD_df[401:498,]

t2_2_output <- cbind(test_data_t2_2_result,t2_2_original_rounded)
colnames(t2_2_output) = c("Actual","Predicted")
head(t2_2_output)

library(Metrics)
RMSE_t2_nn_2 = rmse(t2_2_output$Actual, t2_2_output$Predicted)
MAE_t2_nn_2 = mae(t2_2_output$Actual, t2_2_output$Predicted)
MAPE_t2_nn_2 = mape(t2_2_output$Actual, t2_2_output$Predicted)
SMAPE_t2_nn_2 = smape(t2_2_output$Actual, t2_2_output$Predicted)
RMSE_t2_nn_2
MAE_t2_nn_2
MAPE_t2_nn_2
SMAPE_t2_nn_2
```

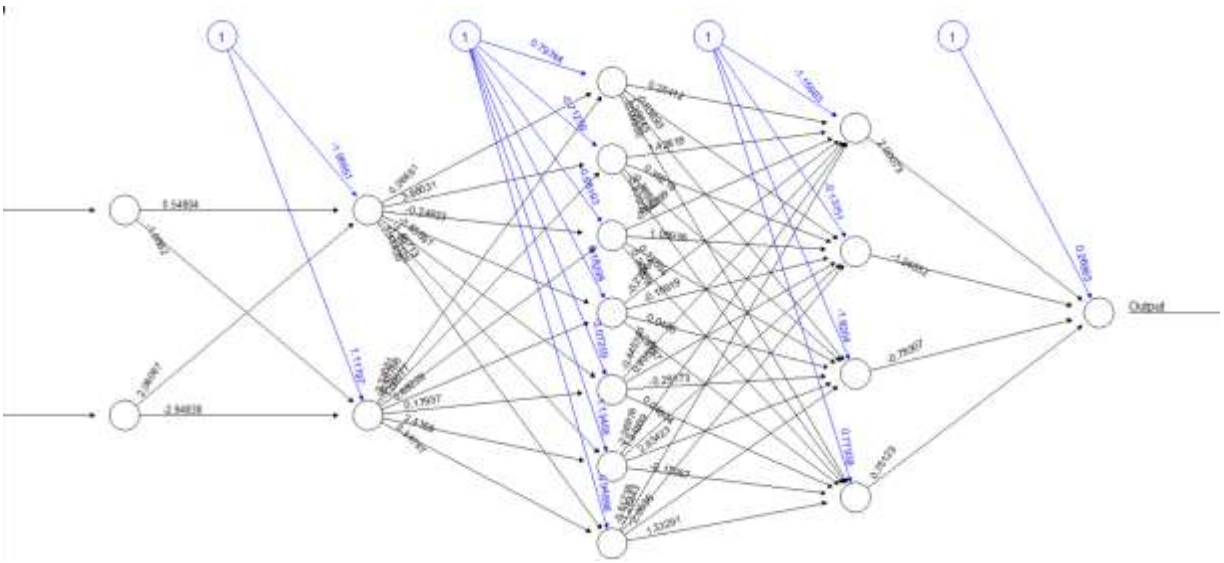
Figure 59:code block for model 8



4. Model 9 (with 1 input)

```r
#==============9th MLP with 2 input variables and 3 hidden layer=======================

# Train your neural network model
t2_nn_3 <- neuralnet(Output ~ Input1 + Input2  ,
                     data = training_Type_t2,
                     hidden = c(2,7,4),
                     linear.output = TRUE)

# Plot the model if desired
#plot(t2_nn_3)
# Use the model to make predictions on testing data
t2_nn_3_result <- predict(t2_nn_3, testing_Type_t2)
#t2_nn_3_result

t2_3_original <- unnormalize(t2_nn_3_result,min_t2_lagged_data,max_t2_lagged_data)
#t2_3_original
t2_3_original_rounded <-as.data.frame(sapply(t2_3_original,round,digits=2))
#t2_3_original_rounded

test_data_t2_3_result <- exchangeUSD_df[401:498,]

t2_3_output <- cbind(test_data_t2_3_result,t2_3_original_rounded )
colnames(t2_3_output) = c("Actual","Predicted")
#head(t2_3_output)

library(Metrics)
RMSE_t2_nn_3 = rmse(t2_3_output$Actual, t2_3_output$Predicted)
MAE_t2_nn_3 = mae(t2_3_output$Actual, t2_3_output$Predicted)
MAPE_t2_nn_3 = mape(t2_3_output$Actual, t2_3_output$Predicted)
SMAPE_t2_nn_3 = smape(t2_3_output$Actual, t2_3_output$Predicted)
RMSE_t2_nn_3
MAE_t2_nn_3
MAPE_t2_nn_3
SMAPE_t2_nn_3
```

Figure 60:code block of model 9



The remaining model outputs and code blocks are in Appendix B.

2.5 Understanding of Statistics Indices

1. **Root Mean Squared Error (RMSE):**

   While RMSE and MSE are comparable, RMSE is the square root of the average squared difference between the dataset's actual and projected values.

   - The average size of the errors between the anticipated and actual values is measured by RMSE.
   - The square root of the mean of the squared discrepancies between the expected and actual values is used to compute it.
   - RMSE is simple to read since it is expressed in the same units as the target variable.

2. **Mean Absolute Error (MAE)**

   The average absolute difference (MAE) between the dataset's true and expected values is measured. Because MAE fails to penalize huge mistakes with the same severity as MSE and RMSE, it is more resilient to outliers.

   - It is computed by averaging the percentage differences between the values that were anticipated and those that were observed.
   - Because it does not square the errors, MAE is less susceptible to outliers than RMSE.

3. **Mean Absolute Percentage Error (MAPE)**

   The average percentage difference between the dataset's real values and its expected ones is measured by MAPE. It's frequently applied in forecasting to evaluate the accuracy of forecasts' actual value size.

   - The mean of the absolute percentage deviations between the expected and actual numbers is computed.
   - Because MAPE is reported as a percentage, it is simple to compare and comprehend across various models and datasets.

4. **Symmetric Mean Absolute Percentage Error (SMAPE)**

SMAPE is an additional metric for evaluating forecast accuracy about actual value size. It is comparable to MAPE but symmetric that is, it weighs estimates equally—in both directions.

- A symmetric version of MAPE called SMAPE considers situations where both overestimation and underestimation are equally unacceptable.

2.6 comparison table of their testing performances

Model	No. of Inputs	No. of hidden layers	No. of neurons	Linear Output	Structure Description
1	4	1	12	FALSE	4 input, a single hidden layer with 12 neurons
2		2	(5,9)	TRUE	4 inputs,2 hidden layers with 5 and 9 neurons
3		1	8	TRUE	4 input,a single hidden layer with 8 neurons
4	3	1	12	TRUE	3 input, a single hidden layer with 12 neurons
5		2	(8,3)	FALSE	3 inputs,2 hidden layers with 8 and 3 neurons
6		2	(2,6)	TRUE	3 inputs,2hidden layers with 2 and 6 neurons
7	2	1	10	FALSE	2 input, a single hidden layer with 10 neurons
8		2	(7,5)	TRUE	2 inputs, 2 hidden layers with 7 and 5 neurons
9		3	(2,7,4)	TRUE	2 inputs, 3 hidden layers with 2,7 and 4 neurons
10		1	15	FALSE	1 input, a single hidden layer with 5 neurons

11	1	1	4	FALSE	1 input, a single hidden layer with 4 neurons
12		3	(9,3,5)	TRUE	1 input,3 hidden layers with 9,3 and 5 neurons

```
> comparison_table
      Model         RMSE          MAE         MAPE        SMAPE
1   Model 1 0.010583163 0.002592929 0.006225731 0.002188463
2   Model 2 0.010436475 0.002910101 0.006167662 0.002200879
3   Model 3 0.010471350 0.002889899 0.006169029 0.001961350
4   Model 4 0.008238212 0.005006122 0.004636459 0.003448100
5   Model 5 0.009200633 0.004538776 0.005250684 0.003439395
6   Model 6 0.008472642 0.004553061 0.004636459 0.003790523
7   Model 7 0.005793047 0.006219588 0.003451305 0.004636823
8   Model 8 0.005868299 0.006926804 0.003439145 0.005249024
9   Model 9 0.006561188 0.006116495 0.003451305 0.004715234
10 Model 10 0.003408886 0.008129167 0.002188729 0.006218464
11 Model 11 0.003438390 0.008129167 0.002200165 0.006166229
12 Model 12 0.002941226 0.008202083 0.001961115 0.006218464
```

Figure 61:output of comparison table

2.7 Finding best MLP model

1. Best one-hidden layer network:

- One-hidden layer networks: model 1, model 3, model 4, model 7, model 10, model 11

Higher prediction accuracy is shown by lower RMSE, MAE, MAPE and SMAPE values, which show a more substantial alignment between expected and actual values.

We get model 10 as the best one-hidden layer network with lower RMSE, MAE, MAPE and SMAPE values from all the One-hidden layer networks.

```
> RMSE_t1_nn_1 = rmse(t1_1_output$Actual, t1_1_output$Predicted)
> MAE_t1_nn_1 = mae(t1_1_output$Actual, t1_1_output$Predicted)
> MAPE_t1_nn_1 = mape(t1_1_output$Actual, t1_1_output$Predicted)
> SMAPE_t1_nn_1 = smape(t1_1_output$Actual, t1_1_output$Predicted)
> RMSE_t1_nn_1
[1] 0.003441326
> MAE_t1_nn_1
[1] 0.002912121
> MAPE_t1_nn_1
[1] 0.002204105
> SMAPE_t1_nn_1
[1] 0.002203518
```

Figure 62:output values of RMSE,MAE,MAPE,SMAPE of model 10

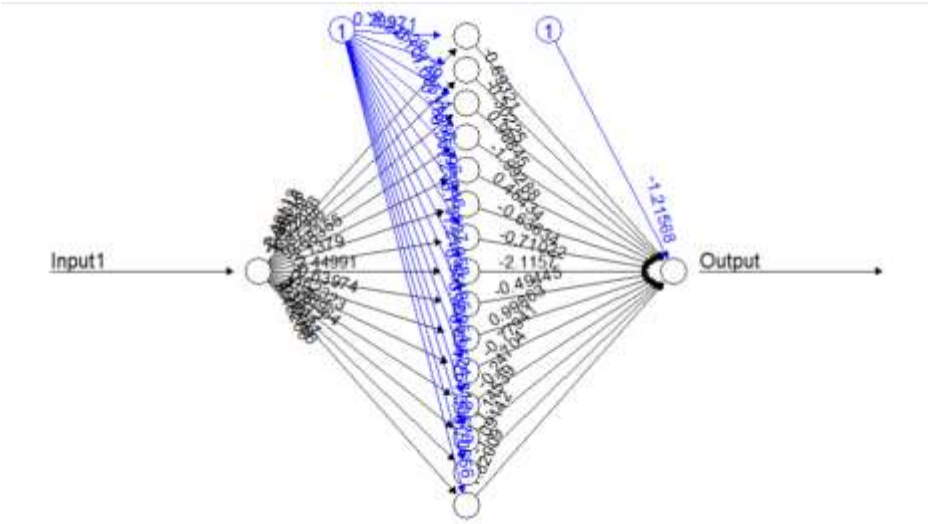


Figure 63:plot of model 10

2. Best two-hidden layer network:

As we get the best one-hidden layer network, we also get the model with the least values of RMSE, MAE, MAPE and SMAPE to find the best two-hidden layer network.

Model 8 is the best two-hidden layer network with lower RMSE, MAE, MAPE and SMAPE values than all the One-hidden layer networks.

```
> RMSE_t2_nn_2 = rmse(t2_2_output$Actual, t2_2_output$Predicted)
> MAE_t2_nn_2 = mae(t2_2_output$Actual, t2_2_output$Predicted)
> MAPE_t2_nn_2 = mape(t2_2_output$Actual, t2_2_output$Predicted)
> SMAPE_t2_nn_2 = smape(t2_2_output$Actual, t2_2_output$Predicted)
> RMSE_t2_nn_2
[1] 0.005871776
> MAE_t2_nn_2
[1] 0.004542857
> MAPE_t2_nn_2
[1] 0.003442162
> SMAPE_t2_nn_2
[1] 0.003442847
```

Figure 65:output values of RMSE,MAE,MAPE,SMAPE of model 8

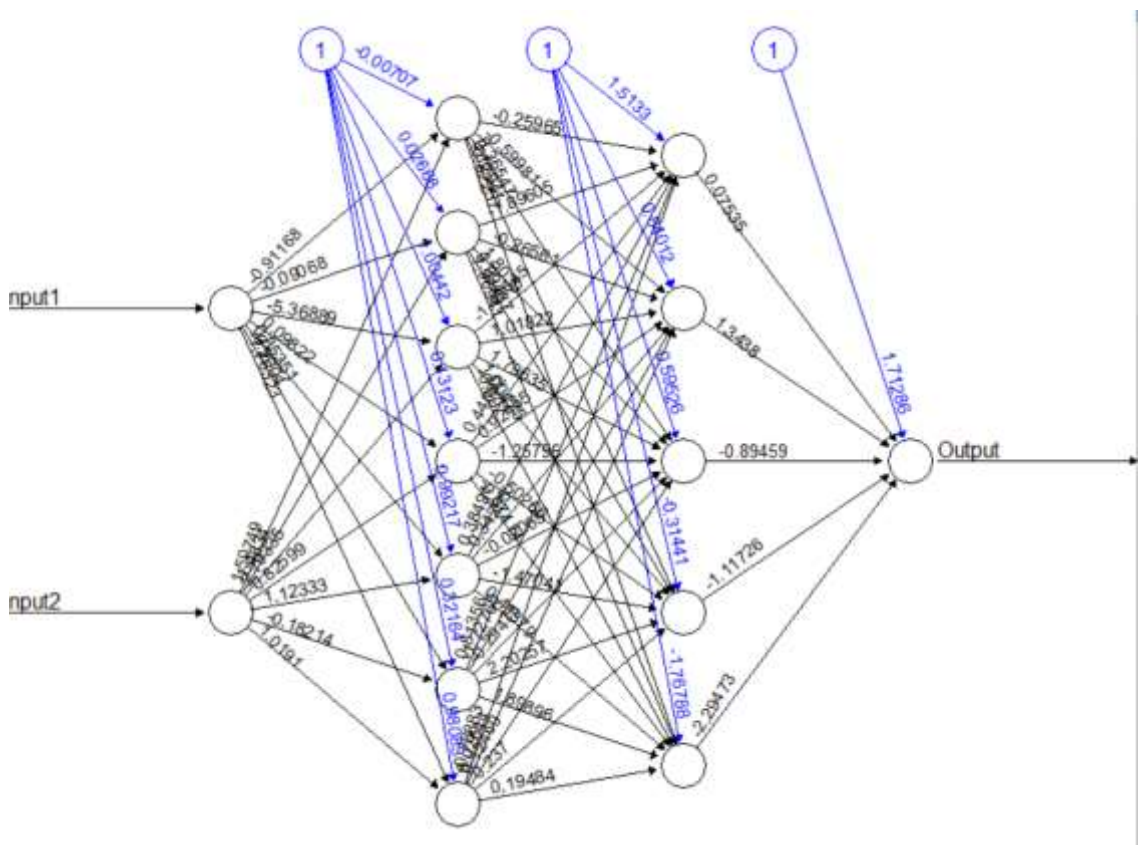


Figure 64:plot of model 8

To get the best MLP,

Step 1: calculate one hidden layer network parameters.

- Model 10
    - Weights between the input layer and 1st hidden layer:
      1 inputs*15 nodes = 15 weights ,
    - Biases of first hidden layer: 15 nodes = 15 biases,
    - Weights between the hidden layer and output layer: 15*1 node = 15 weights
    - Biases of output layer: 1 bias.
    - Total number of parameters (weights and biases): 46 parameters

Step 2: calculate two hidden layer network parameters.

- Model 8
    - Weights between the input layer and 1st hidden layer:
      1. 2 inputs*7 nodes = 14 weights,
    - Biases of first hidden layer: 7 nodes = 7 biases,
    - Weights between the hidden layer and 2nd hidden layer: 7*5 node = 35 weights
    - Biases of second hidden layer: 5 nodes = 5 biases,
    - Weights between the 2nd hidden layer and output layer: 5*1 node = 5 weights
    - Biases of output layer: 1 bias.
    - Total number of parameters (weights and biases): 67 parameters

In general, models with fewer parameters have a higher probability of being practical and less probable to overfit, mainly when used to smaller datasets or more straightforward issues. However, larger datasets and appropriate regularization approaches are needed to minimize overfitting in models with more parameters, such as Model 1 (73 parameters) and Model 8 (67 parameters), which have a more substantial potential to learn more complicated patterns in the data.

The comparison demonstrates that **Model 10,** which has lower number of parameters (46 parameters), is the most efficient network in terms of simplicity and efficiency. However, because of its small size, its performance could be restricted due to more complicated issues.

2.8 (best MLP)Model 10 in graphically

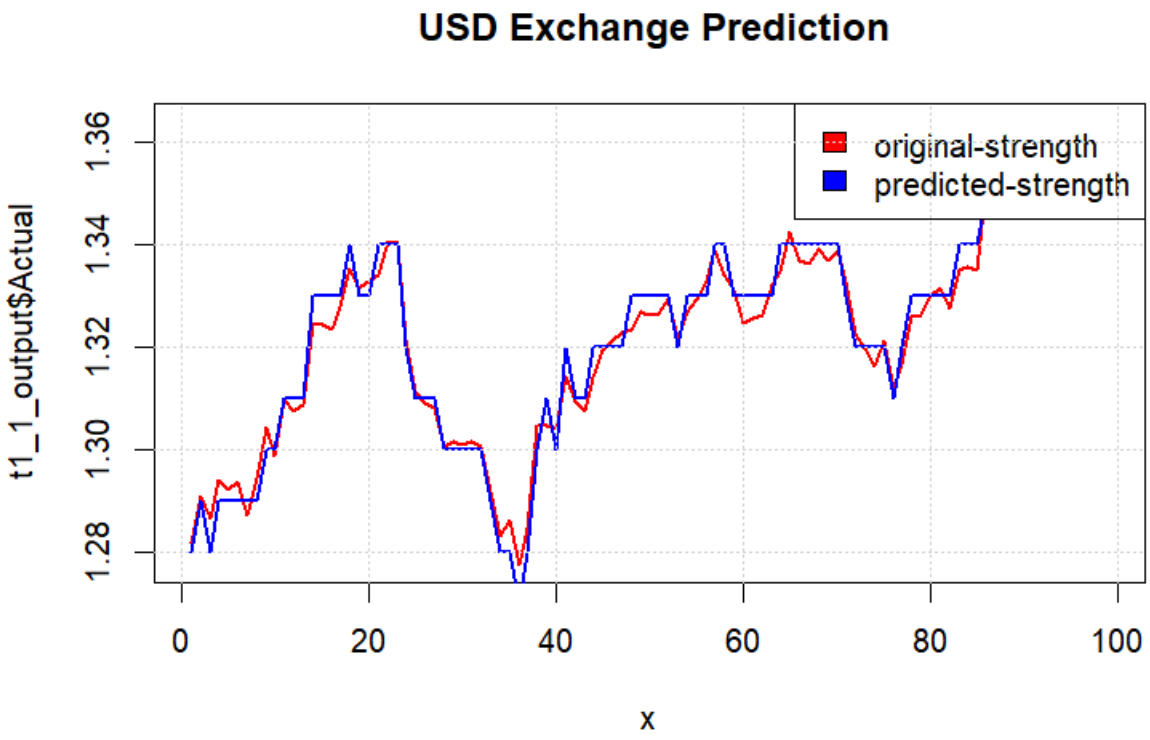
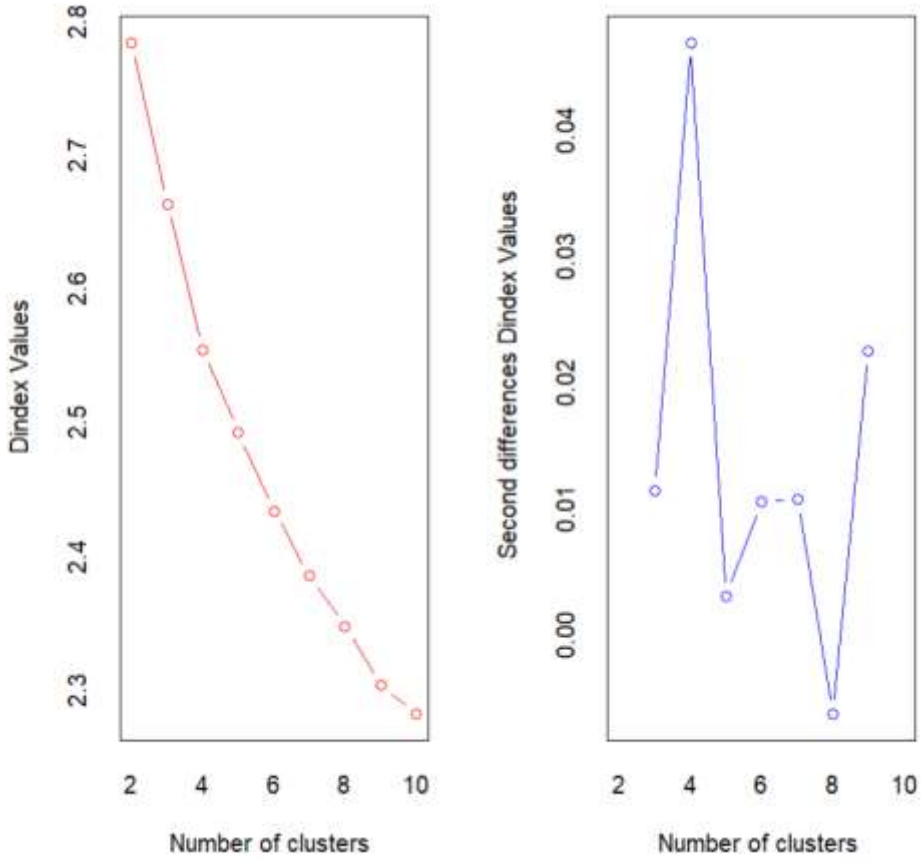


Figure 66:model 10 in graphically

Based on the simple line chart, Overall trends and variations appear to be quite well captured by the model, with the projected values nearly matching the original values. There are certain variations, nevertheless, especially when correctly expressing the peak magnitudes. The model seems to be able to anticipate USD exchange rate fluctuations rather well, making it a potentially helpful tool for study and forecasting in the currency markets.

Appendix

Appendix A.1



Appendix A.2-Code for first subtask of part A

```
#Part1 subtask1_w1953836

#install the packages
library(readxl)
#import the dataset
dataSet <- read_excel("Whitewine_v6.xlsx")
summary(data)
boxplot(dataSet)
#Remove 12th column from data set(Output column)
data_excluded_clomn12 <- dataSet[, -12]
#Function for find outliers using IQR
find_outliers <- function(x) {
  q1 <- quantile(x, 0.25)
  q3 <- quantile(x, 0.75)
  iqr <- q3 - q1
  lower_bound <- q1 - 1.5 * iqr
  upper_bound <- q3 + 1.5 * iqr
  outliers <- x < lower_bound | x > upper_bound
  return(outliers)
}
#_w1953836_use find_outliers function to each columns of the data set
outlier_index <- apply(data_excluded_clomn12, 2, find_outliers)

outlier_rows <- rowSums(outlier_index) > 0
#_w1953836_scale the data set that cleaned
cleaned_data_step1 <- data_excluded_clomn12[!outlier_rows, ]

scaled_cleaned_data_step1 <- as.data.frame(scale(cleaned_data_step1))
#_w1953836_check the boxplot in step1
boxplot (scaled_cleaned_data_step1)
```

```
#_w1953836_Apply the outliers find function to the 2,3,4,6,9 column of the
scaled_cleaned_data
outlier_column_2 <- find_outliers(scaled_cleaned_data_step1[[2]])

outlier_column_3 <- find_outliers(scaled_cleaned_data_step1[[3]])

outlier_column_4 <- find_outliers(scaled_cleaned_data_step1[[4]])

outlier_column_6 <- find_outliers(scaled_cleaned_data_step1[[6]])

outlier_column_9 <- find_outliers(scaled_cleaned_data_step1[[9]])
#_w1953836_Remove outliers from the scaled_cleaned_data dataframe for the
specified columns
scaled_cleaned_data_step2 <- scaled_cleaned_data_step1[
  !outlier_column_2 &
    !outlier_column_3 &
    !outlier_column_4 &
    !outlier_column_6 &
    !outlier_column_9,
]
#_w1953836_Apply the outliers detection function to the 3rd column of the
cleaned_scaled_data
outlier_column_3_again <- find_outliers(scaled_cleaned_data_step2[[3]])
#_w1953836_Remove outliers from the cleaned_scaled_data dataframe for the third
column
cleaned_scaled_data_last <- scaled_cleaned_data_step2[!outlier_column_3_again, ]

#_w1953836_Create a boxplot to visualize outliers for the specified columns after
removal
boxplot(cleaned_scaled_data_last)
head(cleaned_scaled_data_last)

#task2_w1953836

#_w1953836_install the " NbClust" package [ NBclust Method]
```

```
library(NbClust)
#_w1953836_Number of cluster centers
#_w1953836_Distance= euclidean
clusterNo=NbClust(cleaned_scaled_data_last,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")
#_w1953836_Distance= manhattan
clusterNo=NbClust(cleaned_scaled_data_last, distance="manhattan",
min.nc=2,max.nc=10,method="kmeans",index="all")
#_w1953836_Distance= maximum
clusterNo=NbClust(cleaned_scaled_data_last, distance="maximum",
min.nc=2,max.nc=10,method="kmeans",index="all")

#_w1953836_Elbow method
x= cleaned_scaled_data_last
y<- data$quality
library(factoextra)
fviz_nbclust(x, kmeans, method = 'wss')
library(factoextra)
#_w1953836_Gap Statistic Algorithm
fviz_nbclust(x, kmeans, method = 'gap_stat')
library(factoextra)
#_w1953836_Average Silhouette Method
fviz_nbclust(x, kmeans, method = 'silhouette')

#_w1953836_task 3
x=cleaned_scaled_data_last
y=data$quality
kc <- kmeans(x,2)
#_w1953836_wss1 and wss2
wss = kc$withinss
wss
#_w1953836_total of wss
wss=kc$tot.withinss
wss
#_w1953836_bss
```

```
bss = kc$betweenss
#_w1953836_TSS
TSS = kc$tot.withinss + kc$betweenss
TSS

#_w1953836_task 4
k=2
#_w1953836_Assuming 'x' is your cleaned and scaled data
library(cluster)
#_w1953836_Perform kmeans clustering with k=2
kmeans_data <- kmeans(x, centers = 2, nstart = 10)
#_w1953836_Calculate silhouette widths
sil_width <- silhouette(kmeans_data$cluster, dist(x))
sil <- silhouette(kmeans_data$cluster, dist(x))
fviz_silhouette(sil)

avg_silhouette_width <- mean(sil[,"sil_width"])
cat("average silhouette width score: ",avg_silhouette_width )
```

Appendix A.3 -Code for the Second subtask of part A

```
#Part1 subtask2_w1953836_ # PCA


#_w1953836_install the packages
```

```
library(readxl)
#_w1953836_import the dataset
dataSet <- read_excel("Whitewine_v6.xlsx")
#_w1953836_Remove 12th column from data set(Output column)
data_excluded_clomn12 <- dataSet[, -12]
#_w1953836_Function for find outliers using IQR
find_outliers <- function(x) {
  q1 <- quantile(x, 0.25)
  q3 <- quantile(x, 0.75)
  iqr <- q3 - q1
  lower_bound <- q1 - 1.5 * iqr
  upper_bound <- q3 + 1.5 * iqr
  outliers <- x < lower_bound | x > upper_bound
  return(outliers)
}
#_w1953836_use “find_outliers” function to each columns of the data set
outlier_index <- apply(data_excluded_clomn12, 2, find_outliers)

outlier_rows <- rowSums(outlier_index) > 0
#_w1953836_scale the data set that cleaned
cleaned_data_step1 <- data_excluded_clomn12[!outlier_rows, ]

scaled_cleaned_data_step1 <- as.data.frame(scale(cleaned_data_step1))
#_w1953836_check the boxplot in step1
boxplot (scaled_cleaned_data_step1)
#_w1953836_Apply the outliers find function to the 2,3,4,6,9 column of the
scaled_cleaned_data
outlier_column_2 <- find_outliers(scaled_cleaned_data_step1[[2]])

outlier_column_3 <- find_outliers(scaled_cleaned_data_step1[[3]])

outlier_column_4 <- find_outliers(scaled_cleaned_data_step1[[4]])

outlier_column_6 <- find_outliers(scaled_cleaned_data_step1[[6]])
```

```
outlier_column_9 <- find_outliers(scaled_cleaned_data_step1[[9]])
#_w1953836_Remove outliers from the scaled_cleaned_data dataframe for the
specified columns
scaled_cleaned_data_step2 <- scaled_cleaned_data_step1[
  !outlier_column_2 &
    !outlier_column_3 &
    !outlier_column_4 &
    !outlier_column_6 &
    !outlier_column_9,
]
#_w1953836_Apply the outliers detection function to the 3rd column of the
cleaned_scaled_data
outlier_column_3_again <- find_outliers(scaled_cleaned_data_step2[[3]])
#_w1953836_Remove outliers from the cleaned_scaled_data dataframe for the third
column
cleaned_scaled_data_last <- scaled_cleaned_data_step2[!outlier_column_3_again, ]

#_w1953836_Create a boxplot to visualize outliers for the specified columns after
removal
boxplot(cleaned_scaled_data_last)

#_w1953836_Compute the Covariance Matrix
wine_cov <- cov(cleaned_scaled_data_last)
wine_cov

#_w1953836_Compute the Eigenvalues and eigenvectors
wine_eigen <- eigen(wine_cov)
str(wine_eigen)

#_w1953836_Access the Eigenvalues
wine_eigen$values
#To access eigenvalues and eigenVectors separately
wine_eigen$vectors

#_w1953836_The Proportion of Variance Explained
```

```
PVE <- wine_eigen$values / sum(wine_eigen$values)
round(PVE,2)

#_w1953836_choose 7 components
components <- 7
(wine_matrix <- wine_eigen$vectors[,1:components])
#_w1953836_Assign row and column names to the dataframe
phi <-- as.data.frame(wine_matrix)
row_nam <- c("fixed.acidity", "volatile.acidity", "citric.acid",
"residual.sugar", "chlorides", "free.sulfur.dioxide", "total.sulfur.dioxide",
"density", "pH", "sulphates", "alcohol")
col_nam <- c("PC1", "PC2", "PC3", "PC4", "PC5", "PC6", "PC7")

rownames(phi) <- row_nam
colnames(phi) <- col_nam

print(phi)

#_w1953836_Calculate Principal Components scores
x=cleaned_scaled_data_last
PC1 <- as.matrix(x) %*% phi[,1]
PC2 <- as.matrix(x) %*% phi[,2]
PC3 <- as.matrix(x) %*% phi[,3]
PC4 <- as.matrix(x) %*% phi[,4]
PC5 <- as.matrix(x) %*% phi[,5]
PC6 <- as.matrix(x) %*% phi[,6]
PC7 <- as.matrix(x) %*% phi[,7]

#_w1953836_Create data frame with Principal Components scores
PC <- data.frame(State = rownames(cleaned_scaled_data_last), PC1,
PC2,PC3,PC4,PC5,PC6,PC7)
#_w1953836_Assuming your data frame is named 'PC'
#_w1953836_Reset row names as a regular column
PC$State <- rownames(PC)
```

```r
#_w1953836_Remove the row names
rownames(PC) <- NULL

# Now you can remove the State column
PC_without_state <- subset(PC, select = -State)

head(PC_without_state)

#_w1953836_Plot Principal Components for each State
library(ggplot2)
ggplot(PC_without_state, aes(PC1,PC2,PC3,PC4,PC5,PC6,PC7)) +
  modelr::geom_ref_line(h = 0) +
  modelr::geom_ref_line(v = 0) +
  geom_text(aes(label = rownames(PC_without_state)), size = 3) +
  xlab("Principal Component 1") +
  ylab("Principal Component 2") +
  ggtitle("First 2 Principal Components of Wine Data")

#_w1953836_PVE plot
PVEplot <- qplot(1:11, PVE ) + geom_line() +
  xlab("Principal Component") +
  ylab("PVE") +
  ggtitle("Scree Plot") +
  ylim(0,1)

cumPVE <- qplot(c(1:11), cumsum(PVE)) +
  geom_line() +
  xlab("Principal Component") +
  ylab(NULL) +
  ggtitle("Cumulative Scree Plot") +
  ylim(0,1)
library(gridExtra)
grid.arrange(PVEplot, cumPVE, ncol = 8)

# _w1953836_e_part-Automated Tools on PCA-based Data set
```

```
library(NbClust)
#Distance= euclidean
#clusterNo=NbClust(cleaned_scaled_data_last,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")
#Distance= manhattan
#clusterNo=NbClust(cleaned_scaled_data_last, distance="manhattan",
min.nc=2,max.nc=10,method="kmeans",index="all")

#_w1953836_Elbow method
x= PC_without_state
y<- data$quality

library(factoextra)
fviz_nbclust(x, kmeans, method = 'wss')
library(factoextra)
#_w1953836_Gap Statistic Algorithm
fviz_nbclust(x, kmeans, method = 'gap_stat')
library(factoextra)
#_w1953836_Average Silhouette Method
fviz_nbclust(x, kmeans, method = 'silhouette')

#kmeans
x=PC_without_state

#_w1953836_cluster sizes
library(dplyr)
cluster_sizes <- cluster_assignments_simple %>%
  as.factor() %>%
  table()
cluster_sizes_df <- data.frame(cluster = names(cluster_sizes),
                               size = as.vector(cluster_sizes))
cluster_sizes_df
#_w1953836_K-means clustering with 2 clusters of sizes 1273, 886
kc2 <- kmeans(x,2)
kc2
```

```
#_w1953836_wss1 and wss2
wss = kc2$withinss #7024.343 10657.264

#_w1953836_total of wss
wss_tot=kc2$tot.withinss #17681.61

#_w1953836_bss
bss = kc2$betweenss # 5458.482

#_w1953836_task h

k=2
#_w1953836_Assuming 'x' is your cleaned and scaled data
library(cluster)
library(factoextra)
x=PC_without_state
sil_width <- silhouette(kc2$cluster, dist(x))
#_w1953836_Calculate silhouette widths
sil <- silhouette(kc2$cluster, dist(x))
fviz_silhouette(sil)

avg_silhouette_width <- mean(sil[,"sil_width"])
cat("average silhouette width score: ",avg_silhouette_width,"\n")


#_w1953836_last task

#_w1953836_Assuming 'x' is your cleaned and scaled data
library(fpc)

#_w1953836_calculate the  Calinski-Harabasz Index
ch_index_check <- calinhara(x,kc2$cluster,2)

#_w1953836_find the Calinski-Harabasz index value
```

```
cat("Calinski-Harabasz Index:", ch_index_check, )
```

Appendix B.1- code for part B

```
library(readxl)
library(neuralnet)
library(grid)
library(dplyr)
library(MASS)

exchangeUSD_data <- read_excel("ExchangeUSD.xlsx")
str(exchangeUSD_data)

#_w1953836_to scale every feature in the dataset
normalize=function(x){
  return((x-min(x))/(max(x)-min(x)))
}
unnormalize = function(x, min, max) {
  return( (max - min)*x + min )
}

#_w1953836_Extract the "USD/EUR" column
exchangeUSD_df <- exchangeUSD_data[,3]
#exchangeUSD_df
#-------------------------------------------------------------------------------
---------
#_w1953836_create matrix for (t-4)
t4_lagged_data <- data.frame(
  Input4 = lag(exchangeUSD_df, 4),
  Input3 = lag(exchangeUSD_df, 3),
  Input2 = lag(exchangeUSD_df, 2),
  Input1 = lag(exchangeUSD_df, 1),
```

```
  Predicted_output = exchangeUSD_df
)

#_w1953836_remove NA value rows
t4_lagged_data <- t4_lagged_data[complete.cases(t4_lagged_data),]
colnames(t4_lagged_data) = c("Input1","Input2","Input3","Input4","Output")
#head(t4_lagged_data)
#t4_lagged_data$Input1

#_w1953836_minimum and maximum value of t4_lagged_data
min_t4_lagged_data <- min(t4_lagged_data)
#min_t4_lagged_data
max_t4_lagged_data <- max(t4_lagged_data)
#max_t4_lagged_data

#_w1953836_normalizes t4_lagged_data
normalized_t4_lagged_data <- as.data.frame(lapply(t4_lagged_data,normalize))
#normalized_t4_lagged_data

#_w1953836_this is the desired output of the training dataset
training_Type_t4 = normalized_t4_lagged_data[1:400,]
testing_Type_t4 = normalized_t4_lagged_data[401:nrow(normalized_t4_lagged_data),]
testing_Type_t4

#_w1953836_i/o mtrix
io_matrix_data_t4 <- rbind(training_Type_t4, testing_Type_t4)
io_matrix_t4 <- io_matrix_data[, c("Input1", "Input2", "Input3", "Input4",
"Output")]
#io_matrix_t4

#_w1953836_io_matrix_t4
unnormalized_io_matrix_t4 <- data.frame(
  Input1 = unnormalize(io_matrix_t4$Input1, min_t4_lagged_data,
max_t4_lagged_data),
```

```
  Input2 = unnormalize(io_matrix_t4$Input2, min_t4_lagged_data,
max_t4_lagged_data),
  Input3 = unnormalize(io_matrix_t4$Input3, min_t4_lagged_data,
max_t4_lagged_data),
  Input4 = unnormalize(io_matrix_t4$Input4, min_t4_lagged_data,
max_t4_lagged_data),
  Output = unnormalize(io_matrix_t4$Output, min_t4_lagged_data,
max_t4_lagged_data)
)
#_w1953836_Print the unnormalized I/O matrix
print(unnormalized_io_matrix_t4)


#==============1st MLP with 4 input variables and 1 hidden
layer=======================

#_w1953836_Train your neural network model
t4_nn_1 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4,
                     data = training_Type_t4,
                     hidden = 12,
                     linear.output = FALSE)

#_w1953836_Plot the model if desired
#plot(t4_nn_1)
#_w1953836_Use the model to make predictions on testing data
t4_nn_1_result <- predict(t4_nn_1, testing_Type_t4)
#_w1953836_t4_nn_1_result

t4_1_original <-
unnormalize(t4_nn_1_result,min_t4_lagged_data,max_t4_lagged_data)
#t4_1_original
t4_1_original_rounded <-as.data.frame(sapply(t4_1_original,round,digits=2))
#t4_1_original_rounded

test_data_t4_1_result <- exchangeUSD_df[401:496,]
```

```
t4_1_output <- cbind(test_data_t4_1_result,t4_1_original_rounded )
colnames(t4_1_output) = c("Actual","Predicted")
head(t4_1_output)

library(Metrics)
RMSE_t4_nn_1 = rmse(t4_1_output$Actual, t4_1_output$Predicted)
MAE_t4_nn_1 = mae(t4_1_output$Actual, t4_1_output$Predicted)
MAPE_t4_nn_1 = mape(t4_1_output$Actual, t4_1_output$Predicted)
SMAPE_t4_nn_1= smape(t4_1_output$Actual, t4_1_output$Predicted)
RMSE_t4_nn_1
MAE_t4_nn_1
MAPE_t4_nn_1
SMAPE_t4_nn_1


#===========2nd MLP with 4 input variables and 2 hidden
layer==========================

#_w1953836_Train neural network model
t4_nn_2 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4,
                     data = training_Type_t4,
                     hidden = c(5,9),
                     linear.output = TRUE )

#_w1953836_Plot the model if desired
#plot(t4_nn_2)
#_w1953836_Use the model to make predictions on testing data
t4_nn_2_result <- predict(t4_nn_2, testing_Type_t4)
#t4_nn_2_result

t4_2_original <-
unnormalize(t4_nn_2_result,min_t4_lagged_data,max_t4_lagged_data)
#t4_1_original
t4_2_original_rounded <-as.data.frame(sapply(t4_2_original,round,digits=2))
```

```
#t4_1_original_rounded

test_data_t4_2_result <- exchangeUSD_df[401:496,]

t4_2_output <- cbind(test_data_t4_2_result,t4_2_original_rounded )
colnames(t4_2_output) = c("Actual","Predicted")
#head(t4_2_output)

library(Metrics)
RMSE_t4_nn_2 = rmse(t4_2_output$Actual, t4_2_output$Predicted)
MAE_t4_nn_2 = mae(t4_2_output$Actual, t4_2_output$Predicted)
MAPE_t4_nn_2 = mape(t4_2_output$Actual, t4_2_output$Predicted)
SMAPE_t4_nn_2 = smape(t4_2_output$Actual, t4_2_output$Predicted)
RMSE_t4_nn_2
MAE_t4_nn_2
MAPE_t4_nn_2
SMAPE_t4_nn_2


#===========3rd MLP with 4 input variables and 1 hidden
layer=========================

#_w1953836_Train neural network model
t4_nn_3 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4,
                     data = training_Type_t4,
                     hidden = 8,
                     linear.output = TRUE)

#_w1953836_Plot the model
#plot(t4_nn_3)
#_w1953836_Use the model to make predictions on testing data
t4_nn_3_result <- predict(t4_nn_3, testing_Type_t4)
#t4_nn_3_result
```

```
t4_3_original <-
unnormalize(t4_nn_3_result,min_t4_lagged_data,max_t4_lagged_data)
#t4_1_original
t4_3_original_rounded <-as.data.frame(sapply(t4_3_original,round,digits=2))
#t4_1_original_rounded

test_data_t4_3_result <- exchangeUSD_df[401:496,]

t4_3_output <- cbind(test_data_t4_3_result,t4_3_original_rounded )
colnames(t4_3_output) = c("Actual","Predicted")
#head(t4_3_output)

library(Metrics)
RMSE_t4_nn_3 = rmse(t4_3_output$Actual, t4_3_output$Predicted)
MAE_t4_nn_3 = mae(t4_3_output$Actual, t4_3_output$Predicted)
MAPE_t4_nn_3 = mape(t4_3_output$Actual, t4_3_output$Predicted)
SMAPE_t4_nn_3 = smape(t4_3_output$Actual, t4_3_output$Predicted)
RMSE_t4_nn_3
MAE_t4_nn_3
MAPE_t4_nn_3
SMAPE_t4_nn_3

#==============================================================================
========
#_w1953836_create matrix for (t-3)

t3_lagged_data <- data.frame(
  Input3 = lag(exchangeUSD_df, 3),
  Input2 = lag(exchangeUSD_df, 2),
  Input1 = lag(exchangeUSD_df, 1),
  Predicted_output = exchangeUSD_df
)

#_w1953836_remove NA value rows
t3_lagged_data <- t3_lagged_data[complete.cases(t3_lagged_data),]
```

```
colnames(t3_lagged_data) = c("Input1","Input2","Input3","Output")
#head(t3_lagged_data)
#t3_lagged_data$Input1

#_w1953836_minimum and maximum value of t3_lagged_data
min_t3_lagged_data <- min(t3_lagged_data)
#min_t3_lagged_data
max_t3_lagged_data <- max(t3_lagged_data)
#max_t3_lagged_data

#_w1953836_normalizes t3_lagged_data
normalized_t3_lagged_data <- as.data.frame(lapply(t3_lagged_data,normalize))
#normalized_t3_lagged_data

#_w1953836_this is the desired output of the training dataset
training_Type_t3 = normalized_t3_lagged_data[1:400,]
testing_Type_t3 = normalized_t3_lagged_data[401:nrow(normalized_t3_lagged_data),]
#testing_Type_t3

#_w1953836_i/o mtrix
io_matrix_data_t3 <- rbind(training_Type_t3, testing_Type_t3)
io_matrix_t3 <- io_matrix_data[, c("Input1", "Input2", "Input3", "Input4",
"Output")]
#_w1953836_io_matrix_t3
unnormalized_io_matrix_t3 <- data.frame(
  Input1 = unnormalize(io_matrix_t3$Input1, min_t3_lagged_data,
max_t3_lagged_data),
  Input2 = unnormalize(io_matrix_t3$Input2, min_t3_lagged_data,
max_t3_lagged_data),
  Input3 = unnormalize(io_matrix_t3$Input3, min_t3_lagged_data,
max_t3_lagged_data),
  Output = unnormalize(io_matrix_t3$Output, min_t3_lagged_data,
max_t3_lagged_data)
)
```

```
#_w1953836_Print the unnormalized I/O matrix
print(unnormalized_io_matrix_t3)

#==============4th MLP with 3 input variables and 1 hidden
layer=======================

#_w1953836_Train neural network model
t3_nn_1 <- neuralnet(Output ~ Input1 + Input2 + Input3 ,
                     data = training_Type_t3,
                     hidden = 12,
                     linear.output = TRUE)

#_w1953836_Plot the model
plot(t3_nn_1)
#_w1953836_Use the model to make predictions on testing data
t3_nn_1_result <- predict(t3_nn_1, testing_Type_t3)
#t3_nn_1_result

t3_1_original <-
unnormalize(t3_nn_1_result,min_t3_lagged_data,max_t3_lagged_data)
#t3_1_original
t3_1_original_rounded <-as.data.frame(sapply(t3_1_original,round,digits=2))
#t3_1_original_rounded

test_data_t3_1_result <- exchangeUSD_df[401:497,]

t3_1_output <- cbind(test_data_t3_1_result, t3_1_original_rounded)
colnames(t3_1_output) <- c("Actual","Predicted")
#head(t3_1_output)

library(Metrics)
RMSE_t3_nn_1 = rmse(t3_1_output$Actual, t3_1_output$Predicted)
MAE_t3_nn_1 = mae(t3_1_output$Actual, t3_1_output$Predicted)
MAPE_t3_nn_1 = mape(t3_1_output$Actual, t3_1_output$Predicted)
SMAPE_t3_nn_1 = smape(t3_1_output$Actual, t3_1_output$Predicted)
```

```
RMSE_t3_nn_1
MAE_t3_nn_1
MAPE_t3_nn_1
SMAPE_t3_nn_1

#==============5th MLP with 3 input variables and 2 hidden
layer=======================

#_w1953836_Train neural network model
t3_nn_2 <- neuralnet(Output ~ Input1 + Input2 + Input3 ,
                     data = training_Type_t3,
                     hidden = c(8,3),
                     linear.output = FALSE)

#_w1953836_Plot the model if desired
plot(t3_nn_2)
#_w1953836_Use the model to make predictions on testing data
t3_nn_2_result <- predict(t3_nn_2, testing_Type_t3)
#t3_nn_2_result

t3_2_original <-
unnormalize(t3_nn_2_result,min_t3_lagged_data,max_t3_lagged_data)
#t3_2_original
t3_2_original_rounded <-as.data.frame(sapply(t3_2_original,round,digits=2))
#t3_2_original_rounded

test_data_t3_2_result <- exchangeUSD_df[401:497,]

t3_2_output = cbind(test_data_t3_2_result,t3_2_original_rounded)
colnames(t3_2_output) <- c("Actual","Predicted")
head(t3_2_output)

library(Metrics)
RMSE_t3_nn_2 = rmse(t3_2_output$Actual, t3_2_output$Predicted)
MAE_t3_nn_2 = mae(t3_2_output$Actual, t3_2_output$Predicted)
```

```
MAPE_t3_nn_2 = mape(t3_2_output$Actual, t3_2_output$Predicted)
SMAPE_t3_nn_2 = smape(t3_2_output$Actual, t3_2_output$Predicted)
RMSE_t3_nn_2
MAE_t3_nn_2
MAPE_t3_nn_2
SMAPE_t3_nn_2

#==============6th MLP with 3 input variables and 2 hidden
layer=======================

#_w1953836_Train your neural network model
t3_nn_3 <- neuralnet(Output ~ Input1 + Input2 + Input3 ,
                     data = training_Type_t3,
                     hidden = c(2,6),
                     linear.output = TRUE)

#_w1953836_Plot the model if desired
plot(t3_nn_3)
#_w1953836_Use the model to make predictions on testing data
t3_nn_3_result <- predict(t3_nn_3, testing_Type_t3)
t3_nn_3_result

t3_3_original <-
unnormalize(t3_nn_3_result,min_t3_lagged_data,max_t3_lagged_data)
#t4_1_original
t3_3_original_rounded <-as.data.frame(sapply(t3_3_original,round,digits=2))
#t4_1_original_rounded

test_data_t3_3_result <- exchangeUSD_df[401:497,]
#nrow(test_data_t3_1_result)
t3_3_output <- cbind(test_data_t3_3_result,t3_3_original_rounded )
colnames(t3_3_output) = c("Actual","Predicted")
head(t3_3_output)

library(Metrics)
```

```
RMSE_t3_nn_3 = rmse(t3_3_output$Actual, t3_3_output$Predicted)
MAE_t3_nn_3 = mae(t3_3_output$Actual, t3_3_output$Predicted)
MAPE_t3_nn_3 = mape(t3_3_output$Actual, t3_3_output$Predicted)
SMAPE_t3_nn_3 = smape(t3_3_output$Actual, t3_3_output$Predicted)
RMSE_t3_nn_3
MAE_t3_nn_3
MAPE_t3_nn_3
SMAPE_t3_nn_3

#==================================================================================
========
#_w1953836_create matrix for (t-2)

t2_lagged_data <- data.frame(
  Input2 = lag(exchangeUSD_df, 2),
  Input1 = lag(exchangeUSD_df, 1),
  Predicted_output = exchangeUSD_df
)

#_w1953836_remove NA value rows
t2_lagged_data <- t2_lagged_data[complete.cases(t2_lagged_data),]
colnames(t2_lagged_data) = c("Input1","Input2","Output")
#head(t2_lagged_data)
#t4_lagged_data$Input1

#_w1953836_minimum and maximum value of t4_lagged_data
min_t2_lagged_data <- min(t2_lagged_data)
#min_t4_lagged_data
max_t2_lagged_data <- max(t2_lagged_data)
#max_t4_lagged_data

#_w1953836_normalizes t4_lagged_data
normalized_t2_lagged_data <- as.data.frame(lapply(t2_lagged_data,normalize))
normalized_t2_lagged_data
```

```
#_w1953836_this is the desired output of the training dataset
training_Type_t2 = normalized_t2_lagged_data[1:400,]
testing_Type_t2 = normalized_t2_lagged_data[401:nrow(normalized_t2_lagged_data),]
#testing_Type_t2

#_w1953836_i/o mtrix
io_matrix_data_t2 <- rbind(training_Type_t2, testing_Type_t2)
io_matrix_t2 <- io_matrix_data[, c("Input1", "Input2", "Output")]

unnormalized_io_matrix_t2 <- data.frame(
  Input1 = unnormalize(io_matrix_t2$Input1, min_t2_lagged_data,
max_t2_lagged_data),
  Input2 = unnormalize(io_matrix_t2$Input2, min_t2_lagged_data,
max_t2_lagged_data),
  Output = unnormalize(io_matrix_t2$Output, min_t2_lagged_data,
max_t2_lagged_data)
)

#_w1953836_Print the unnormalized I/O matrix
print(unnormalized_io_matrix_t2)

#==============7th MLP with 2 input variables and 1 hidden
layer=======================

#_w1953836_Train neural network model
t2_nn_1 <- neuralnet(Output ~ Input1 + Input2  ,
                     data = training_Type_t2,
                     hidden = 10,
                     linear.output = FALSE)

#_w1953836_Plot the model if desired
plot(t2_nn_1)
#_w1953836_Use the model to make predictions on testing data
t2_nn_1_result <- predict(t2_nn_1, testing_Type_t2)
#t2_nn_1_result
```

```
t2_1_original <-
unnormalize(t2_nn_1_result,min_t2_lagged_data,max_t2_lagged_data)
#t4_1_original
t2_1_original_rounded <-as.data.frame(sapply(t2_1_original,round,digits=2))
t2_1_original_rounded

test_data_t2_1_result <- exchangeUSD_df[401:498,]

t2_1_output <- cbind(test_data_t2_1_result,t2_1_original_rounded )
colnames(t2_1_output) = c("Actual","Predicted")
#head(t2_1_output)

library(Metrics)
RMSE_t2_nn_1 = rmse(t2_1_output$Actual, t2_1_output$Predicted)
MAE_t2_nn_1 = mae(t2_1_output$Actual, t2_1_output$Predicted)
MAPE_t2_nn_1 = mape(t2_1_output$Actual, t2_1_output$Predicted)
SMAPE_t2_nn_1 = smape(t2_1_output$Actual, t2_1_output$Predicted)
RMSE_t2_nn_1
MAE_t2_nn_1
MAPE_t2_nn_1
SMAPE_t2_nn_1
#==============8th MLP with 2 input variables and 2 hidden
layer=======================

#_w1953836_Train neural network model
t2_nn_2 <- neuralnet(Output ~ Input1 + Input2  ,
                     data = training_Type_t2,
                     hidden = c(7,5),
                     linear.output = TRUE)

#_w1953836_Plot the model if desired
plot(t2_nn_2)
#_w1953836_Use the model to make predictions on testing data
t2_nn_2_result <- predict(t2_nn_2, testing_Type_t2)
```

```
#t2_nn_2_result

t2_2_original <-
unnormalize(t2_nn_2_result,min_t2_lagged_data,max_t2_lagged_data)
#t4_1_original
t2_2_original_rounded <-as.data.frame(sapply(t2_2_original,round,digits=2))
#t4_1_original_rounded

test_data_t2_2_result <- exchangeUSD_df[401:498,]

t2_2_output <- cbind(test_data_t2_2_result,t2_2_original_rounded)
colnames(t2_2_output) = c("Actual","Predicted")
head(t2_2_output)

library(Metrics)
RMSE_t2_nn_2 = rmse(t2_2_output$Actual, t2_2_output$Predicted)
MAE_t2_nn_2 = mae(t2_2_output$Actual, t2_2_output$Predicted)
MAPE_t2_nn_2 = mape(t2_2_output$Actual, t2_2_output$Predicted)
SMAPE_t2_nn_2 = smape(t2_2_output$Actual, t2_2_output$Predicted)
RMSE_t2_nn_2
MAE_t2_nn_2
MAPE_t2_nn_2
SMAPE_t2_nn_2

#==============9th MLP with 2 input variables and 3 hidden
layer=======================

#_w1953836_Train neural network model
t2_nn_3 <- neuralnet(Output ~ Input1 + Input2  ,
                     data = training_Type_t2,
                     hidden = c(2,7,4),
                     linear.output = TRUE)

#_w1953836_Plot the model if desired
plot(t2_nn_3)
```

```
#_w1953836_Use the model to make predictions on testing data
t2_nn_3_result <- predict(t2_nn_3, testing_Type_t2)
#t2_nn_3_result

t2_3_original <-
unnormalize(t2_nn_3_result,min_t2_lagged_data,max_t2_lagged_data)
#t2_3_original
t2_3_original_rounded <-as.data.frame(sapply(t2_3_original,round,digits=2))
#t2_3_original_rounded

test_data_t2_3_result <- exchangeUSD_df[401:498,]

t2_3_output <- cbind(test_data_t2_3_result,t2_3_original_rounded )
colnames(t2_3_output) = c("Actual","Predicted")
#head(t2_3_output)

library(Metrics)
RMSE_t2_nn_3 = rmse(t2_3_output$Actual, t2_3_output$Predicted)
MAE_t2_nn_3 = mae(t2_3_output$Actual, t2_3_output$Predicted)
MAPE_t2_nn_3 = mape(t2_3_output$Actual, t2_3_output$Predicted)
SMAPE_t2_nn_3 = smape(t2_3_output$Actual, t2_3_output$Predicted)
RMSE_t2_nn_3
MAE_t2_nn_3
MAPE_t2_nn_3
SMAPE_t2_nn_3


#==============================================================================
========
#_w1953836_create matrix for (t-1)

t1_lagged_data <- data.frame(
  Input1 = lag(exchangeUSD_df, 1),
  Predicted_output = exchangeUSD_df
)
```

```
#_w1953836_remove NA value rows
t1_lagged_data <- t1_lagged_data[complete.cases(t1_lagged_data),]
colnames(t1_lagged_data) = c("Input1","Output")
head(t1_lagged_data)
#t1_lagged_data$Input1

#_w1953836_minimum and maximum value of t4_lagged_data
min_t1_lagged_data <- min(t1_lagged_data)
#min_t1_lagged_data
max_t1_lagged_data <- max(t1_lagged_data)
#max_t1_lagged_data

#_w1953836_normalizes t1_lagged_data
normalized_t1_lagged_data <- as.data.frame(lapply(t1_lagged_data,normalize))
normalized_t1_lagged_data

#_w1953836_this is the desired output of the training dataset
training_Type_t1 = normalized_t1_lagged_data[1:400,]
testing_Type_t1 = normalized_t1_lagged_data[401:nrow(normalized_t1_lagged_data),]
#testing_Type_t1

io_matrix_data_t1 <- rbind(training_Type_t1, testing_Type_t1)
io_matrix_t1 <- io_matrix_data[, c("Input1", "Output")]
#_w1953836_Unnormalize the I/O matrix
unnormalized_io_matrix_t1 <- data.frame(
  Input1 = unnormalize(io_matrix_t1$Input1, min_t1_lagged_data,
max_t1_lagged_data),
  Output = unnormalize(io_matrix_t1$Output, min_t1_lagged_data,
max_t1_lagged_data)
)

#_w1953836_Print the unnormalized I/O matrix
print(unnormalized_io_matrix_t1)
```

```
#==============10th MLP with 1 input variables and 1 hidden
layer========================

#_w1953836_Train neural network model
t1_nn_1 <- neuralnet(Output ~ Input1 ,
                     data = training_Type_t1,
                     hidden = 15,
                     linear.output = FALSE)

#_w1953836_Plot the model if desired
plot(t1_nn_1)
#_w1953836_Use the model to make predictions on testing data
t1_nn_1_result <- predict(t1_nn_1, testing_Type_t1)
#t1_nn_1_result

t1_1_original <-
unnormalize(t1_nn_1_result,min_t1_lagged_data,max_t1_lagged_data)
#t2_1_original
t1_1_original_rounded <-as.data.frame(sapply(t1_1_original,round,digits=2))
#t1_1_original_rounded

test_data_t1_1_result <- exchangeUSD_df[401:499,]

t1_1_output <- cbind(test_data_t1_1_result,t1_1_original_rounded )
colnames(t1_1_output) = c("Actual","Predicted")
#head(t1_1_output)

library(Metrics)
RMSE_t1_nn_1 = rmse(t1_1_output$Actual, t1_1_output$Predicted)
MAE_t1_nn_1 = mae(t1_1_output$Actual, t1_1_output$Predicted)
MAPE_t1_nn_1 = mape(t1_1_output$Actual, t1_1_output$Predicted)
SMAPE_t1_nn_1 = smape(t1_1_output$Actual, t1_1_output$Predicted)
RMSE_t1_nn_1
MAE_t1_nn_1
MAPE_t1_nn_1
```

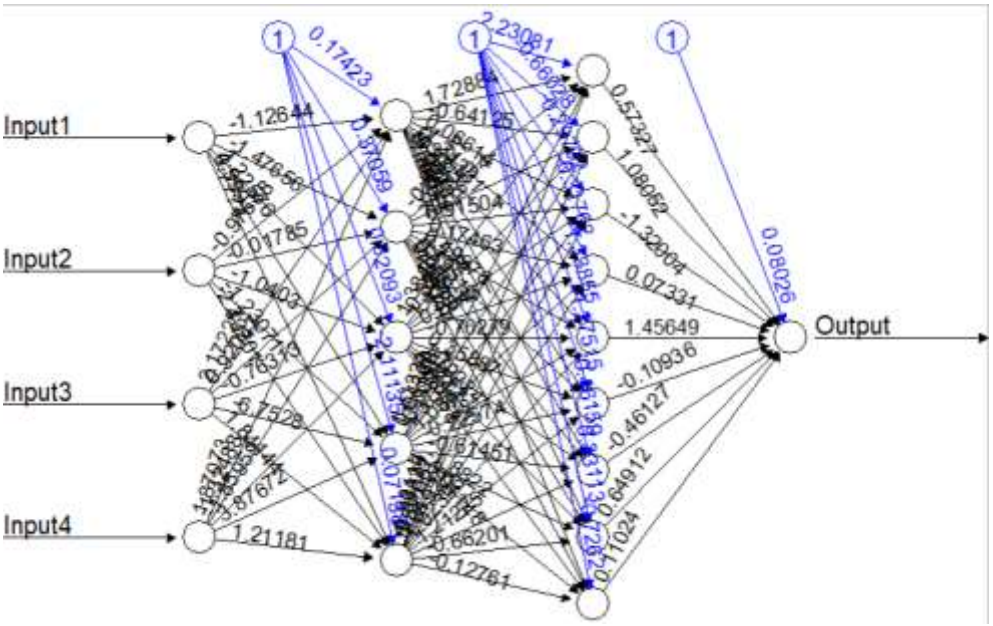
```
SMAPE_t1_nn_1


#==============11 th MLP with 1 input variables and 1 hidden
layer=======================

#_w1953836_Train neural network model
t1_nn_2 <- neuralnet(Output ~ Input1 ,
                     data = training_Type_t2 ,
                     hidden = 4,
                     linear.output = FALSE)

#_w1953836_Plot the model if desired
plot(t1_nn_2)
#_w1953836_Use the model to make predictions on testing data
t1_nn_2_result <- predict(t1_nn_2, testing_Type_t1)
t1_nn_2_result

t1_2_original <-
unnormalize(t1_nn_2_result,min_t1_lagged_data,max_t1_lagged_data)
#t1_1_original
t1_2_original_rounded <-as.data.frame(sapply(t1_2_original,round,digits=2))
#t1_1_original_rounded

test_data_t1_2_result <- exchangeUSD_df[401:499,]

t1_2_output <- cbind(test_data_t1_2_result,t1_2_original_rounded )
colnames(t1_2_output) = c("Actual","Predicted")
head(t1_2_output)

library(Metrics)
RMSE_t1_nn_2 = rmse(t1_2_output$Actual, t1_2_output$Predicted)
MAE_t1_nn_2 = mae(t1_2_output$Actual, t1_2_output$Predicted)
MAPE_t1_nn_2 = mape(t1_2_output$Actual, t1_2_output$Predicted)
SMAPE_t1_nn_2 = smape(t1_2_output$Actual, t1_2_output$Predicted)
```

```
RMSE_t1_nn_2
MAE_t1_nn_2
MAPE_t1_nn_2
SMAPE_t1_nn_2



#==============12 th MLP with 1 input variables and 3 hidden
layer=======================

#_w1953836_Train neural network model
t1_nn_3 <- neuralnet(Output ~ Input1 ,
                     data = training_Type_t1 ,
                     hidden = c(9,3,5),
                     linear.output = TRUE)

#_w1953836_Plot the model if desired
plot(t1_nn_3)
#_w1953836_Use the model to make predictions on testing data
t1_nn_3_result <- predict(t1_nn_3, testing_Type_t1)
t1_nn_3_result

t1_3_original <-
unnormalize(t1_nn_3_result,min_t1_lagged_data,max_t1_lagged_data)
#t1_3_original
t1_3_original_rounded <-as.data.frame(sapply(t1_3_original,round,digits=2))
#t1_3_original_rounded

test_data_t1_3_result <- exchangeUSD_df[401:499,]

t1_3_output <- cbind(test_data_t1_3_result,t1_3_original_rounded )
colnames(t1_3_output) = c("Actual","Predicted")
head(t1_3_output)

library(Metrics)
```

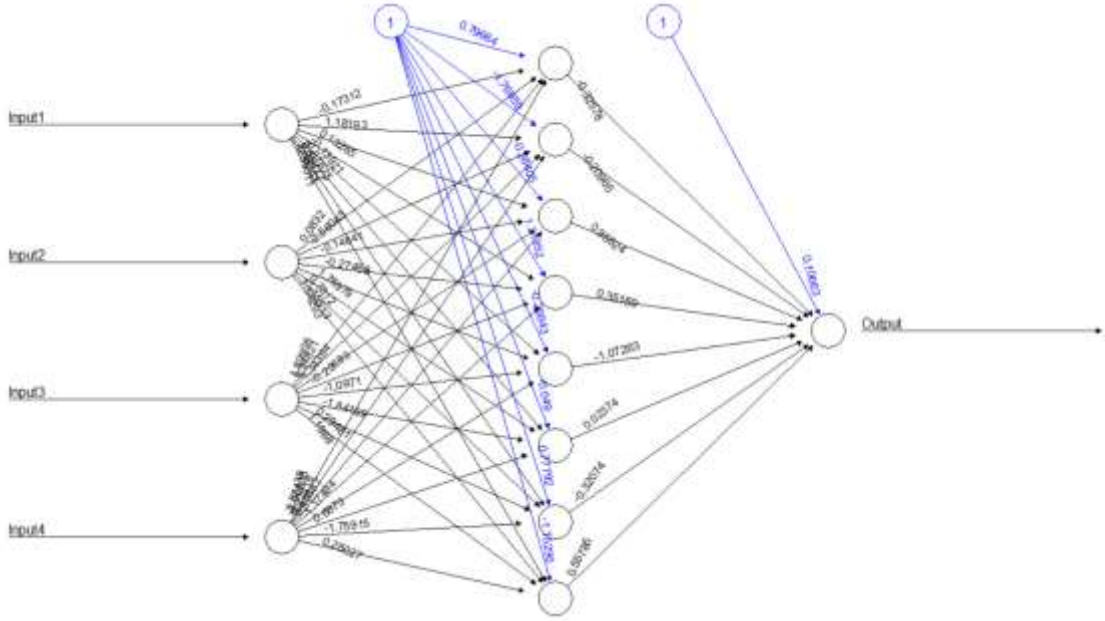
```
RMSE_t1_nn_3 = rmse(t1_3_output$Actual, t1_3_output$Predicted)
MAE_t1_nn_3 = mae(t1_3_output$Actual, t1_3_output$Predicted)
MAPE_t1_nn_3 = mape(t1_3_output$Actual, t1_3_output$Predicted)
SMAPE_t1_nn_3 = smape(t1_3_output$Actual, t1_3_output$Predicted)
RMSE_t1_nn_3
MAE_t1_nn_3
MAPE_t1_nn_3
SMAPE_t1_nn_3


#==================================================================================
==============================

RMSE_all =
c(RMSE_t4_nn_1,RMSE_t4_nn_2,RMSE_t4_nn_3,RMSE_t3_nn_1,RMSE_t3_nn_2,RMSE_t3_nn_3,
             RMSE_t2_nn_1,RMSE_t2_nn_2,RMSE_t2_nn_3,RMSE_t1_nn_1,RMSE_t1_nn_2,RMS
E_t1_nn_3)

MAE_all
=c(MAE_t1_nn_3,MAE_t1_nn_2,MAE_t1_nn_1,MAE_t2_nn_3,MAE_t2_nn_2,MAE_t2_nn_1,
           MAE_t3_nn_3,MAE_t3_nn_2,MAE_t3_nn_1,MAE_t4_nn_3,MAE_t4_nn_2,MAE_t4_nn_
1)
MAPE_all =
c(MAPE_t4_nn_1,MAPE_t4_nn_2,MAPE_t4_nn_3,MAPE_t3_nn_1,MAPE_t3_nn_2,MAPE_t3_nn_1,
             MAPE_t2_nn_1,MAPE_t2_nn_2,MAPE_t2_nn_1,MAPE_t1_nn_1,MAPE_t1_nn_2,MAP
E_t1_nn_3)
SMAPE_all =
c(SMAPE_t1_nn_1,SMAPE_t1_nn_2,SMAPE_t1_nn_3,SMAPE_t2_nn_1,SMAPE_t2_nn_2,SMAPE_t2_
nn_3,
              SMAPE_t3_nn_1,SMAPE_t3_nn_2,SMAPE_t3_nn_3,SMAPE_t4_nn_1,SMAPE_t4_nn
_2,SMAPE_t4_nn_1)

#_w1953836_create the comparison table
comparison_table = data.frame(Model =c("Model 1","Model 2","Model 3","Model
4","Model 5",
```

```
                                "Model 6","Model 7","Model 8","Model 9","Model 10",
                                "Model 11","Model 12"),
RMSE = RMSE_all,
MAE = MAE_all,
MAPE = MAPE_all,
SMAPE =SMAPE_all)
comparison_table

#==============================================================================
========================


#_w1953836_plot for best MLP Model 10
par(mfrow=c(1,1))
plot(t1_1_output$Actual, t1_1_output$Predicted,
     col = 'red',
     main = 'real vs predicted NN',
     pch=18,cex=0.7)
abline(a=0,b=1,h=90,v=90)
#_w1953836_model 10 in graphically
x=1:length(t1_1_output$Actual)
plot(x,t1_1_output$Actual,col='red',type="l",lwd=2,
     main=" USD Exchange Prediction")
lines(x,t1_1_output$Predicted,col='blue',lwd=2)
legend("topright", legend = c("original-strength","predicted-strength"),
       fill = c("red","blue"),col = 2:3,adj = c(0,0.6))
grid()
```

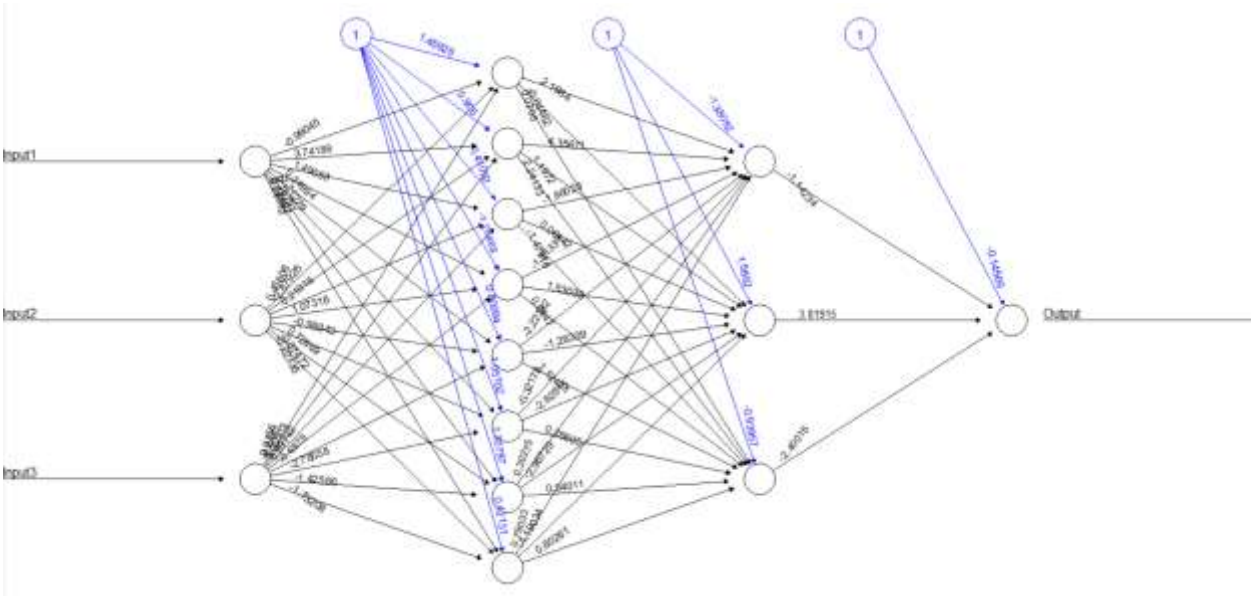
Appendix B.2 – the plot of model 2



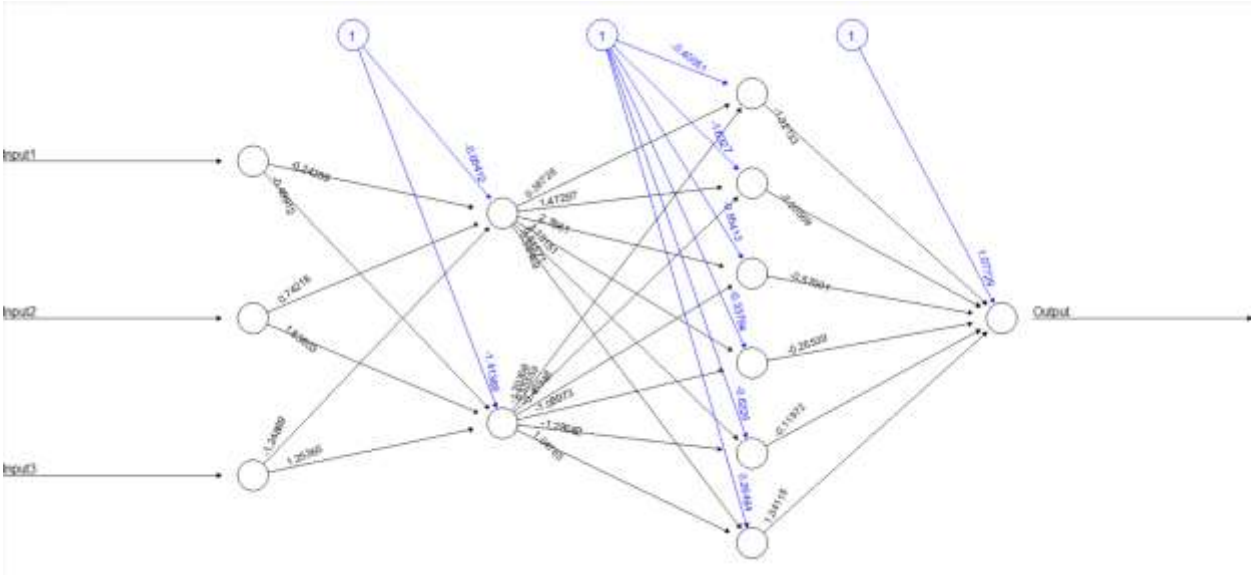
Appendix B.3 – the plot of model 3



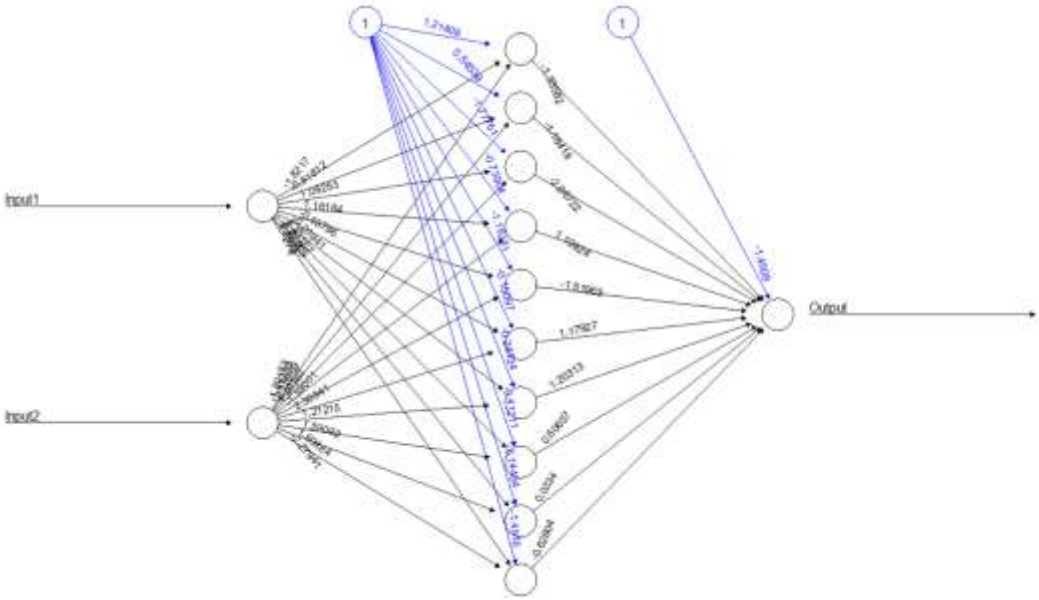
Appendix B.4 – the plot of model 5



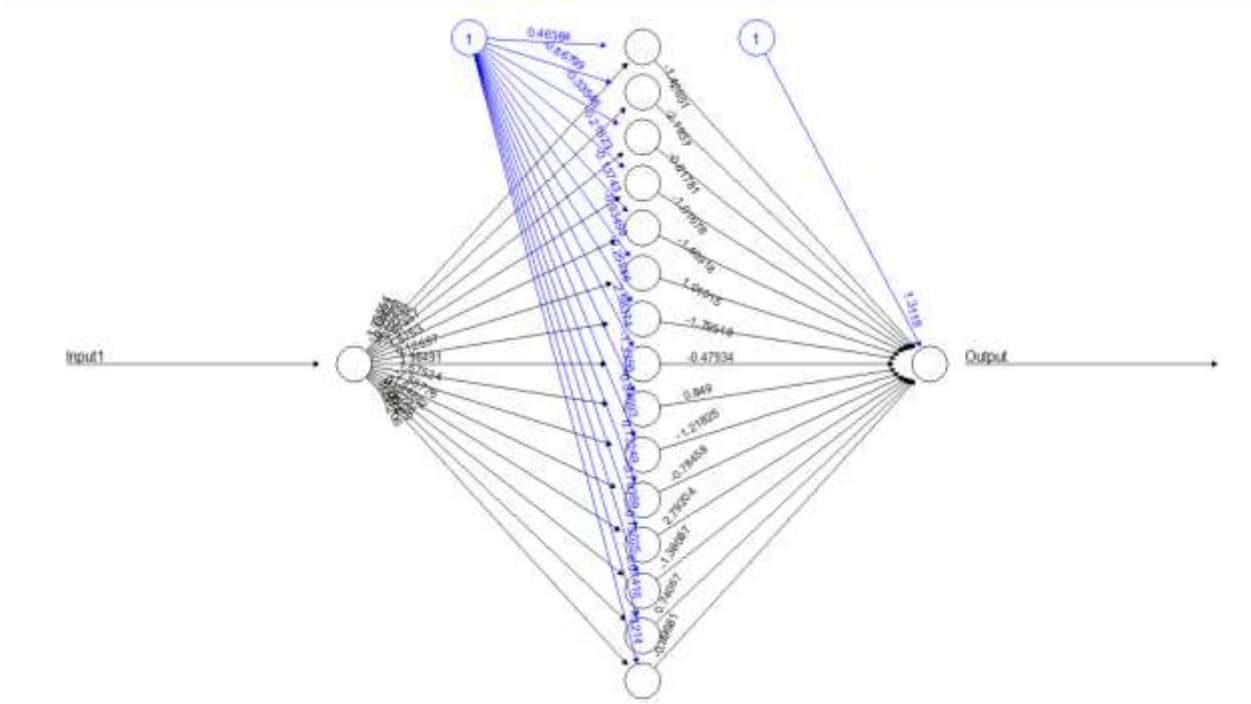
Appendix B.5 – the plot of model 6



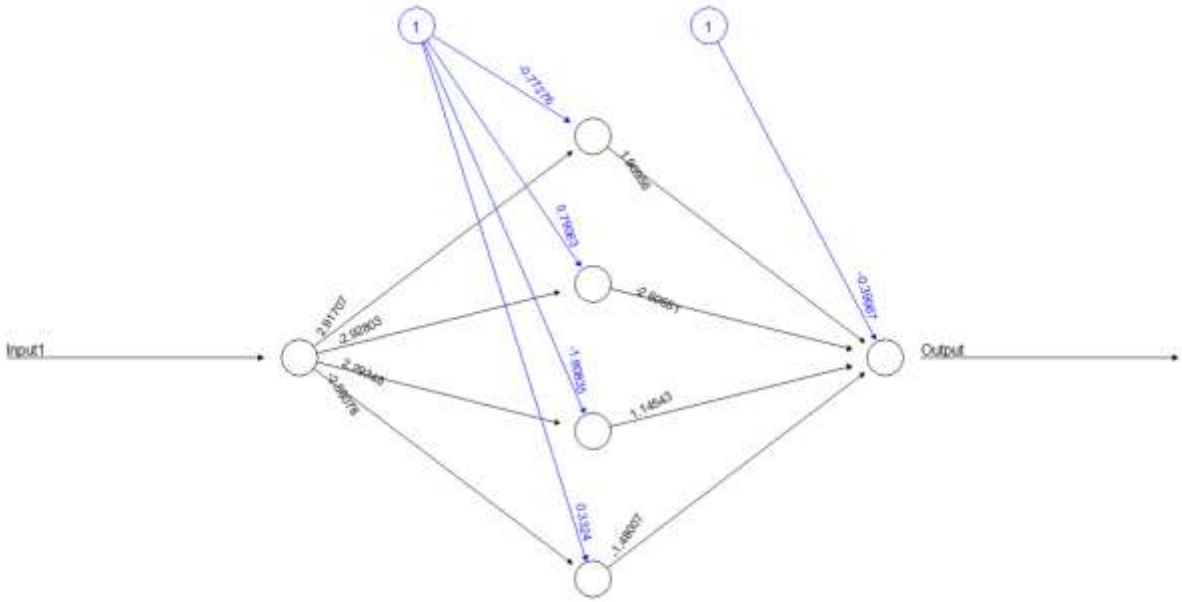
Appendix B.6– the plot of model 7



Appendix B.7 – the plot of model 10



Appendix B.8 – the plot of model 11



Appendix B.9 – the plot of model 12