Informatics Institute of Technology
University of Westminster

Machine Learning and Data Mining.

5DATA001C

Module Leader: Mr. Nipuna Senanayake
Date of Submission: 02/05/2024
Assignment Type: Individual

Serasinghe Pathiranage Imasha Udayangi

UOW No - w1956115
IIT No – 20221357

Tutorial Group: - L5 SE-C

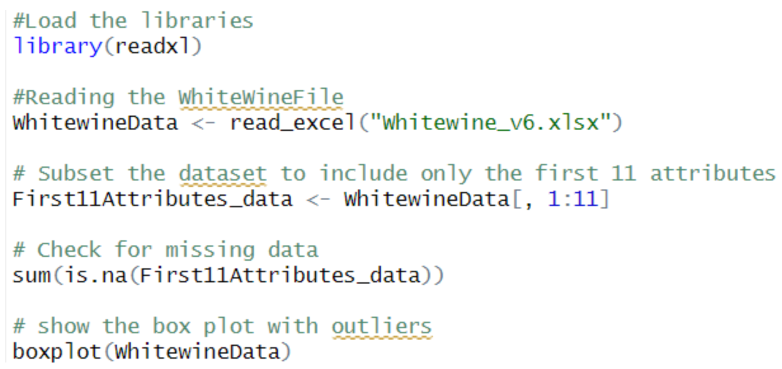
Table of Contents

1. PARTITIONING CLUSTERING PART.

<u>1st Subtask Objectives:</u>

<u>a) Outlier detection and removal and scaling.</u>

The whole data set is more valuable because of the various actions that this process depends on. Removing outliers comes first in data preprocessing since K-means clustering is more sensitive to outliers. Boxplot has been used to see it while the code is running. The code that follows refers to that,



```
#Load the libraries
library(readxl)

#Reading the WhiteWineFile
WhitewineData <- read_excel("Whitewine_v6.xlsx")

# Subset the dataset to include only the first 11 attributes
First11Attributes_data <- WhitewineData[, 1:11]

# Check for missing data
sum(is.na(First11Attributes_data))

# show the box plot with outliers
boxplot(WhitewineData)
```

In R studios, a boxplot can be created to visualize the outliers that require additional attention.

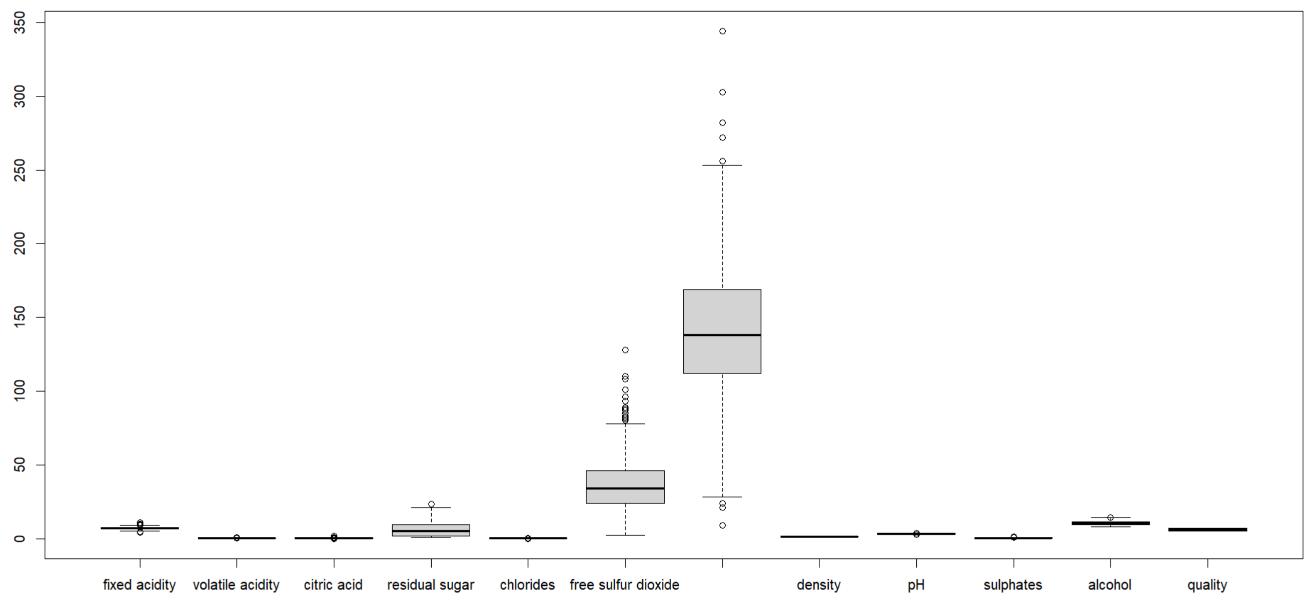
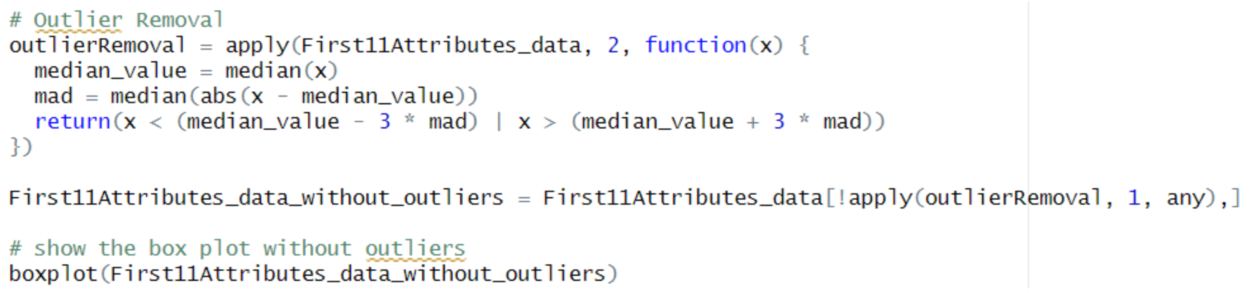


Figure 1: Check the plots with outliers.

In the following code, I used the Median Absolute Deviation (MAD) method to go through the data set, identify the outliers and remove them from the original data set.



```
# Outlier Removal
outlierRemoval = apply(First11Attributes_data, 2, function(x) {
  median_value = median(x)
  mad = median(abs(x - median_value))
  return(x < (median_value - 3 * mad) | x > (median_value + 3 * mad))
})

First11Attributes_data_without_outliers = First11Attributes_data[!apply(outlierRemoval, 1, any),]

# show the box plot without outliers
boxplot(First11Attributes_data_without_outliers)
```

After the outlier removal we can again use a boxplot to see that the outliers have been removed correctly,

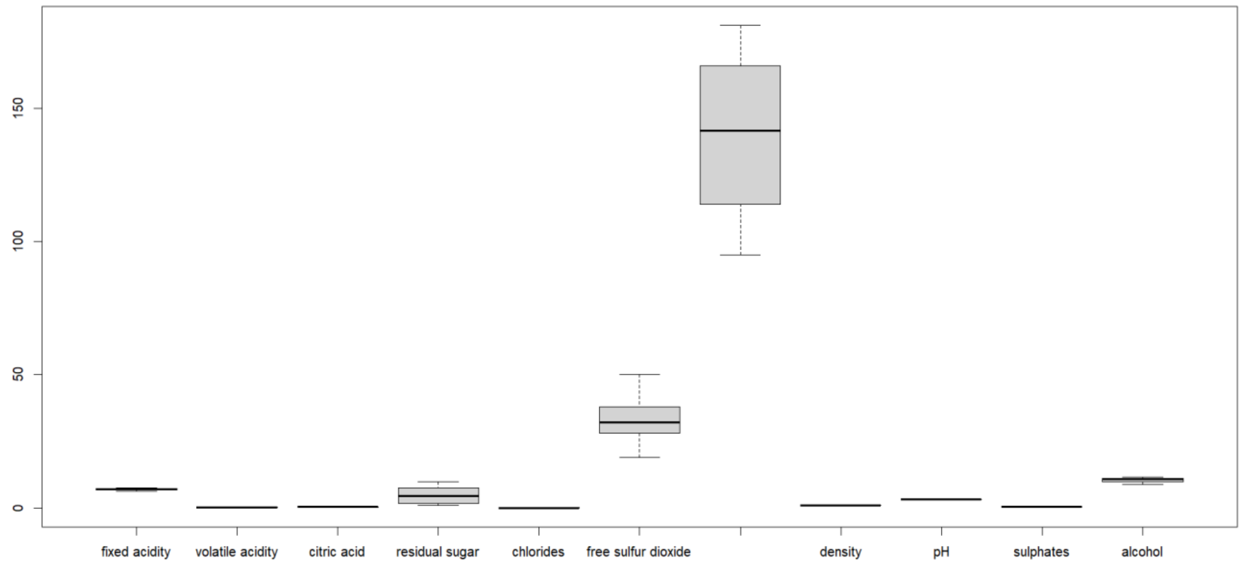
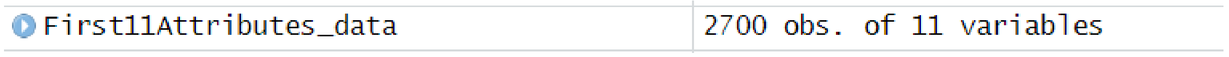


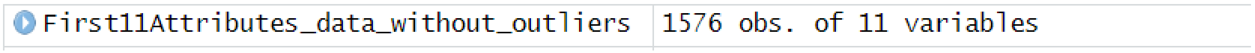
Figure 2: Check the plots without outliers

After removing outliers, the data set reduces from 2700 to 1576 records, which indicates that 1124 records were outliers in this white wine data set.



```
First11Attributes_data        2700 obs. of 11 variables
```

Figure 3: Wine Dataset with outliers

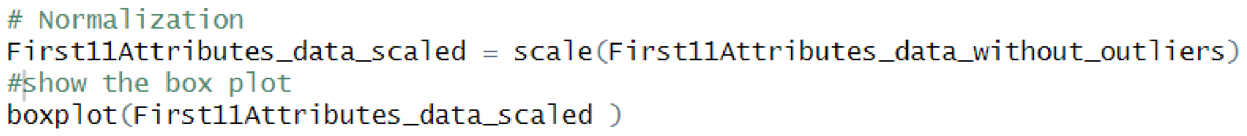


```
First11Attributes_data_without_outliers   1576 obs. of 11 variables
```

Figure 4: Wine Dataset after removing outliers

The next step is to scale the data. The range of features is standardised through scaling so that they all have the same scale and can be readily compared. This is important because different units or scales of features could lead to biased clustering results. Scaling also helps k-means to reach faster convergence by limiting the influence of features with larger magnitudes on the clustering process.

Here, we use the built-in scaling function, which uses the z-score normalization. After scaling once more, a box plot can be used to show it as follows:



```
# Normalization
First11Attributes_data_scaled = scale(First11Attributes_data_without_outliers)
#show the box plot
boxplot(First11Attributes_data_scaled )
```

Figure 5: Normalizing with scale() function

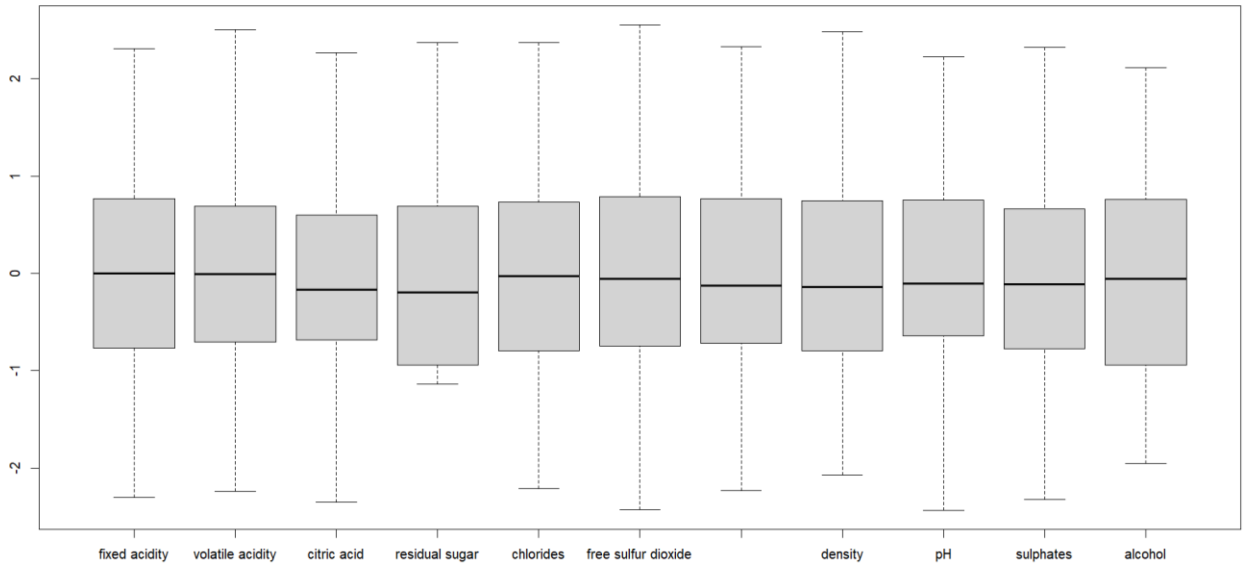
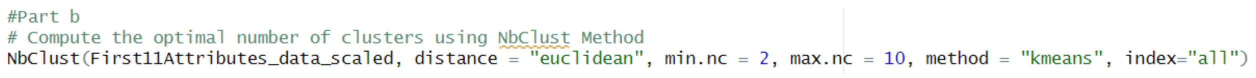


Figure 6: Boxplots after scaling

b) Determine the number of clusters using automated tools.

A variety of techniques are used as tools for automated clustering. In this project, NbClust, Elbow method, Gap statistics method, and silhouette method are considered.

NbClust



```
#Part b
# Compute the optimal number of clusters using NbClust Method
NbClust(First11Attributes_data_scaled, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index="all")
```

Figure 7: NbClust method

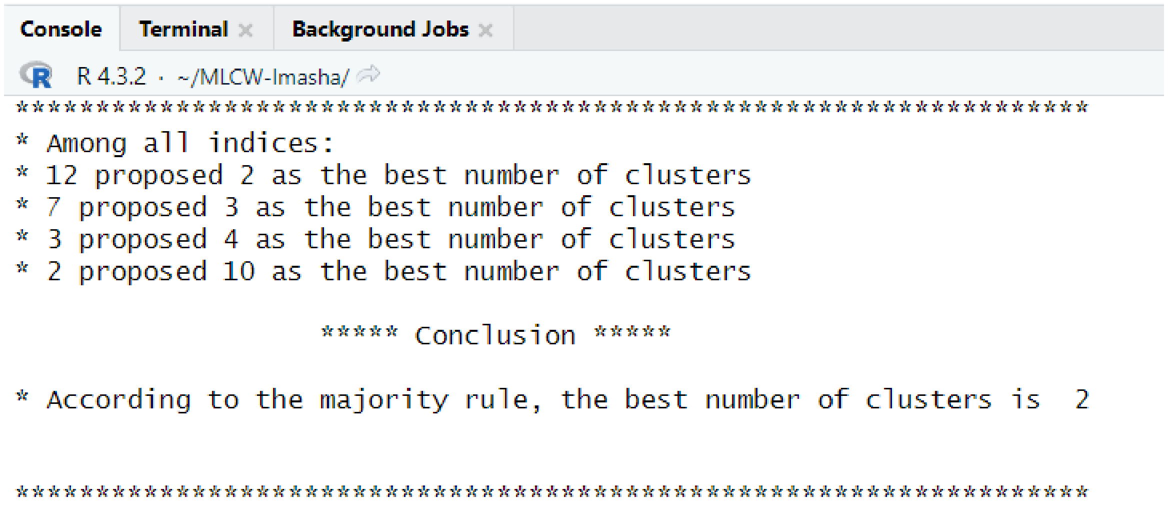
Here, it uses the NbClust() function from the NbClust package in R to conduct a cluster analysis on the scaled numerical variables in the *First11Attributes_data_scaled* dataset.

The distance parameter, which is set to "euclidean" in this instance, indicates the distance metric to be used. "Kmeans" is the clustering algorithm specified by the method parameter.

The range of clusters to be considered is specified by the min.nc and max.nc parameters, which are set to 2 and 10, respectively. Ultimately, the index parameter set to "all" to compute all available indices indicates which cluster validity indices to compute.

The results of the cluster analysis are returned by the NbClust() function as a list that includes information about the number of clusters that work best based on the selected index, the value of the selected index for each number of clusters, and the clustering results for each number of clusters. The majority rule suggests that 2 clusters are an ideal number of clusters.

That output can be visualized as follows,



```
*******************************************************************
* Among all indices:
* 12 proposed 2 as the best number of clusters
* 7 proposed 3 as the best number of clusters
* 3 proposed 4 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 8: NbClust method's output in console

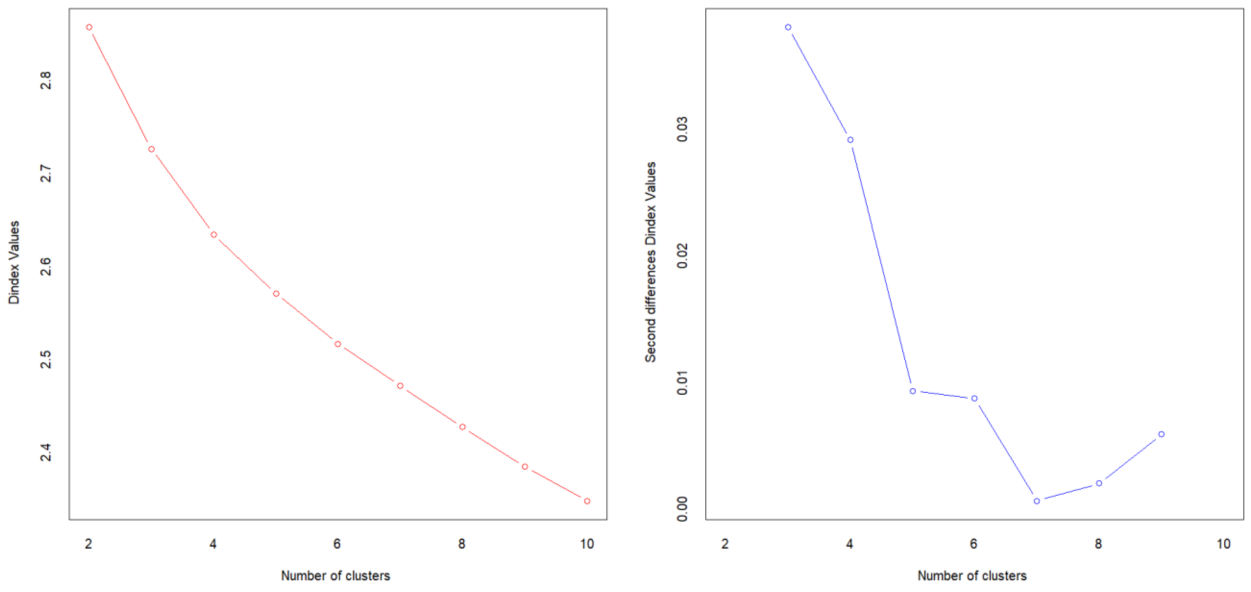


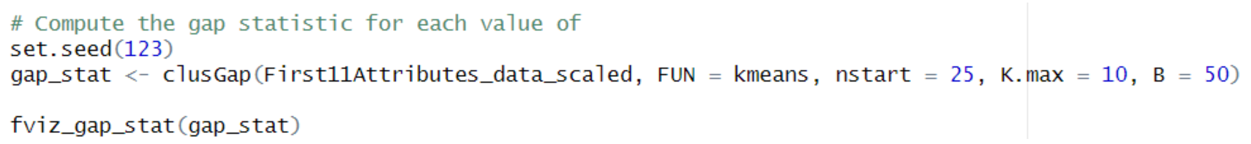
Figure 9: NbClust method's output

Gap statistics method

A statistical methodology for figuring out how many clusters in a dataset are best for use with a clustering algorithm is the gap statistic approach. The process compares the predicted value under a null reference distribution to the total within-cluster variance, the sum of squared distances between data points and their cluster centre.

The clusGap() function uses the k-means clustering method with nstart = 25 random starts and B = 50 bootstrap replicates to compute the gap statistic for varying numbers of clusters (from 1 to K.max = 10 in this case). The gap statistic data are visualized using the factoextra package's fviz_gap_stat() function.

The gap statics method suggested the best number of clusters as 10 clusters,



```
# Compute the gap statistic for each value of
set.seed(123)
gap_stat <- clusGap(First11Attributes_data_scaled, FUN = kmeans, nstart = 25, K.max = 10, B = 50)

fviz_gap_stat(gap_stat)
```

Figure 10: Gap statistics method

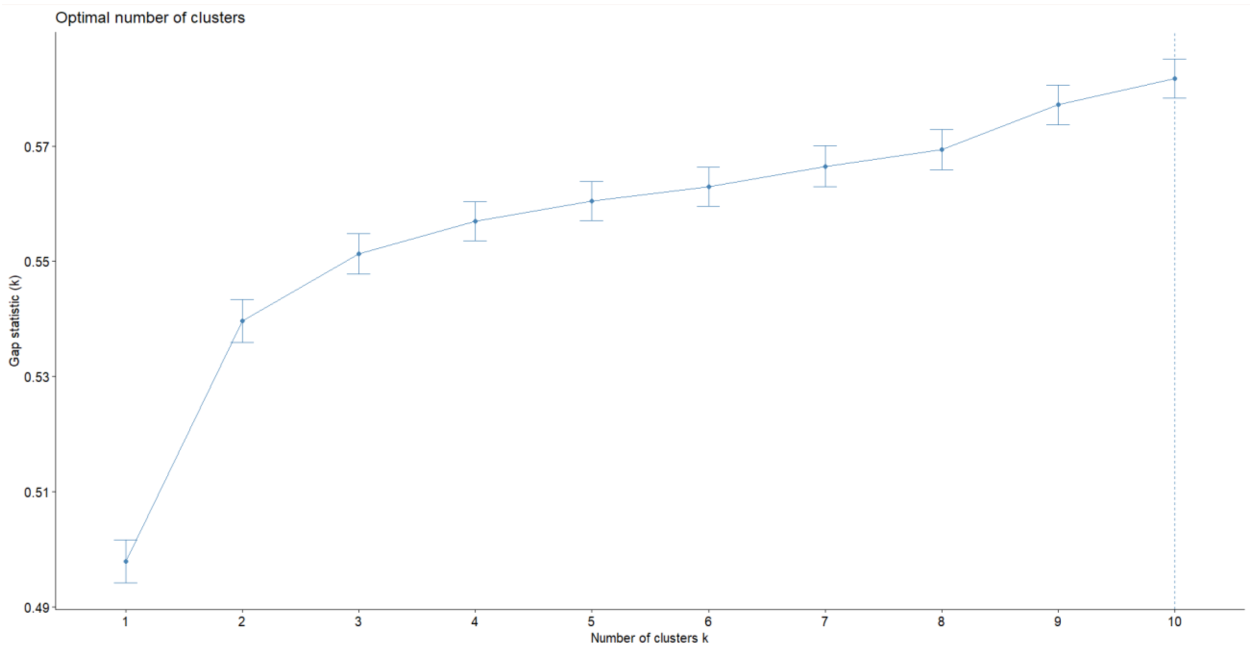


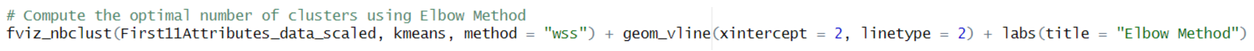
Figure 11: Gap statistics method's output

Elbow method

The elbow method is another automated way of determining how many clusters would work for a dataset. The Within Cluster Sum of Squares approach is used to do this. The WSS value is saved after the program computes the WSS ten times. The WSS value for the number of calculated clusters is determined in each cycle. When there are fewer clusters, the WSS score is relatively high; as the number of clusters increases, the score falls. The elbow approach finds the point where the elbow shape forms with the relevant WSS values and number of clusters, hence determining the optimal number of clusters for the provided dataset.

Using the k-means clustering algorithm, the following code creates a plot of the within-cluster sum of squares for varying numbers of clusters in the *First11Attributes_data_scaled* dataset. The "elbow" point, shown by the vertical line at x = 2, can be used to calculate the optimal number of clusters.

The elbow method also suggests the best number of clusters as 2 clusters.



```
# Compute the optimal number of clusters using Elbow Method
fviz_nbclust(First11Attributes_data_scaled, kmeans, method = "wss") + geom_vline(xintercept = 2, linetype = 2) + labs(title = "Elbow Method")
```

Figure 12: Elbow method

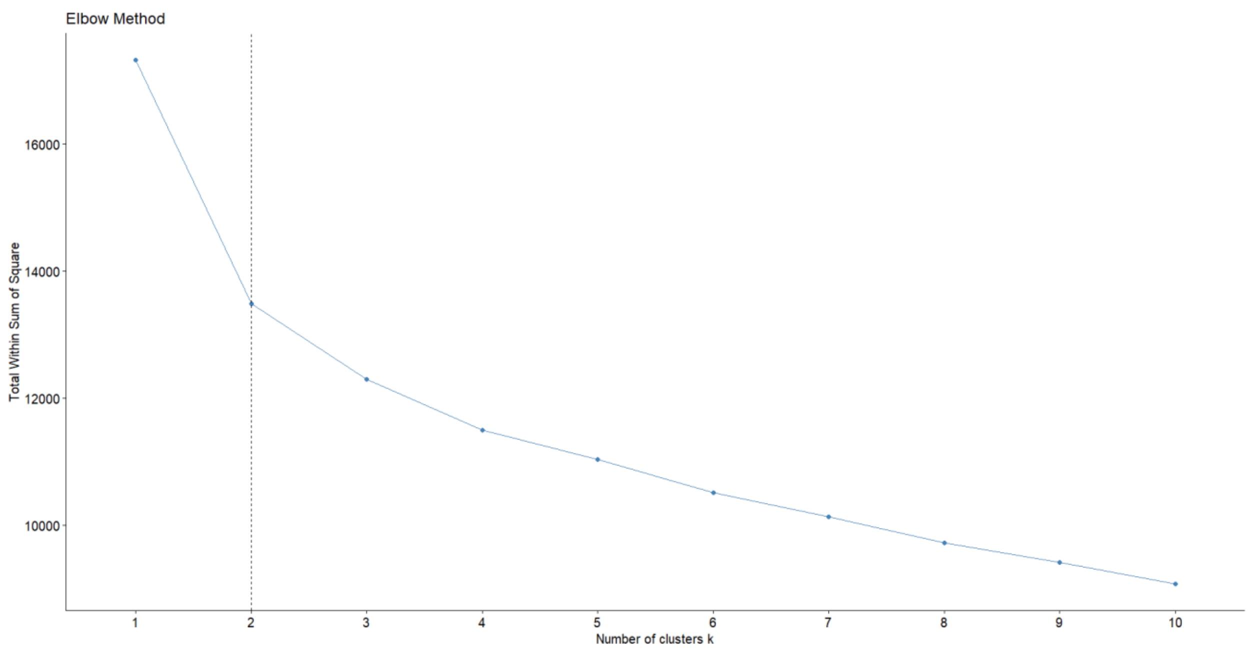


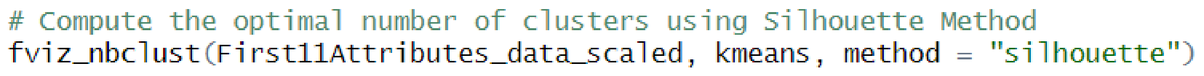
Figure 13: Elbow methods output

Silhouette method.

The silhouette method can be used to find the optimal number of clusters in a dataset by finding the silhouette coefficient for a range of values of k (the number of clusters) and choosing the value of k that maximizes the mean silhouette coefficient. This methodology is commonly used with other clustering validation methods, like the elbow method or gap statistic, to determine the optimal number of clusters for a given dataset.

Here, we use the silhouette method to determine the ideal number of clusters and plot the outcomes of the k-means clustering algorithm applied to the First11Attributes_data_scaled dataset. The optimal number of clusters can be found using the plot to examine the clustering indices visually.

Here the silhouette method suggests 2 clusters as the best number of clusters.



```
# Compute the optimal number of clusters using Silhouette Method
fviz_nbclust(First11Attributes_data_scaled, kmeans, method = "silhouette")
```

Figure 14: Silhouette method

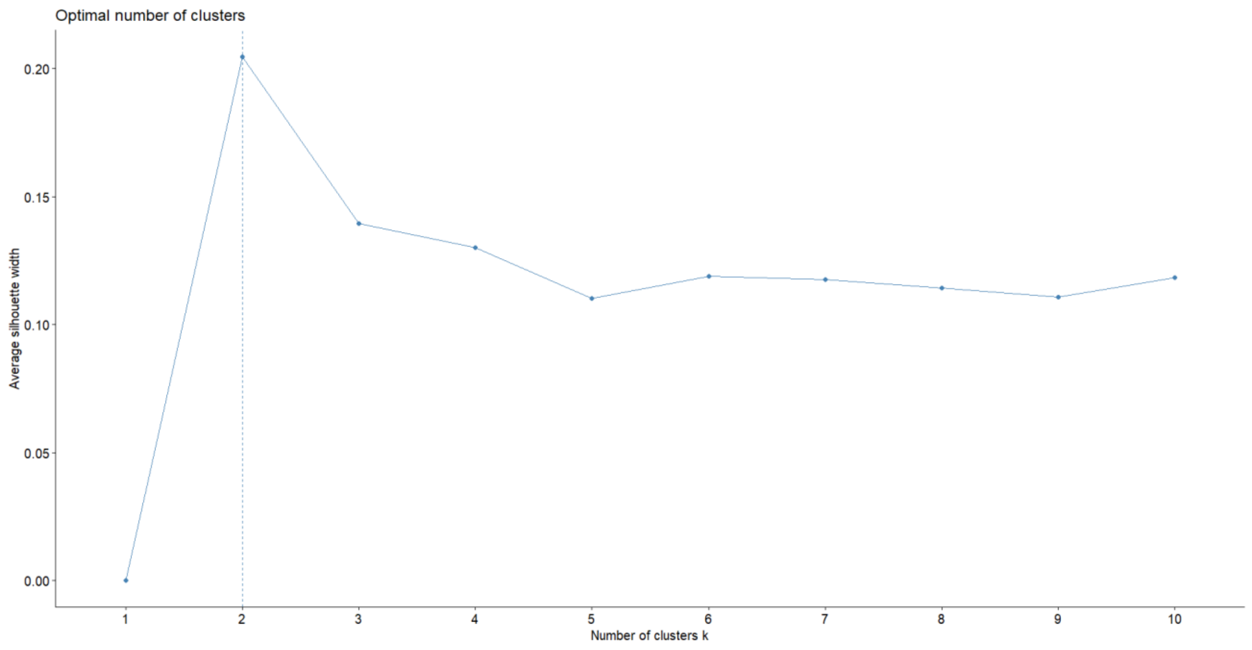
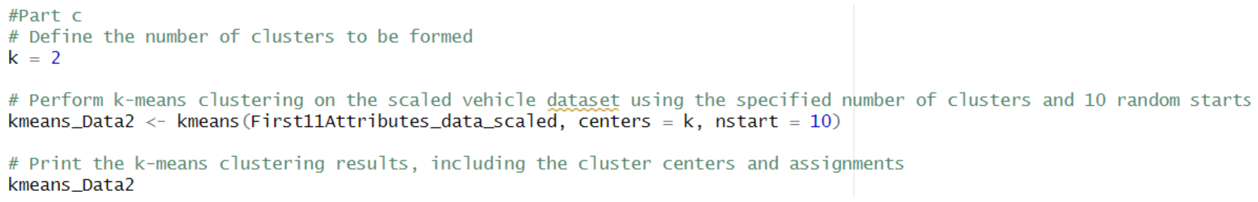


Figure 15: Silhouette methods output

In conclusion 3 out of 4 automated tools suggested the best number of clusters to be used for clustering this data set as 2 clusters. Only in the gap statics method suggested to use 10 clusters. Therefore using the majority rule, it has been decided to use 2 cluster for the clustering process.

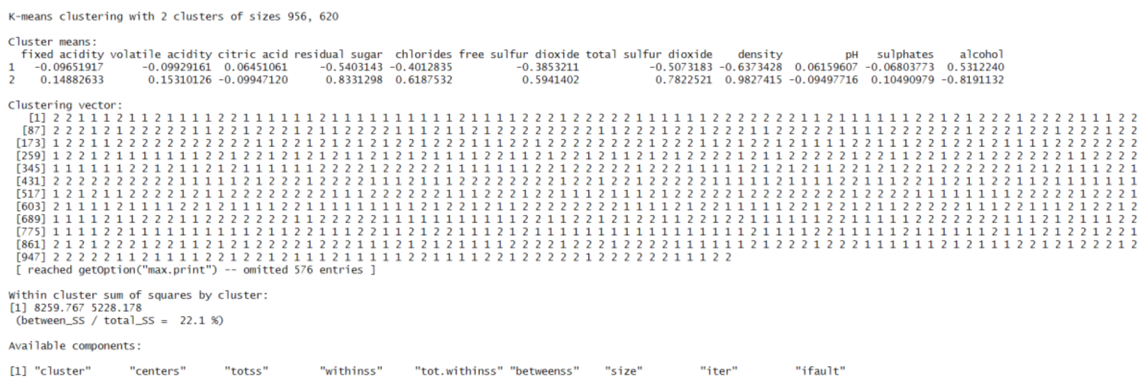
c) Kmeans clustering investigation

After determining the number of clusters to use, the clustering procedure uses the kmeans() function. The *First11Attributes_data_scaled* dataset is subjected to the k-means clustering algorithm with k=2 clusters, and the result is stored in the *kmeans_Data2* object.



```r
#Part c
# Define the number of clusters to be formed
k = 2

# Perform k-means clustering on the scaled vehicle dataset using the specified number of clusters and 10 random starts
kmeans_Data2 <- kmeans(First11Attributes_data_scaled, centers = k, nstart = 10)

# Print the k-means clustering results, including the cluster centers and assignments
kmeans_Data2
```

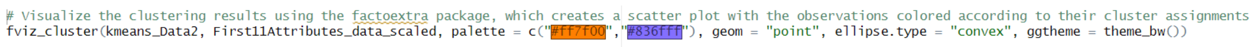
Figure 16: kmeans() function



```
K-means clustering with 2 clusters of sizes 956, 620

Cluster means:
  fixed acidity volatile acidity citric acid residual sugar  chlorides free sulfur dioxide total sulfur dioxide    density          pH   sulphates    alcohol
1   -0.09651917      -0.09929161  0.06451061     -0.5403143 -0.4012835          -0.3853211           -0.5073183 -0.6373428  0.06159607 -0.06803773  0.5312240
2    0.14882633       0.15310126 -0.09947120      0.8331298  0.6187532           0.5941402            0.7822521  0.9827415 -0.09497716  0.10490979 -0.8191132

Clustering vector:
   [1] 2 2 1 1 1 2 1 1 2 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 2 2 2 2 2 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 1 2 2 1 2 1 2 2 2 1 2 2 2 2 1 1 1 2 2
  [87] 2 2 2 1 2 1 2 2 2 2 2 1 1 2 2 1 2 2 2 1 2 1 1 2 2 2 2 1 1 1 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 2 2 2 1 1 2 2 2 2 2 1 1 1 1 2 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 2 1 2 2
 [173] 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 2 1 2 2 1 2 2 2 1 1 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 1 1 2 1 2 2 1 2 1 1 1 1 2 2 1 1 1 2 2 1 2 2 1 2 1 1 1 1 2 2 2 2 2 2 2
 [259] 1 2 2 1 2 1 1 1 1 1 1 1 2 2 1 2 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 1 1 1 2 2 1 1 2 1 2 2 1 2 1 1 2 1 2 1 2 2 2 2 1 2 1 1 2 2 2 2 2 1 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2
 [345] 1 1 1 1 1 1 2 2 1 2 1 1 2 1 2 1 1 1 1 1 1 2 2 2 2 1 2 2 2 2 2 1 1 1 1 1 1 1 2 2 1 2 2 2 2 2 1 2 2 2 1 1 1 1 1 2 1 2 1 2 1 1 2 2 1 2 1 2 2 1 2 2 2 2 2 2 1 2 1 2 1 1 1 2 2 1
 [431] 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 2 2 2 1 2 2 2 2 1 1 1 2 1 1 1 2 2 2 2 2 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 2 2 1 1 2 1 1 2 2 1 1 2 1 1 1 1 1 1 1
 [517] 1 2 1 2 1 1 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 2 1 1 1 2 2 2 1 2 2 1 1 1 2 1 1 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 1
 [603] 2 1 1 1 1 2 1 1 1 1 2 2 1 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 2 2 1 2 1 1 2 2 2 2 2 2 2 2 1 1 1 1 2 1 2 2 1 1 1 1 1 2 2 1 2 2 1 2 1 1 2 2 2 2 2 1 1 1 2 2 1 1 1 2 1 2 2 1 2
 [689] 1 1 1 1 2 1 1 2 2 2 1 1 2 2 2 2 2 2 2 1 1 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 1 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 2 1 2 1 1 1 2 2
 [775] 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 2 2 1 1 2 1 2 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 2 2 1 2 2 1
 [861] 2 1 2 1 2 2 2 1 2 2 1 1 2 1 2 1 2 2 2 2 1 2 2 2 1 1 1 2 1 2 1 2 1 1 1 1 2 2 2 2 1 2 2 2 1 2 2 2 2 1 1 1 1 1 1 2 1 2 2 2 1 2 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 2 1 2 2 2 1 2
 [947] 2 2 2 2 2 1 1 2 1 1 1 1 2 2 1 2 2 1 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2
 [ reached getOption("max.print") -- omitted 576 entries ]

Within cluster sum of squares by cluster:
[1] 8259.767 5228.178
 (between_SS / total_SS =  22.1 %)

Available components:

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

Figure 17: kmeans cluster with 2 clusters

The following visualization illustrates the clustering results created by the fviz_cluster() function:



```r
# Visualize the clustering results using the factoextra package, which creates a scatter plot with the observations colored according to their cluster assignments
fviz_cluster(kmeans_Data2, First11Attributes_data_scaled, palette = c("#ff7f00","#836fff"), geom = "point", ellipse.type = "convex", ggtheme = theme_bw())
```

Figure 18: fviz_cluster() function

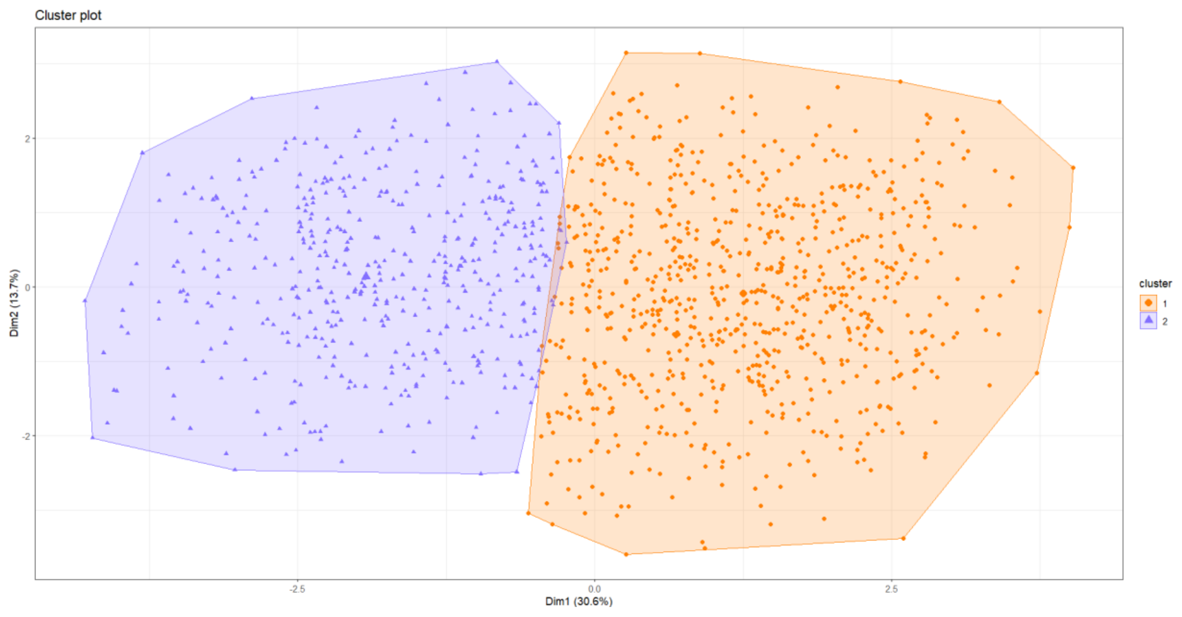
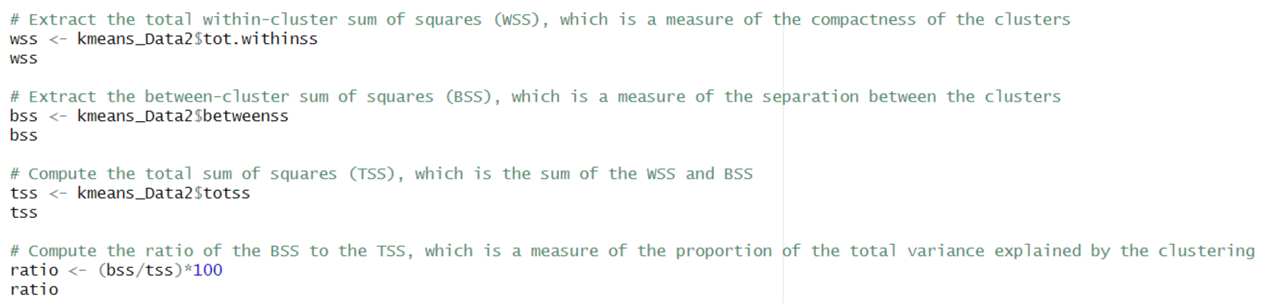


Figure 19: Visualize the clustering results using the factoextra package.

Next, we calculate the WSS, BSS, TSS and the ratio values for the clustering attempts of k=2,



```
# Extract the total within-cluster sum of squares (WSS), which is a measure of the compactness of the clusters
wss <- kmeans_Data2$tot.withinss
wss

# Extract the between-cluster sum of squares (BSS), which is a measure of the separation between the clusters
bss <- kmeans_Data2$betweenss
bss

# Compute the total sum of squares (TSS), which is the sum of the WSS and BSS
tss <- kmeans_Data2$totss
tss

# Compute the ratio of the BSS to the TSS, which is a measure of the proportion of the total variance explained by the clustering
ratio <- (bss/tss)*100
ratio
```

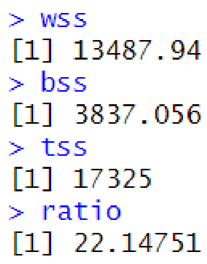
Figure 20: WSS,BSS,TSS & RATIO for k=2

The wss variable = within-cluster sum of squares. This measures the amount of variations Within each cluster.

The bss variable = between-cluster sum of squares. This measures the amount of variations between the cluster centroids.

The tss variable = total sum of squares. This measures the total amount of variations in the data.

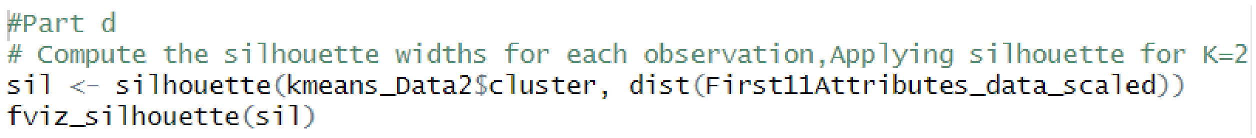
The ratio variable = ratio between the between-cluster sum of squares (bss) and the total sum of squares (tss). This measures the percentage of overall variation that the clustering can account for. To express the ratio as a percentage, multiply the result by 100.



```
> wss
[1] 13487.94
> bss
[1] 3837.056
> tss
[1] 17325
> ratio
[1] 22.14751
```

Figure 21: WSS,BSS,TSS & RATIO for k=2 outputs

d) Silhouette plot analysis of the kmeans attempts.



```
#Part d
# Compute the silhouette widths for each observation,Applying silhouette for K=2
sil <- silhouette(kmeans_Data2$cluster, dist(First11Attributes_data_scaled))
fviz_silhouette(sil)
```

Figure 22: Silhouette plot analysis of the kmeans

The fviz_silhouette function from the factoextra package is then used to visualise the silhouette plot. Each observation's silhouette width is displayed as a vertical bar in this plot, along with the height of the bar representing the clustering result's strength for that particular observation. Additionally, each cluster's average silhouette width is displayed as a horizontal line on the plot.

One valuable tool for assessing the reliability of a clustering model is the silhouette plot, which shows the distribution of silhouette widths for each observation in each cluster. The silhouette width measures an observation's degree of similarity to its own cluster in relation to other clusters. While a narrow profile implies the opposite, a wide silhouette suggests that observation is well-matched to its own cluster but poorly matched to nearby clusters.

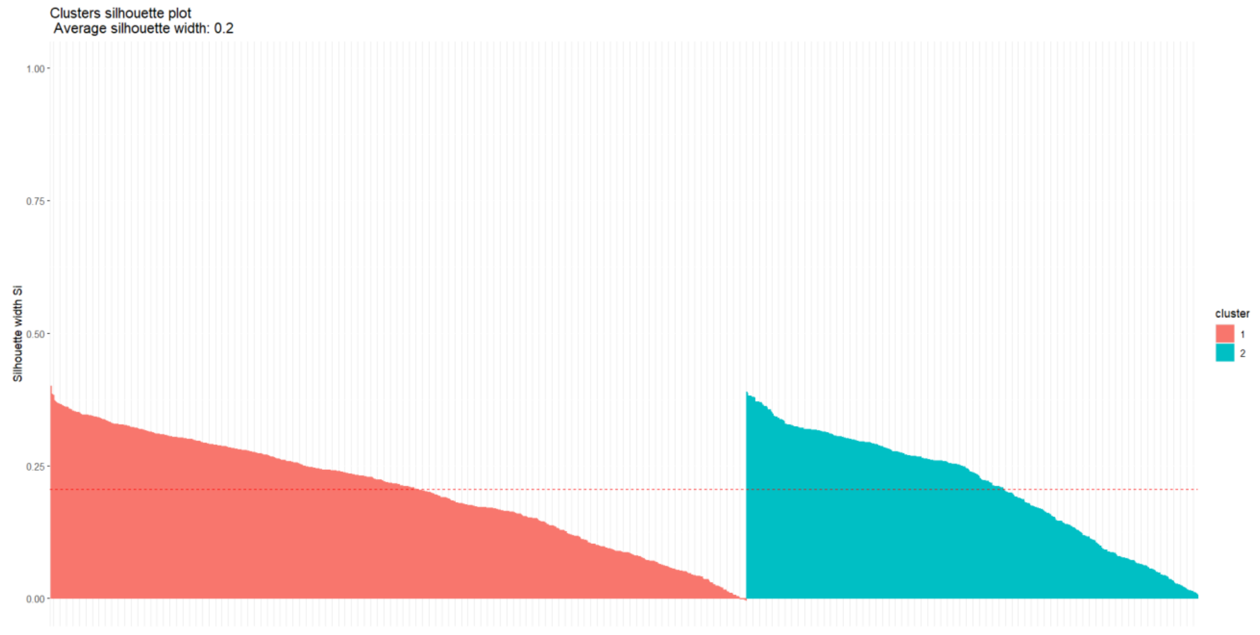


Figure 23: Silhouette plot analysis of the kmeans plot

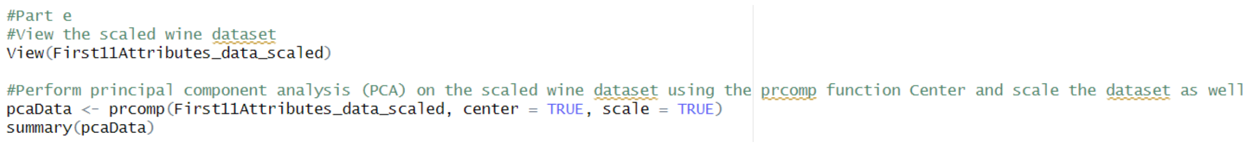
The average silhouette width score is an overall summary metric that shows how practical the clustering approach is. The width of the silhouette ranges from -1 to 1. Whereas a silhouette width value near -1 suggests that the observation would be better suited in a different cluster, a value close to +1 shows that the observation is well-matched to its cluster. When an observation's value is close to zero, it implies that it is close to the decision border between clusters; when its value is less than zero, it signifies that the observations might not be correctly classified.

2nd Subtask Objectives (PCA)

e) Apply the PCA method to this wine dataset.

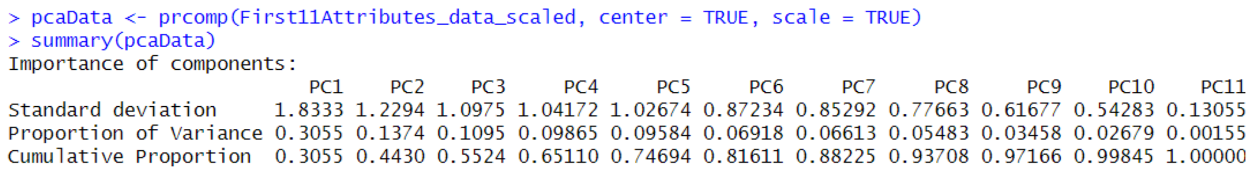
Principal Component Analysis is referred to as PCA. Finding patterns in a high-dimensional dataset and representing it in a lower-dimensional space while preserving as much information as possible is a statistical technique for data reduction.

The View() function is used to view the dataset. The scaled white wine dataset is then centred and scaled, and then PCA is performed using the prcomp() function. The summary() function is then used to summarise the PCA results.



```
#Part e
#View the scaled wine dataset
View(First11Attributes_data_scaled)

#Perform principal component analysis (PCA) on the scaled wine dataset using the prcomp function Center and scale the dataset as well
pcaData <- prcomp(First11Attributes_data_scaled, center = TRUE, scale = TRUE)
summary(pcaData)
```

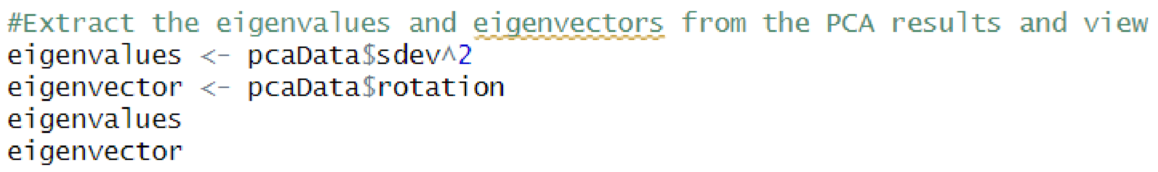
Figure 24: PCA summary()



```
> pcaData <- prcomp(First11Attributes_data_scaled, center = TRUE, scale = TRUE)
> summary(pcaData)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11
Standard deviation     1.8333 1.2294 1.0975 1.04172 1.02674 0.87234 0.85292 0.77663 0.61677 0.54283 0.13055
Proportion of Variance 0.3055 0.1374 0.1095 0.09865 0.09584 0.06918 0.06613 0.05483 0.03458 0.02679 0.00155
Cumulative Proportion  0.3055 0.4430 0.5524 0.65110 0.74694 0.81611 0.88225 0.93708 0.97166 0.99845 1.00000
```

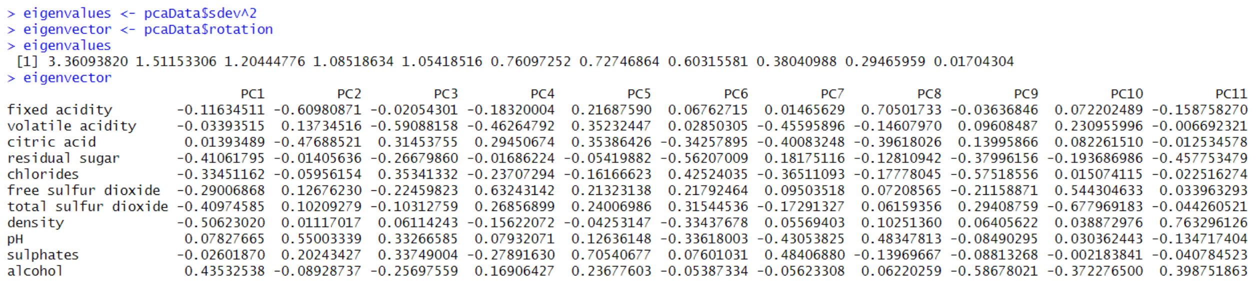
Figure 25: PCA summary() output in console

Next, using the $sdev and $rotation attributes of the prcomp object, each eigenvalues and eigenvectors were extracted from the PCA results. The eigenvalues and eigenvectors are kept separate in the eigenvalues and eigenvector variables, respectively.



```
#Extract the eigenvalues and eigenvectors from the PCA results and view
eigenvalues <- pcaData$sdev^2
eigenvector <- pcaData$rotation
eigenvalues
eigenvector
```

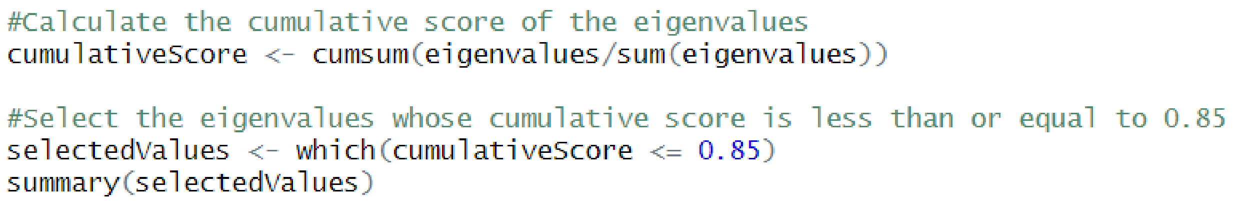
Figure 26: eigenvalues and eigenvectors



```
> eigenvalues <- pcaData$sdev^2
> eigenvector <- pcaData$rotation
> eigenvalues
 [1] 3.36093820 1.51153306 1.20444776 1.08518634 1.05418516 0.76097252 0.72746864 0.60315581 0.38040988 0.29465959 0.01704304
> eigenvector
                             PC1         PC2         PC3         PC4         PC5         PC6         PC7         PC8         PC9         PC10         PC11
fixed acidity        -0.11634511 -0.60980871 -0.02054301 -0.18320004  0.21687590  0.06762715  0.01465629  0.70501733 -0.03636846  0.072202489 -0.158758270
volatile acidity     -0.03393515  0.13734516 -0.59088158 -0.46264792  0.35232447  0.02850305 -0.45595896 -0.14607970  0.09608487  0.230955996 -0.006692321
citric acid           0.01393489 -0.47688521  0.31453755  0.29450674  0.35386426 -0.34257895 -0.40083248 -0.39618026  0.13995866  0.082261510 -0.012534578
residual sugar       -0.41061795 -0.01405636 -0.26679860 -0.01686224 -0.05419882 -0.56207009  0.18175116 -0.12810942 -0.37996156 -0.193686986 -0.457753479
chlorides            -0.33451162 -0.05956154  0.35341332 -0.23707294 -0.16166623  0.42524035 -0.36511093 -0.17778045 -0.57518556  0.015074115 -0.022516274
free sulfur dioxide  -0.29006868  0.12676230 -0.22459823  0.63243142  0.21323138  0.21792464  0.09503518  0.07208565 -0.21158871  0.544304633  0.033963293
total sulfur dioxide -0.40974585  0.10209279 -0.10312759  0.26856899  0.24006986  0.31544536 -0.17291327  0.06159356  0.29408759 -0.677969183 -0.044260521
density              -0.50623020  0.01117017  0.06114243 -0.15622072 -0.04253147 -0.33437678  0.05569403  0.10251360  0.06405622  0.038872976  0.763296126
pH                    0.07827665  0.55003339  0.33266585  0.07932071  0.12636148 -0.33618003 -0.43053825  0.48347813 -0.08490295  0.030362443 -0.134717404
sulphates            -0.02601870  0.20243427  0.33749004 -0.27891630  0.70540677  0.07601031  0.48406880 -0.13969667 -0.08813268 -0.002183841 -0.040784523
alcohol               0.43532538 -0.08928737 -0.25697559  0.16906427  0.23677603 -0.05387334 -0.05623308  0.06220259 -0.58678021 -0.372276500  0.398751863
```

Figure 27: eigenvalues and eigenvectors output in console

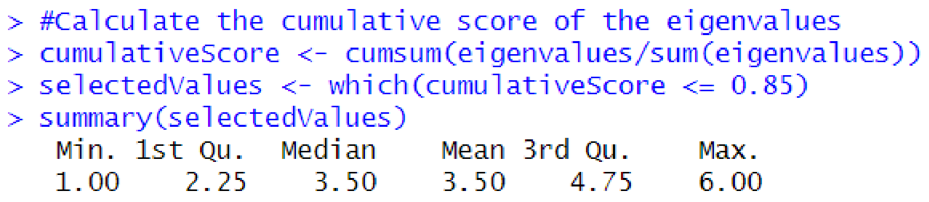
Next, using the cumsum() function, the cumulative score of the eigenvalues is determined and saved in the *cumulativeScore* variable. Using the which() function, the eigenvalues whose cumulative score is less than or equal to 0.85 are chosen and kept in the *selectedValues* variable.



```
#Calculate the cumulative score of the eigenvalues
cumulativeScore <- cumsum(eigenvalues/sum(eigenvalues))

#Select the eigenvalues whose cumulative score is less than or equal to 0.85
selectedValues <- which(cumulativeScore <= 0.85)
summary(selectedValues)
```

Figure 28: cumulativeScore

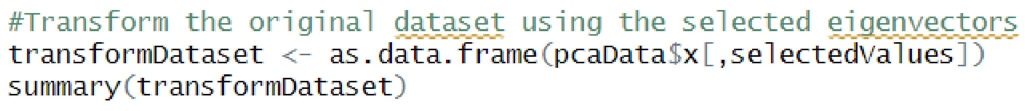
When eigenvalues are chosen and their cumulative score is less than or equal to 0.85, the output result as below,



```
> #Calculate the cumulative score of the eigenvalues
> cumulativeScore <- cumsum(eigenvalues/sum(eigenvalues))
> selectedValues <- which(cumulativeScore <= 0.85)
> summary(selectedValues)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   1.00    2.25    3.50    3.50    4.75    6.00 
```

Figure 29: cumulativeScore result

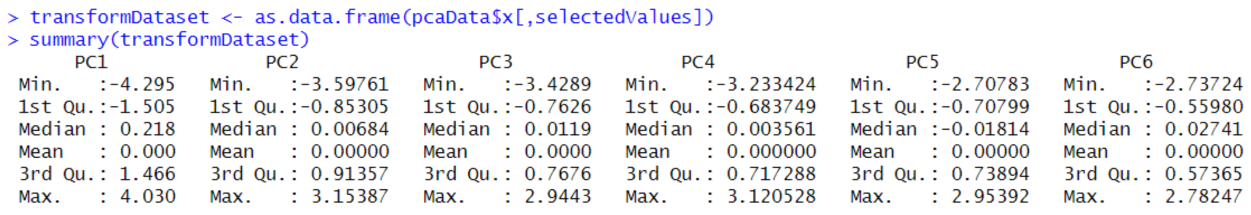
Next, using the as.data.frame() function and the chosen eigenvectors, the original dataset is changed and saved in the *transformDataset* variable. The summary() function is used to summarise the transformed dataset.



```
#Transform the original dataset using the selected eigenvectors
transformDataset <- as.data.frame(pcaData$x[,selectedValues])
summary(transformDataset)
```

Figure 30: Transform Dataset

After using the selected eigenvectors to transform the original dataset, get the summary for the transformed dataset.



```
> transformDataset <- as.data.frame(pcaData$x[,selectedValues])
> summary(transformDataset)
      PC1               PC2                PC3               PC4                 PC5                PC6          
 Min.   :-4.295   Min.   :-3.59761   Min.   :-3.4289   Min.   :-3.233424   Min.   :-2.70783   Min.   :-2.73724  
 1st Qu.:-1.505   1st Qu.:-0.85305   1st Qu.:-0.7626   1st Qu.:-0.683749   1st Qu.:-0.70799   1st Qu.:-0.55980  
 Median : 0.218   Median : 0.00684   Median : 0.0119   Median : 0.003561   Median :-0.01814   Median : 0.02741  
 Mean   : 0.000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.000000   Mean   : 0.00000   Mean   : 0.00000  
 3rd Qu.: 1.466   3rd Qu.: 0.91357   3rd Qu.: 0.7676   3rd Qu.: 0.717288   3rd Qu.: 0.73894   3rd Qu.: 0.57365  
 Max.   : 4.030   Max.   : 3.15387   Max.   : 2.9443   Max.   : 3.120528   Max.   : 2.95392   Max.   : 2.78247  
```

Figure 31: Transform Dataset output in console

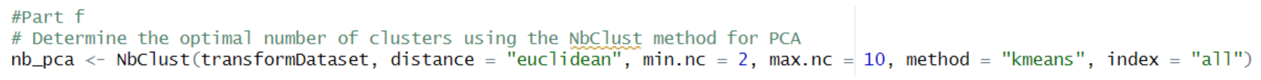
We can see that six main components (PCs) are formed from the dataset based on the results of the PCA analysis. The *cumulativeScore* variable displays the cumulative score for each PC, with the following scores: 0.3055 for the first PC, 0.4429 for the second PC, 0.5524 for the third PC, 0.6511 for the fourth PC, 0.7469 for the fifth PC, and 0.8161 for the sixth PC.

The original standardized data matrix is multiplied by the eigenvectors matrix to produce the transformed dataset, and specific principal components (PCs) are then extracted from the PCA results. The original data set's most crucial information is included in a reduced number of transformed variables produced by keeping the PCs with a cumulative score higher than 0.85. Six PCs with cumulative scores ranging from 0.3055 to 0.8161 were created by the PCA analysis. We considered collecting enough variation to faithfully represent the original data while avoiding overfitting or unnecessary complexity while determining how many PCs to keep. As a result, we decided to keep PCs that contribute at least a cumulative score of 0.85, or at least 85% of the variation in the original dataset, as explained.

f) Determining the optimal number of clusters using automated tools after PCA.

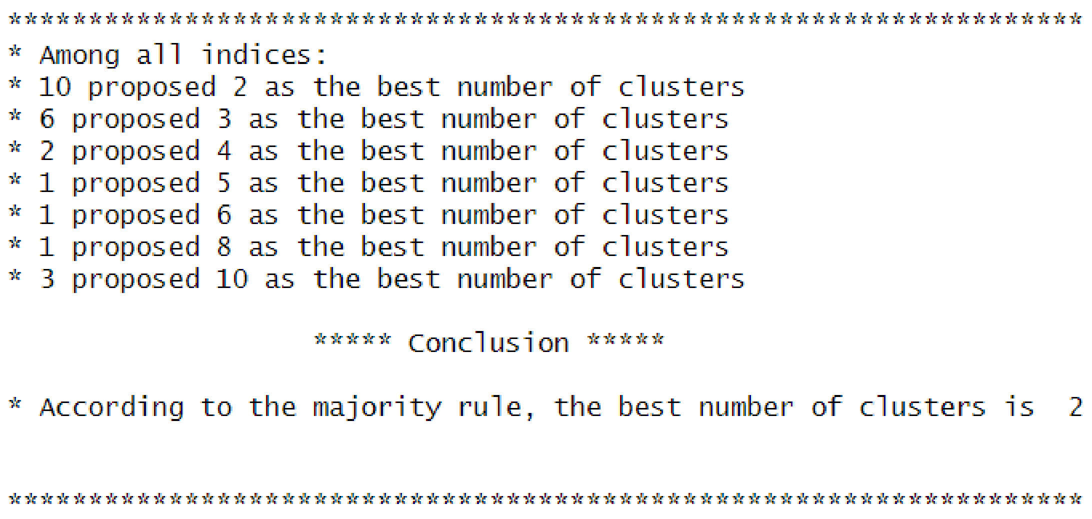
Nbclust Method.

NbClust suggests the most suitable clustering number.



```
#Part f
# Determine the optimal number of clusters using the NbClust method for PCA
nb_pca <- NbClust(transformDataset, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans", index = "all")
```

Figure 32: NbClust method for PCA.



```
******************************************************************
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 6 proposed 3 as the best number of clusters
* 2 proposed 4 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 3 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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


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

Figure 33: NbClust method output for PCA

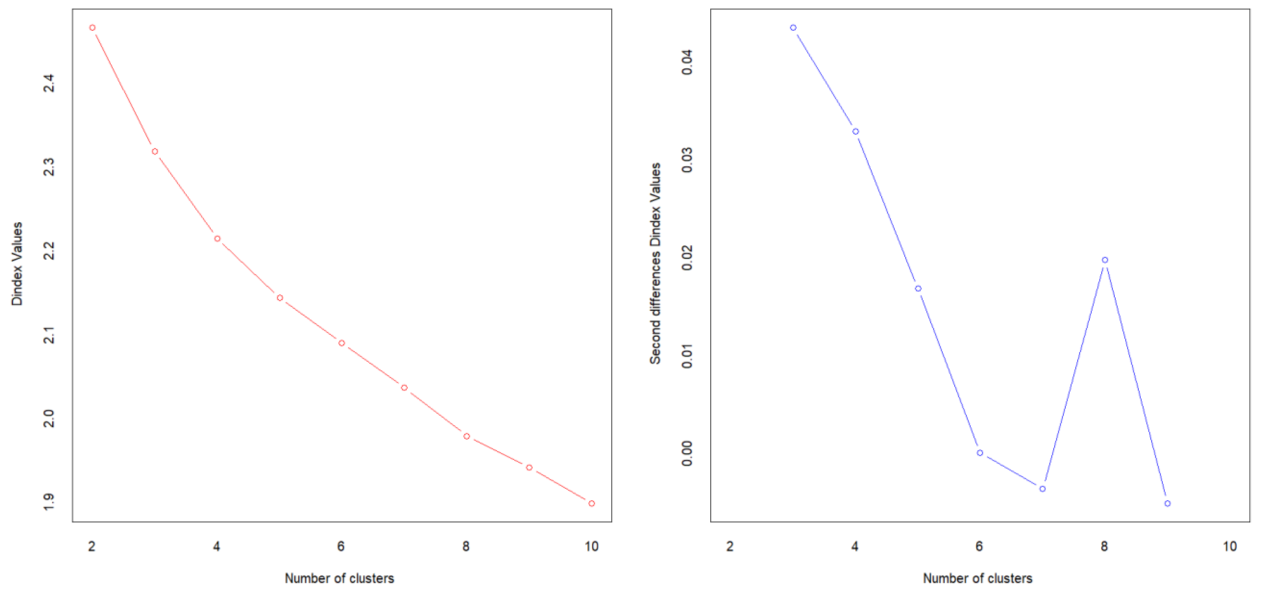
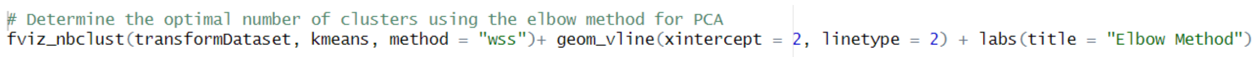


Figure 34: NbClust method result plots for PCA.

The first method is NbClust, which determines the ideal number of clusters based on the index's highest value after computing clustering indices for varying clusters. The number of clusters doesn’t change in Nbclust after doing PCA. It remains to be seen the same number for k as 2.

Elbow Method.



```
# Determine the optimal number of clusters using the elbow method for PCA
fviz_nbclust(transformDataset, kmeans, method = "wss")+ geom_vline(xintercept = 2, linetype = 2) + labs(title = "Elbow Method")
```

Figure 35: Elbow method for PCA.

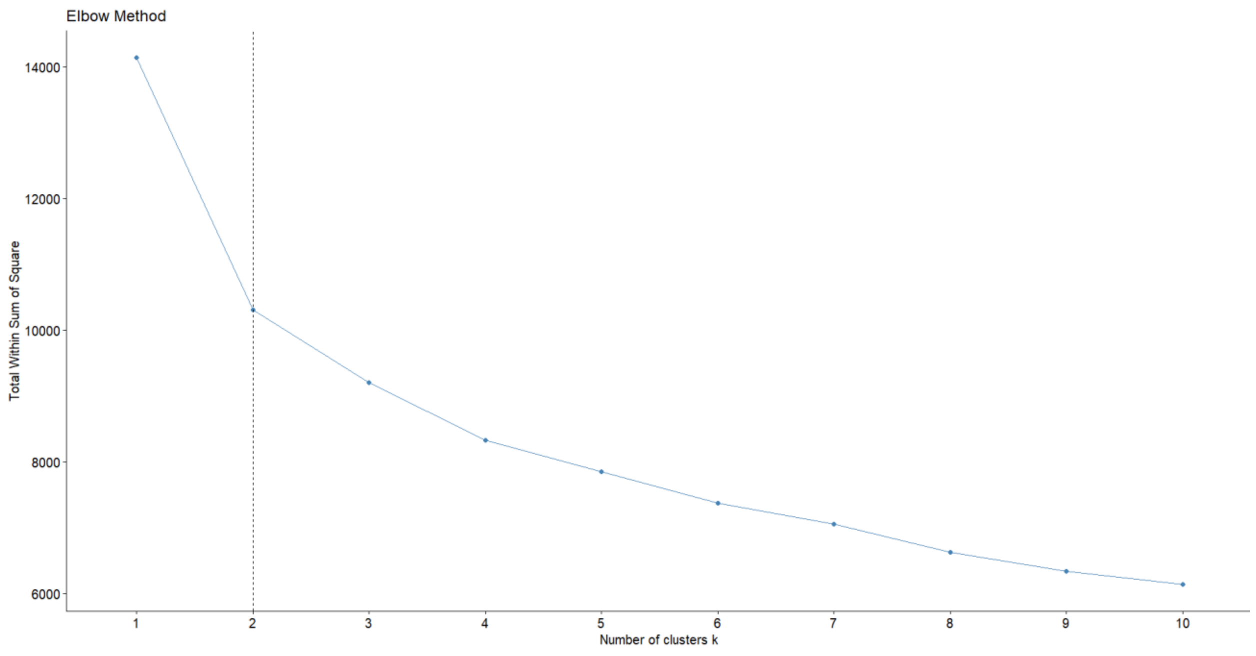
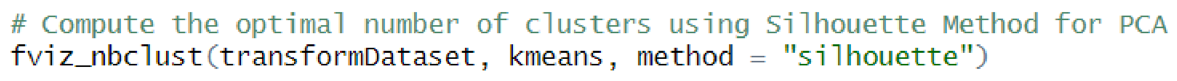


Figure 36: Elbow method result plot for PCA.

This code shows a plot with the within-cluster sum of squares (WSS) for various values of k and uses the elbow technique to get the ideal number of clusters. The output is a figure illustrating the relationship between the WSS and the number of clusters. The optimal number of clusters, or the point at which adding more clusters does not significantly reduce the WSS, is indicated by the "elbow" point in the plot. It's hard to comment on the result without viewing the plot. However, generally, this approach offers a helpful visualisation for identifying the optimal number of clusters.

The number of clusters doesn’t change in the Elbow method after doing PCA. It remains to be seen the same number for k as 2.

<u>Silhouette method.</u>



```
# Compute the optimal number of clusters using Silhouette Method for PCA
fviz_nbclust(transformDataset, kmeans, method = "silhouette")
```

Figure 37: Silhouette method for PCA

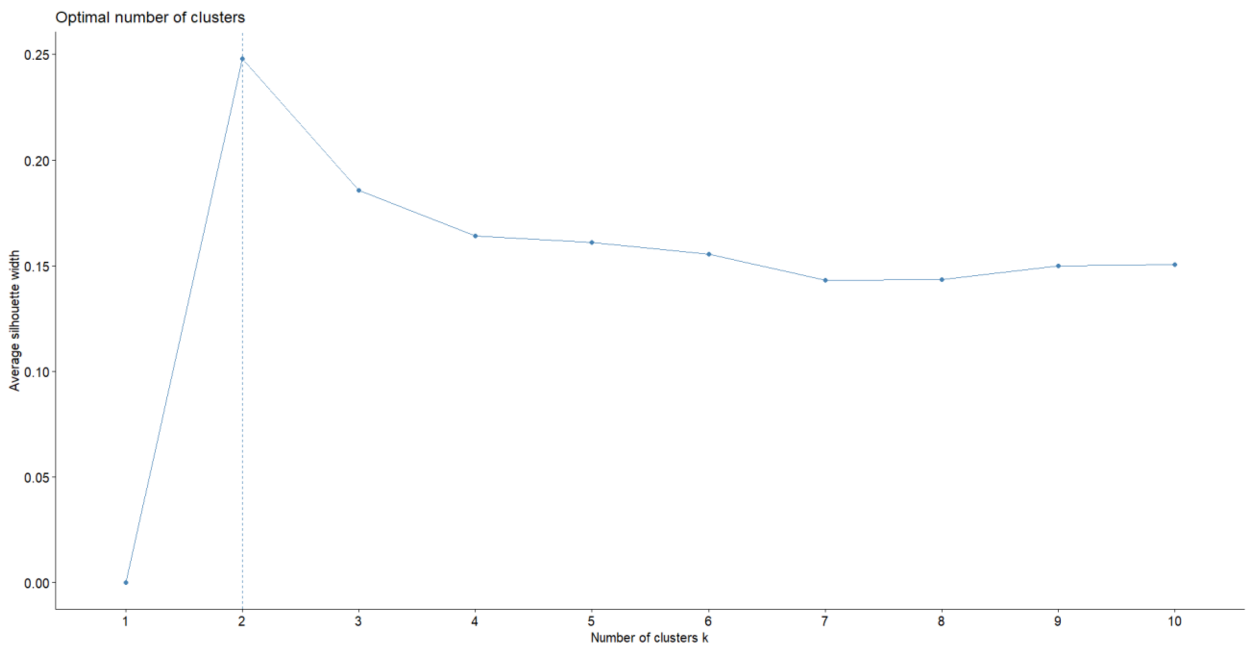
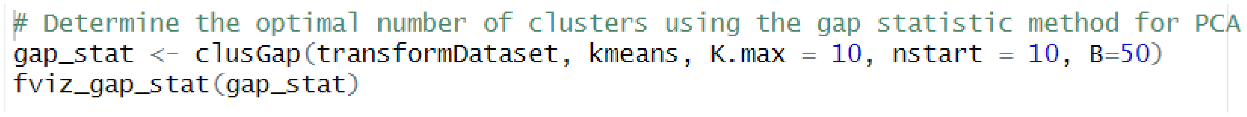


Figure 38: Silhouette method plot result for PCA

A plot showing the average silhouette width for various cluster counts is produced by the fviz_nbclust() function's output when method = "silhouette". The number of clusters with the largest average silhouette width is the number that is considered optimal. In general, better clustering outcomes are indicated by a larger average silhouette width. The fviz_nbclust() function's result suggests that 2 clusters are the optimal number for this data set. The plot's peak at cluster 2, which has the highest average silhouette width, indicates this.

The number of clusters doesn’t change in the Silhouette method after doing PCA. It remains to be seen the same number for k as 2.

Gap statistics Method.



```
# Determine the optimal number of clusters using the gap statistic method for PCA
gap_stat <- clusGap(transformDataset, kmeans, K.max = 10, nstart = 10, B=50)
fviz_gap_stat(gap_stat)
```

Figure 39: Gap statistics method for PCA.

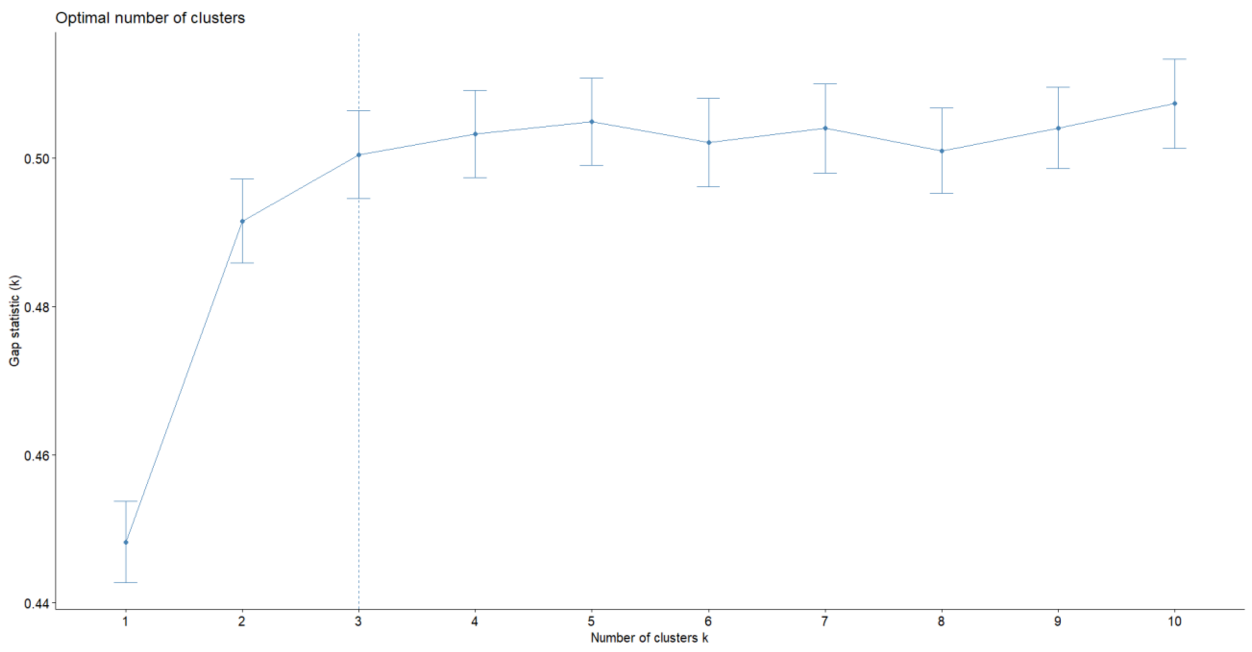


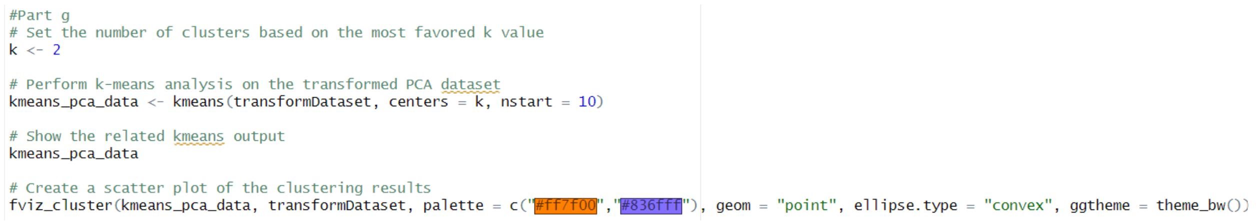
Figure 40: Gap statistics method plot result for PCA

The gap statistic method is the fourth method used to identify the optimal number of clusters. It compares the dataset's within-cluster variation to a null reference distribution. The fviz_gap_stat function is used to visualize the results. Depending on how the standard deviation is estimated, the plot in this instance indicates that 3 clusters would be the ideal number.

The number of clusters changes from 10 to 3 in the Gap Statistic method after doing PCA.

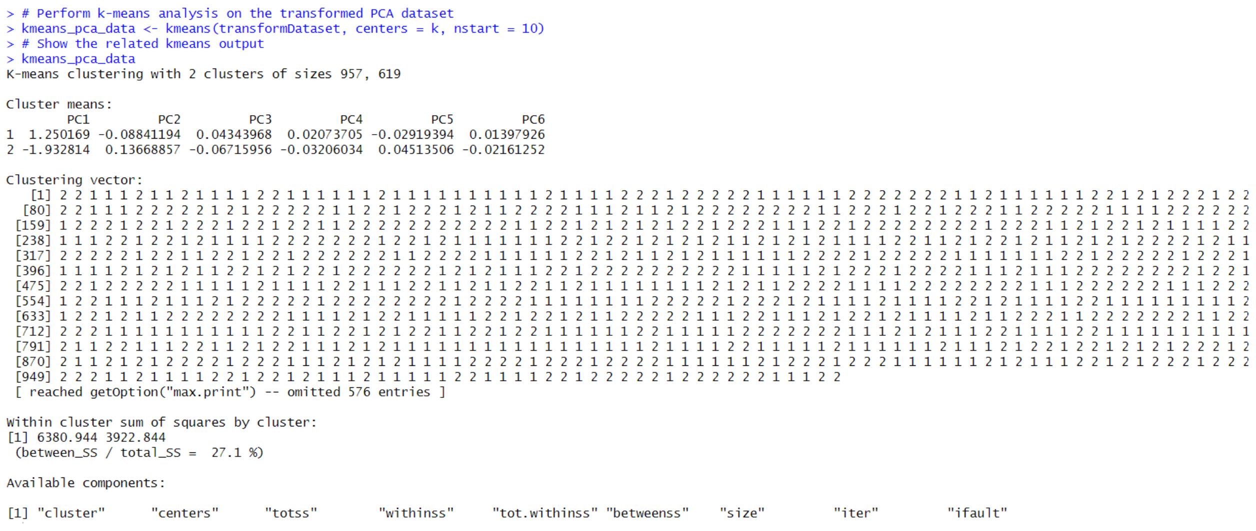
g) Kmeans clustering investigation after PCA.

Using the specified number of clusters and 10 random starts, the code below performs k-means clustering on a scaled wine dataset. The number of clusters is defined as k=2,



```r
#Part g
# Set the number of clusters based on the most favored k value
k <- 2

# Perform k-means analysis on the transformed PCA dataset
kmeans_pca_data <- kmeans(transformDataset, centers = k, nstart = 10)

# Show the related kmeans output
kmeans_pca_data

# Create a scatter plot of the clustering results
fviz_cluster(kmeans_pca_data, transformDataset, palette = c("#ff7f00","#836fff"), geom = "point", ellipse.type = "convex", ggtheme = theme_bw())
```

Figure 41: kmeans() for PCA



```
> # Perform k-means analysis on the transformed PCA dataset
> kmeans_pca_data <- kmeans(transformDataset, centers = k, nstart = 10)
> # Show the related kmeans output
> kmeans_pca_data
K-means clustering with 2 clusters of sizes 957, 619

Cluster means:
        PC1         PC2         PC3         PC4         PC5         PC6
1  1.250169 -0.08841194  0.04343968  0.02073705 -0.02919394  0.01397926
2 -1.932814  0.13668857 -0.06715956 -0.03206034  0.04513506 -0.02161252

Clustering vector:
   [1] 2 2 1 1 1 2 1 1 2 1 1 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 2 2 2 2 2 1 1 1 1 1 1 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 1 2 2 1 2 1 2 2 2 1 2 2
  [80] 2 2 1 1 1 2 2 2 2 2 1 2 1 2 2 2 2 2 1 1 2 2 1 2 2 2 1 2 2 2 1 2 1 1 2 2 2 2 1 1 1 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 2 2 2 1 1 2 2 2 2 2 1 1 1 1 2 2 2 2 2 2
 [159] 1 2 2 2 1 2 2 1 2 2 2 1 2 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 2 1 2 2 1 2 2 2 1 1 1 2 2 1 2 2 2 2 2 2 2 1 2 2 2 1 1 2 1 2 2 1 2 1 1 1 1 2 2
 [238] 1 1 1 2 2 1 2 2 1 2 1 1 1 1 2 2 2 2 2 2 2 1 2 2 1 2 1 1 1 1 1 1 1 2 2 1 2 2 1 2 1 2 1 2 1 1 2 1 2 1 2 1 1 1 1 2 2 1 1 2 1 2 2 1 2 1 1 2 1 2 1 2 2 2 2 1 2 1 1
 [317] 2 2 2 2 2 1 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2 1 1 2 2 2 2 1 1 1 1 1 1 2 2 1 2 1 1 2 1 2 1 1 1 1 1 1 2 2 2 2 1 2 2 2 2 2 1 1 1 1 1 1 1 2 2 1 2 2 2 2 2 1 2 2 2 1
 [396] 1 1 1 1 2 1 2 1 2 1 1 2 2 1 2 1 2 2 1 2 2 2 2 2 2 1 2 1 2 1 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 2 1 2 2 2 1 2 2 2 2 1 1 1 2 1 1 1 2 2 2 2 2 2 2 2 1 2 2 1
 [475] 2 2 1 2 2 2 2 2 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 2 2 1 1 2 1 1 2 2 1 1 2 1 1 1 1 1 1 1 1 2 1 2 1 1 2 2 2 2 1 1 1 1 2 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 2 1 1 1 2 2 2
 [554] 1 2 2 1 1 1 2 1 1 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 1 1 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 1 2 1 1 1 1 2 1 1 1 1 2 2 1 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2
 [633] 1 2 2 1 2 1 1 2 2 2 2 2 2 2 2 1 1 1 1 2 1 2 2 1 2 2 1 2 1 1 2 2 2 2 2 1 1 1 2 2 1 1 1 2 1 2 2 1 2 1 1 1 1 2 1 1 2 2 2 1 1 2 2 2 2 2 2 2 1 1 2 2
 [712] 2 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 2 2 1 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 2 2 2 2 2 2 2 1 1 1 2 1 2 1 1 1 2 2 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1
 [791] 2 1 1 2 2 1 1 1 2 2 1 1 2 1 2 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 2 2 1 2 2 1 2 1 2 1 2 2 2 1 2
 [870] 2 1 1 2 1 2 1 2 2 2 2 1 2 2 2 1 1 1 2 1 2 1 2 1 1 1 1 2 2 2 2 1 2 2 2 1 2 2 2 2 1 1 1 1 1 1 2 1 2 2 2 1 2 2 2 1 1 1 1 1 1 2 1 2 1 1 1 2 2 1 2 1 2 2 2 1 2 2 2
 [949] 2 2 2 1 1 2 1 1 1 1 2 2 1 2 2 1 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 1 1 2 2
 [ reached getOption("max.print") -- omitted 576 entries ]

Within cluster sum of squares by cluster:
[1] 6380.944 3922.844
 (between_SS / total_SS =  27.1 %)

Available components:

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

Figure 42: kmeans() result for PCA

After the PCA process of clustering, the Kmeans classes can be visualised through plotting.

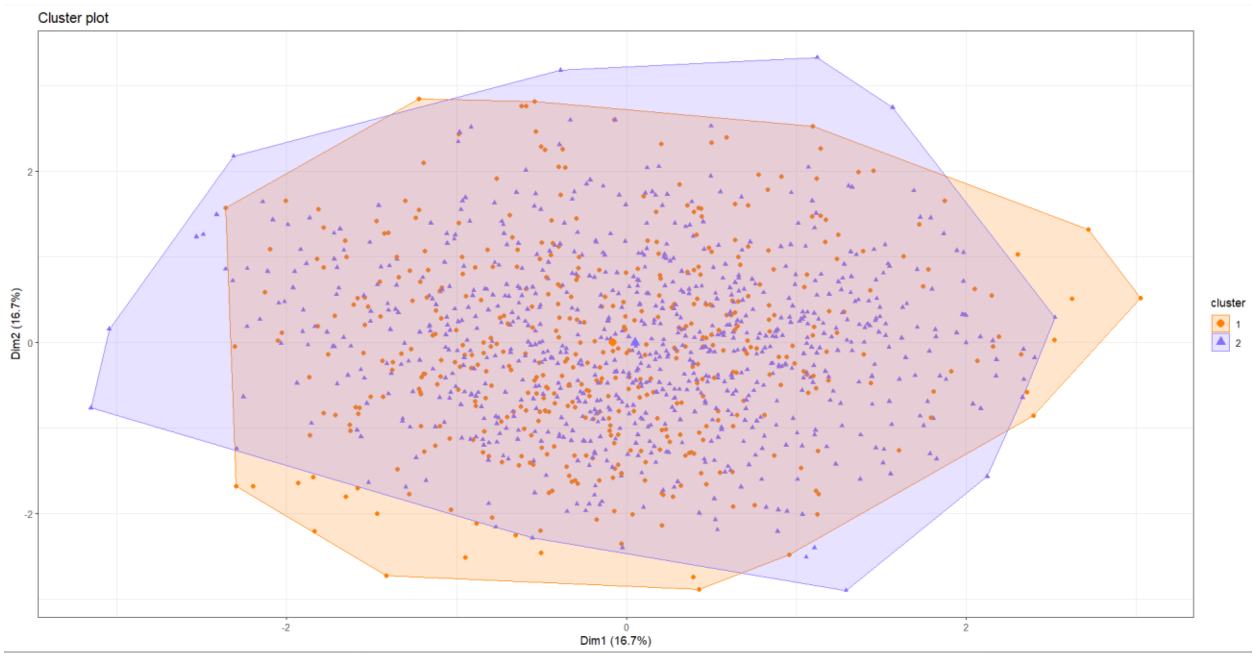
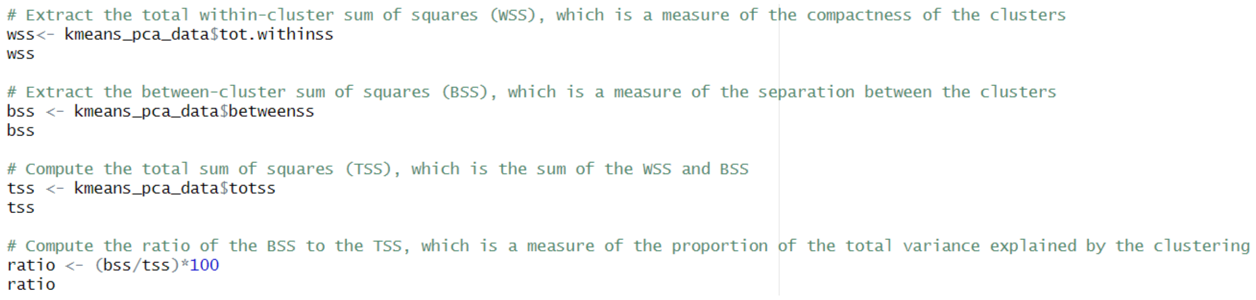


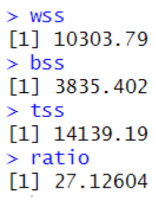
Figure 43: Visualization of the clustering in PCA results using the factoextra package k=2

Additionally, the code below extracts the between-cluster sum of squares (BSS), which measures the distance between the clusters, and the total within-cluster sum of squares (WSS), which measures how compact the clusters are. Next, the total sum of squares (TSS) and the ratio of the BSS to the TSS are calculated. The ratio can be expressed as a percentage by multiplying the result by 100. This indicates the percentage of the total variance connected to the clustering in PCA.



```
# Extract the total within-cluster sum of squares (WSS), which is a measure of the compactness of the clusters
wss<- kmeans_pca_data$tot.withinss
wss

# Extract the between-cluster sum of squares (BSS), which is a measure of the separation between the clusters
bss <- kmeans_pca_data$betweenss
bss

# Compute the total sum of squares (TSS), which is the sum of the WSS and BSS
tss <- kmeans_pca_data$totss
tss

# Compute the ratio of the BSS to the TSS, which is a measure of the proportion of the total variance explained by the clustering
ratio <- (bss/tss)*100
ratio
```

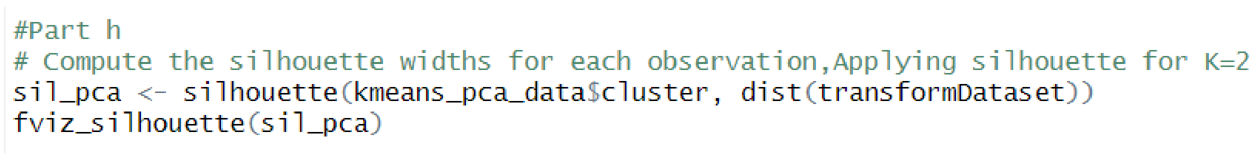
Figure 44: WSS, BSS, TSS & ratio in PCA



```
> wss
[1] 10303.79
> bss
[1] 3835.402
> tss
[1] 14139.19
> ratio
[1] 27.12604
```

Figure 45: WSS, BSS, TSS & ratio results in PCA

h) Silhouette plot analysis of kmeans attempt after PCA.



```
#Part h
# Compute the silhouette widths for each observation,Applying silhouette for K=2
sil_pca <- silhouette(kmeans_pca_data$cluster, dist(transformDataset))
fviz_silhouette(sil_pca)
```

Figure 46: Silhouette Plot in PCA

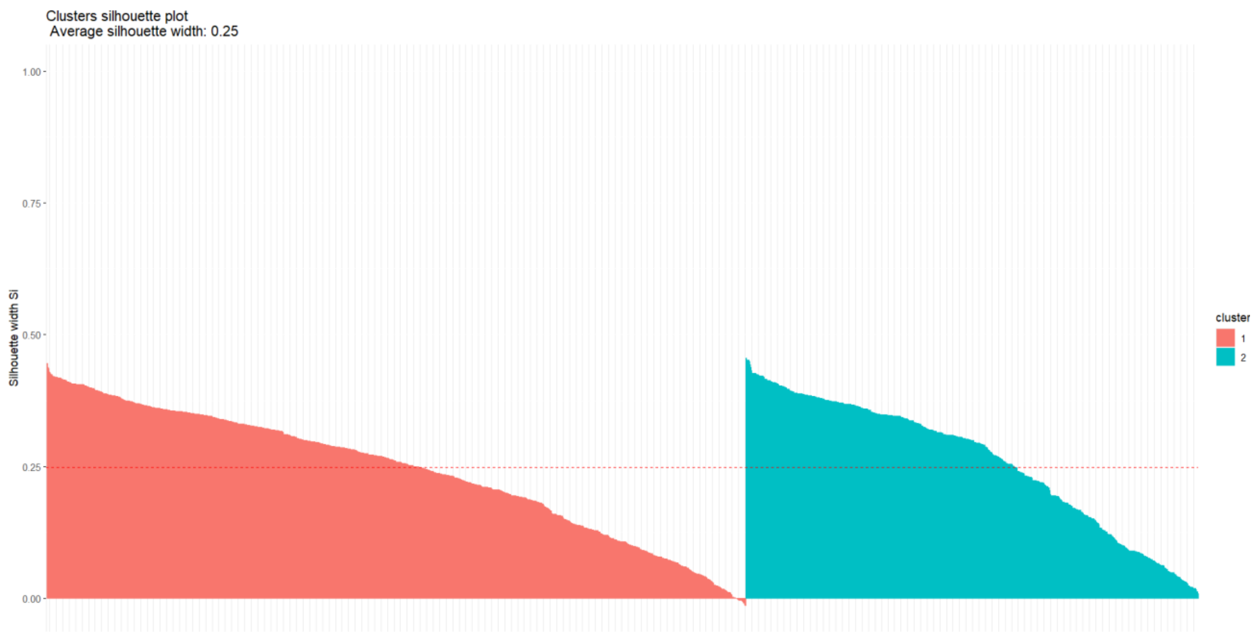
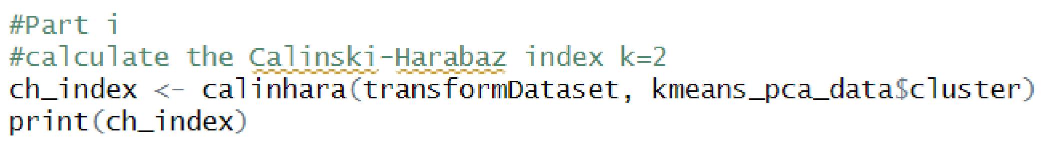


Figure 47: Silhouette Plot result in PCA.

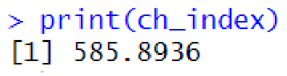
The clusters may overlap to some extent, as indicated by the average silhouette width of 0.25, which suggests they are not well separated. This could indicate that the data are not successfully divided into separate categories by the clustering algorithm according to the features used.

i) Calinski-Harabasz Index



```
#Part i
#calculate the Calinski-Harabaz index k=2
ch_index <- calinhara(transformDataset, kmeans_pca_data$cluster)
print(ch_index)
```

Figure 48: Calinski-Harabasz index



```
> print(ch_index)
[1] 585.8936
```

Figure 49: Calinski-Harabasz index result

The ratio of between-cluster dispersion to within-cluster dispersion is measured by the Calinski-Harabasz index, an internal cluster validation metric that evaluates the quality of a clustering solution. The ratio of the total number of observations minus the number of clusters minus one is multiplied by the sum of squared distances within each cluster to the sum of squared distances between cluster centroids to determine the index. The greater the Calinski-Harabasz index value, the better the clustering solution.
Because it is reasonably high, the clustering solution is high quality, as indicated by its Calinski-Harabasz index score of 585.8936.

2. FINANCIAL FORECASTING PART.

This coursework aims to forecast the USD/EUR exchange rate for the next day using a multilayer neural network (MLP-NN). Only the "autoregressive" method is needed for this task. Up to the (t-4) level, different input vectors will be used to experiment with the AR approach. After the I/O matrices have been normalised, statistical indices like RMSE, MAE, MAPE, and MAPE will be used to assess how well the networks tested. A comparison table will be made to evaluate the different models' performance. Then, the top-performing networks with one and two hidden layers will be determined by adding all the weight parameters in each network.

a) Type of Input variables used in MLP models for exchange rate forecasting.

The input vector is the set of variables input into the neural network model to make predictions. The input vector is essential because it provides the data the model may use to make predictions. In the financial domain, input vectors are created using the following schemes/methods, especially for exchange rate forecasting:

1. ***Autoregressive (AR) approach:***

   This is the approach that the task description refers to. The exchange rate's time-delayed values make up the input vector for this procedure. For instance, if the input vector is defined as (t-1, t-2, t-3, t-4), the neural network will forecast the next value based on the last four exchange rate values.

2. ***Exogenous variables approach:***

   This approach includes additional relevant economic variables or indicators in the input vector and historical exchange rate values that may impact the exchange rate. These might include GDP growth rates, trade balances, interest rates, and inflation rates.

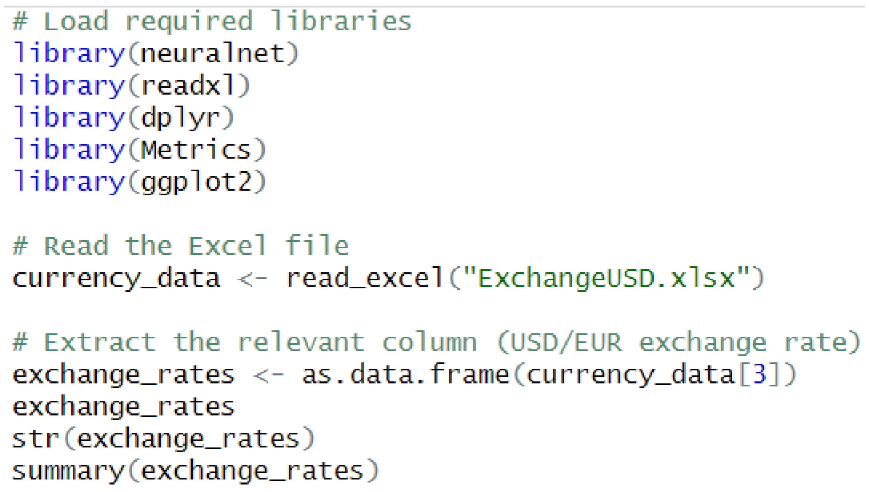
3. ***Technical indicators approach:***

   This approach uses a variety of technical analysis indicators as input variables that are obtained from previous exchange rate data. Moving averages, momentum indicators, oscillators, etc., are a few examples. These indicators identify patterns and trends in the data that could influence future changes in exchange rates.

4. ***Moving Average (MA) approach:***
   The MA approach uses lagged values of the forecast errors compared to the AR approach's use of lagged values of the target variable. This method seeks to smooth out noise in the data and capture temporary changes.

b) Creating Input/Output matrices.

I started by loading data from the "ExchangeUSD.xlsx" file and isolated the USD/EUR column.

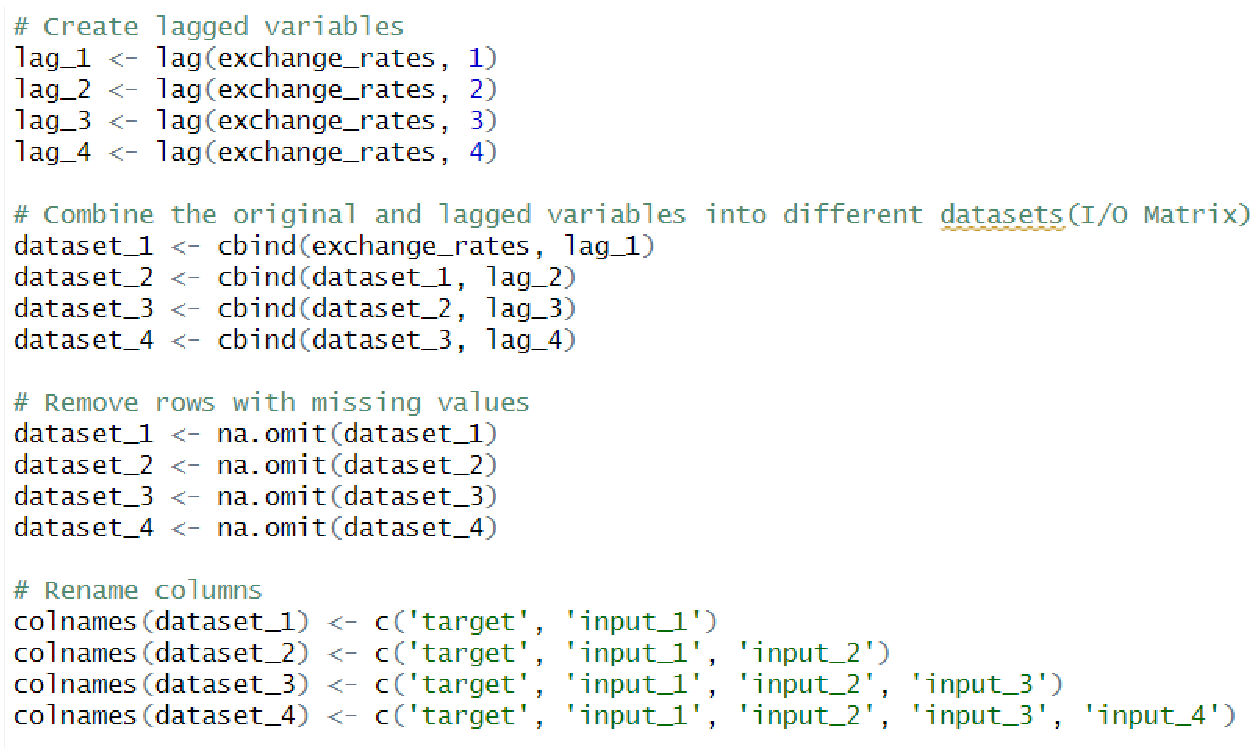


```r
# Load required libraries
library(neuralnet)
library(readxl)
library(dplyr)
library(Metrics)
library(ggplot2)

# Read the Excel file
currency_data <- read_excel("ExchangeUSD.xlsx")

# Extract the relevant column (USD/EUR exchange rate)
exchange_rates <- as.data.frame(currency_data[3])
exchange_rates
str(exchange_rates)
summary(exchange_rates)
```

Figure 50: NN loading data

The process of creating the IO Matrices is covered in this section. 4 different datasets are produced by first creating 4 lag data sets and then combining them with the original dataset. The columns have new names, and the null values are eliminated.

Lagged variables, or time-delayed values, of the exchange rates up to 4 lags (*lag_1, lag_2, lag_3, lag_4*) are created in this section. The original exchange rates (*exchange_rates*) and these lag variables are merged to produce the input/output matrices for the MLP models, which are then divided into many datasets (*dataset_1, dataset_2, dataset_3, dataset_4*). Each dataset has a target variable (the exchange rate to be predicted) in the first column and input variables (delayed exchange rates) in the remaining columns. For clarity, we additionally rename the columns and eliminate any rows that have missing data.



```r
# Create lagged variables
lag_1 <- lag(exchange_rates, 1)
lag_2 <- lag(exchange_rates, 2)
lag_3 <- lag(exchange_rates, 3)
lag_4 <- lag(exchange_rates, 4)

# Combine the original and lagged variables into different datasets(I/O Matrix)
dataset_1 <- cbind(exchange_rates, lag_1)
dataset_2 <- cbind(dataset_1, lag_2)
dataset_3 <- cbind(dataset_2, lag_3)
dataset_4 <- cbind(dataset_3, lag_4)

# Remove rows with missing values
dataset_1 <- na.omit(dataset_1)
dataset_2 <- na.omit(dataset_2)
dataset_3 <- na.omit(dataset_3)
dataset_4 <- na.omit(dataset_4)

# Rename columns
colnames(dataset_1) <- c('target', 'input_1')
colnames(dataset_2) <- c('target', 'input_1', 'input_2')
colnames(dataset_3) <- c('target', 'input_1', 'input_2', 'input_3')
colnames(dataset_4) <- c('target', 'input_1', 'input_2', 'input_3', 'input_4')
```

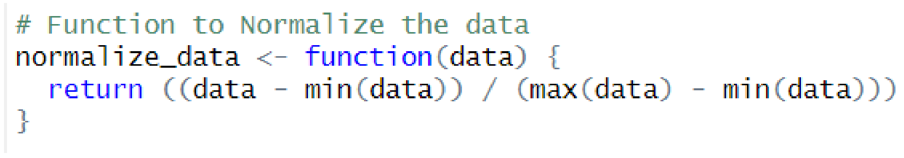
Figure 51: NN IO Matrices

c) Data Normalization.

In order to improve the convergence and performance of the training process, data normalization is a standard procedure for neural networks. It does this by ensuring that the input variables have a similar scale. Normalization is crucial in MLPs because the hidden layers' activation functions—like sigmoid or tanh—are frequently non-linear and have outputs that are sensitive to the input range.

We can stop some input variables from predominating over others and make sure the MLP can learn from all the input characteristics by normalizing the data to a common range, which is usually between 0 and 1.

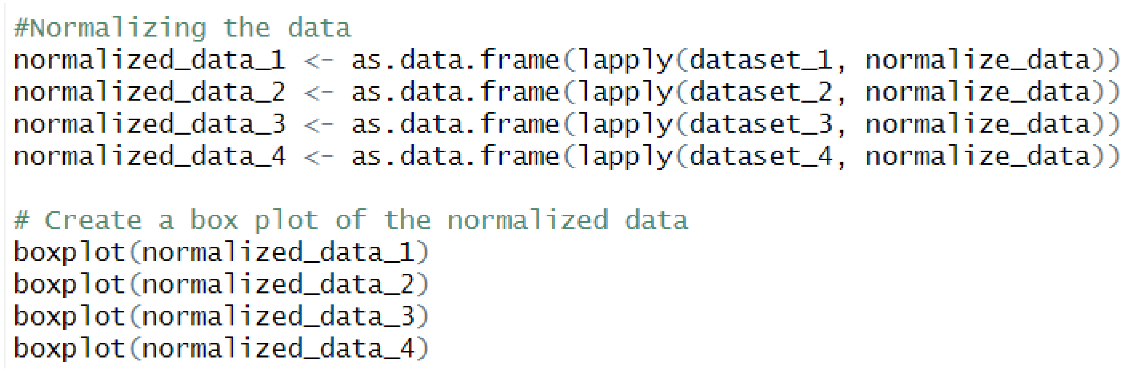
Below is the function used to normalize the data



```r
# Function to Normalize the data
normalize_data <- function(data) {
  return ((data - min(data)) / (max(data) - min(data)))
}
```

Figure 52: NN Normalize function

Below shows the data gets normalized



```
#Normalizing the data
normalized_data_1 <- as.data.frame(lapply(dataset_1, normalize_data))
normalized_data_2 <- as.data.frame(lapply(dataset_2, normalize_data))
normalized_data_3 <- as.data.frame(lapply(dataset_3, normalize_data))
normalized_data_4 <- as.data.frame(lapply(dataset_4, normalize_data))

# Create a box plot of the normalized data
boxplot(normalized_data_1)
boxplot(normalized_data_2)
boxplot(normalized_data_3)
boxplot(normalized_data_4)
```

Figure 53: NN Normalized Data

Below is a box plot of one data set after normalization in order,

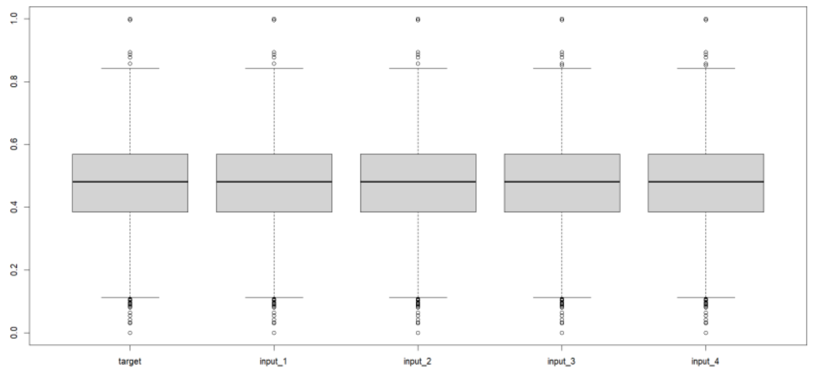
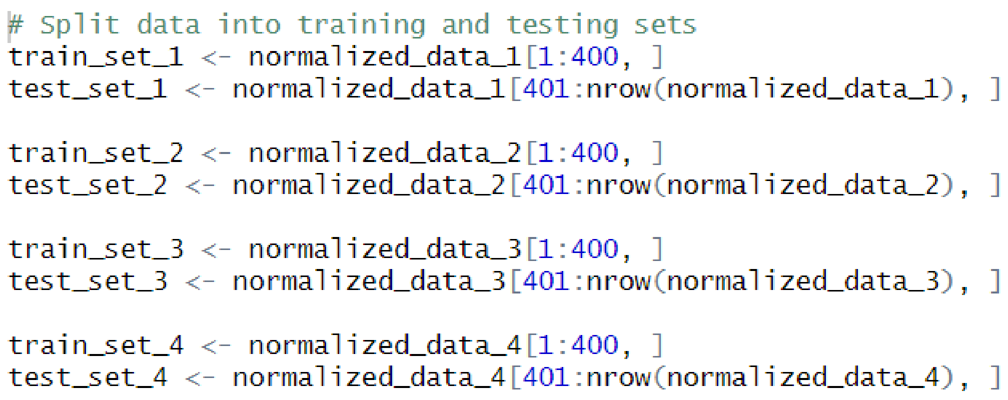


Figure 54: NN One normalized data set boxplot

d) Experimenting with MLP Models.

As given in the coursework description out of 500 rows of data 400 rows of data were used as a training data set and the remaining as a testing data set

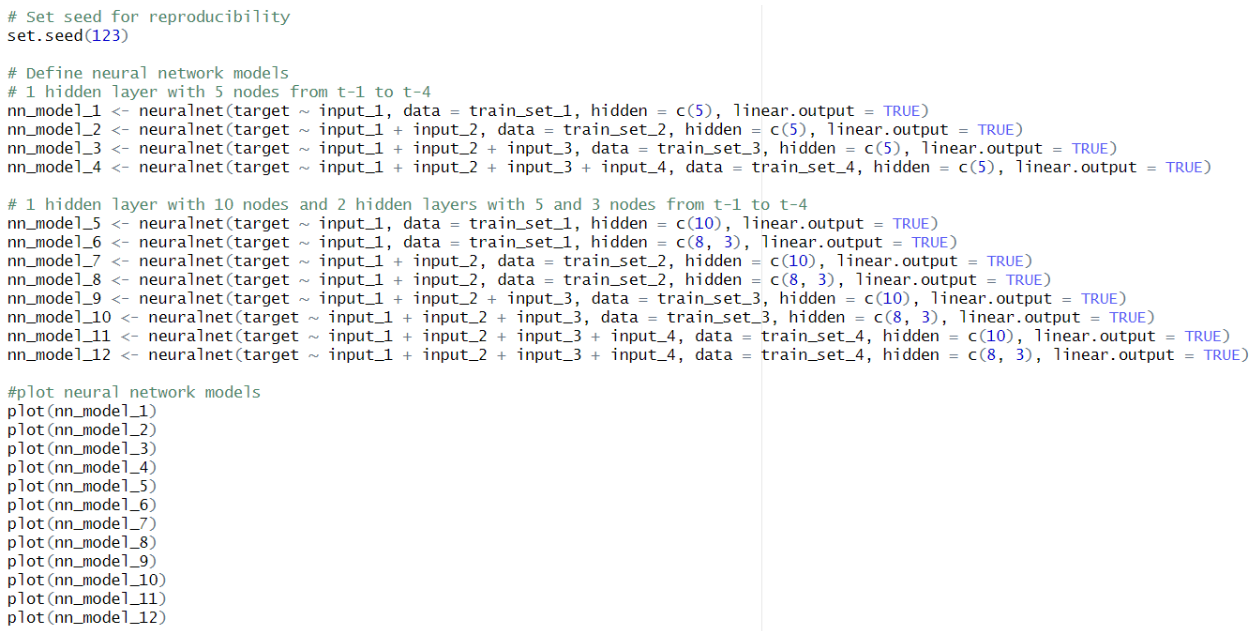
So first we split the data into training and test sets as below,



```r
# Split data into training and testing sets
train_set_1 <- normalized_data_1[1:400, ]
test_set_1 <- normalized_data_1[401:nrow(normalized_data_1), ]

train_set_2 <- normalized_data_2[1:400, ]
test_set_2 <- normalized_data_2[401:nrow(normalized_data_2), ]

train_set_3 <- normalized_data_3[1:400, ]
test_set_3 <- normalized_data_3[401:nrow(normalized_data_3), ]

train_set_4 <- normalized_data_4[1:400, ]
test_set_4 <- normalized_data_4[401:nrow(normalized_data_4), ]
```

Figure 55: Split the data into training and test sets

The below code block shows training various MLP models,



```r
# Set seed for reproducibility
set.seed(123)

# Define neural network models
# 1 hidden layer with 5 nodes from t-1 to t-4
nn_model_1 <- neuralnet(target ~ input_1, data = train_set_1, hidden = c(5), linear.output = TRUE)
nn_model_2 <- neuralnet(target ~ input_1 + input_2, data = train_set_2, hidden = c(5), linear.output = TRUE)
nn_model_3 <- neuralnet(target ~ input_1 + input_2 + input_3, data = train_set_3, hidden = c(5), linear.output = TRUE)
nn_model_4 <- neuralnet(target ~ input_1 + input_2 + input_3 + input_4, data = train_set_4, hidden = c(5), linear.output = TRUE)

# 1 hidden layer with 10 nodes and 2 hidden layers with 5 and 3 nodes from t-1 to t-4
nn_model_5 <- neuralnet(target ~ input_1, data = train_set_1, hidden = c(10), linear.output = TRUE)
nn_model_6 <- neuralnet(target ~ input_1, data = train_set_1, hidden = c(8, 3), linear.output = TRUE)
nn_model_7 <- neuralnet(target ~ input_1 + input_2, data = train_set_2, hidden = c(10), linear.output = TRUE)
nn_model_8 <- neuralnet(target ~ input_1 + input_2, data = train_set_2, hidden = c(8, 3), linear.output = TRUE)
nn_model_9 <- neuralnet(target ~ input_1 + input_2 + input_3, data = train_set_3, hidden = c(10), linear.output = TRUE)
nn_model_10 <- neuralnet(target ~ input_1 + input_2 + input_3, data = train_set_3, hidden = c(8, 3), linear.output = TRUE)
nn_model_11 <- neuralnet(target ~ input_1 + input_2 + input_3 + input_4, data = train_set_4, hidden = c(10), linear.output = TRUE)
nn_model_12 <- neuralnet(target ~ input_1 + input_2 + input_3 + input_4, data = train_set_4, hidden = c(8, 3), linear.output = TRUE)

#plot neural network models
plot(nn_model_1)
plot(nn_model_2)
plot(nn_model_3)
plot(nn_model_4)
plot(nn_model_5)
plot(nn_model_6)
plot(nn_model_7)
plot(nn_model_8)
plot(nn_model_9)
plot(nn_model_10)
plot(nn_model_11)
plot(nn_model_12)
```

Figure 56: Training various MLP models

So, we define and train multiple MLP models using the neuralnet function from the neuralnet package in this section. We test with a range of internal network architectures, including the number of hidden layers and nodes, as well as varied input vectors (from t-1 to t-4 lagged data). The hidden argument like c(5) for a single hidden layer with 5 nodes, c(10) for a single hidden layer with 10 nodes and c(8, 3) for two hidden layers with eight and three nodes, respectively. They describe the structure of the hidden layers. We want a linear output layer, which is appropriate for regression jobs like exchange rate forecasting, as indicated by the linear.output = TRUE option.

Below are the 12 MLP NN models,

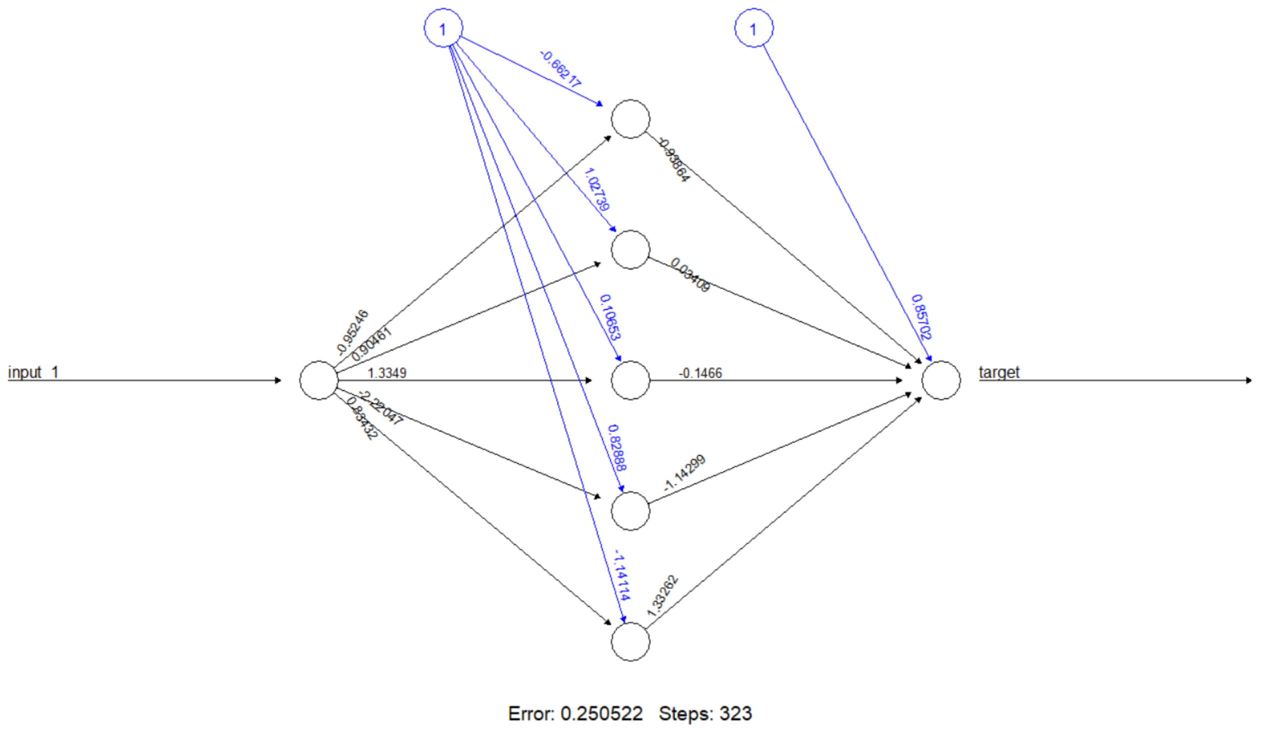


Figure 57: MLP NN model 1

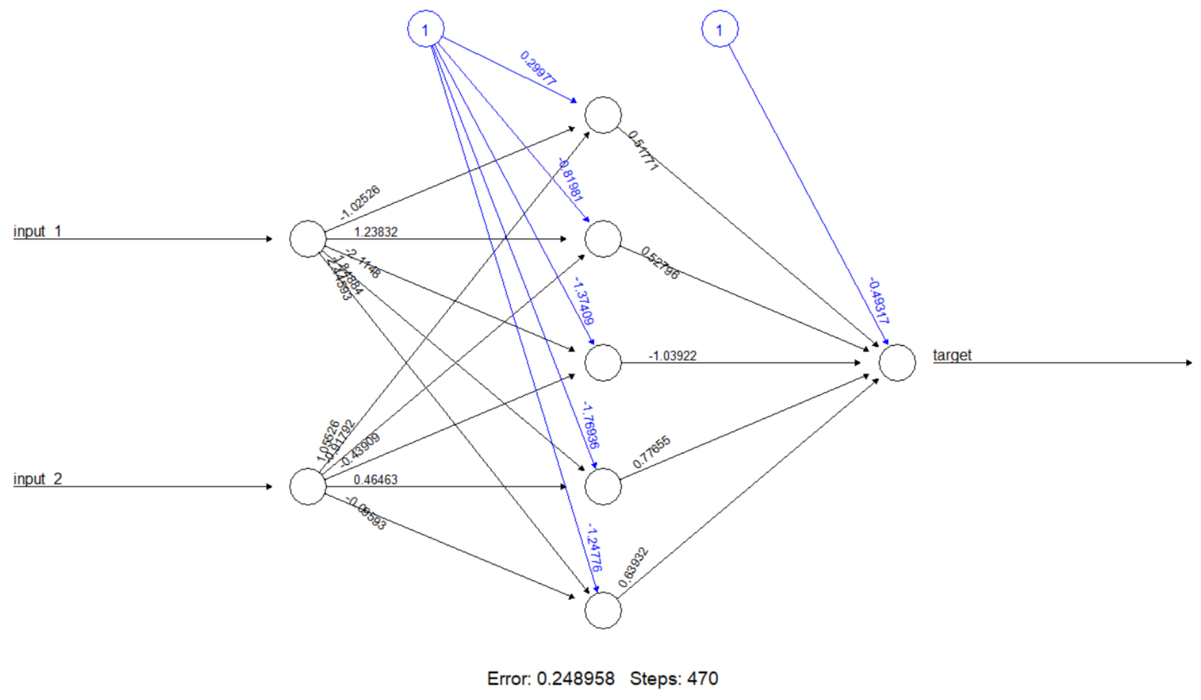


Figure 58: MLP NN model 2

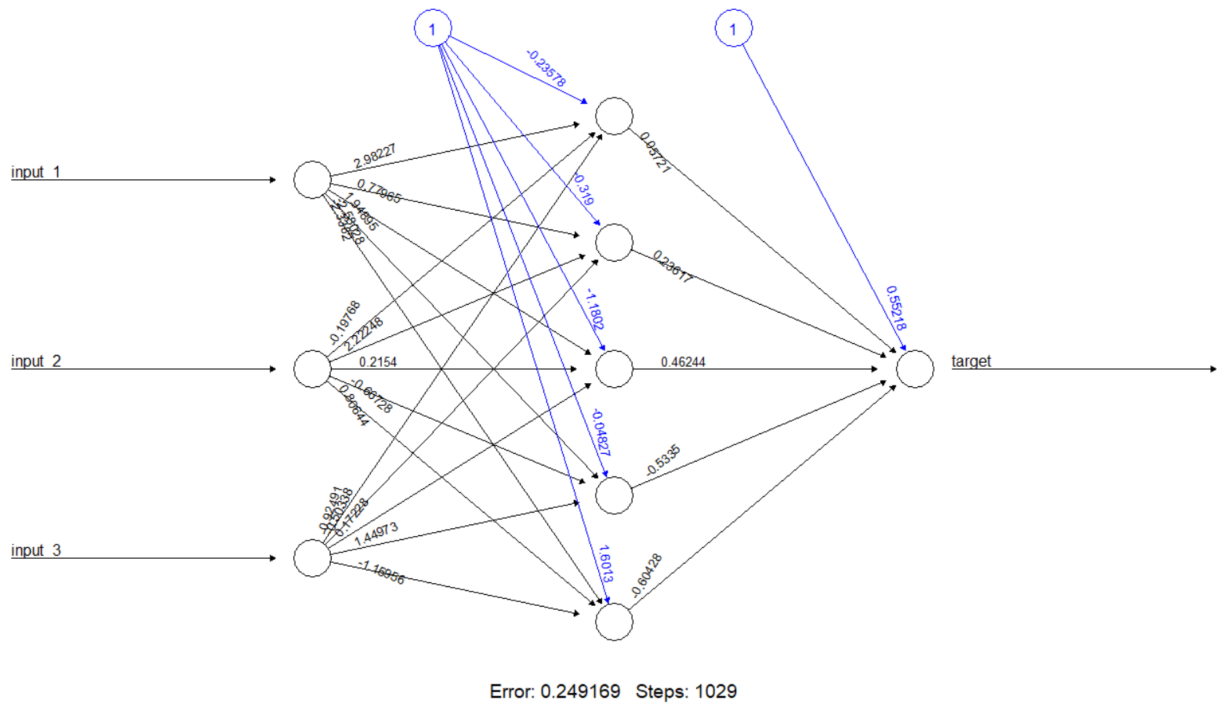


Figure 59: MLP NN model 3

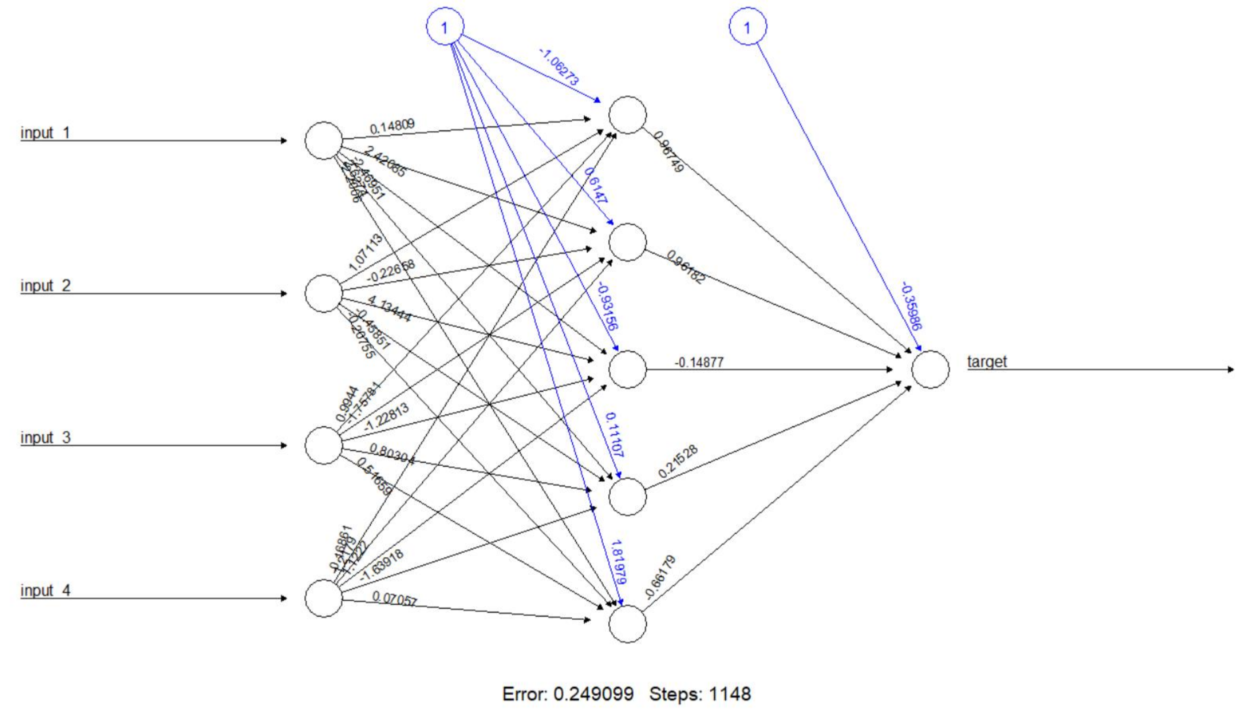


Figure 60: MLP NN model 4

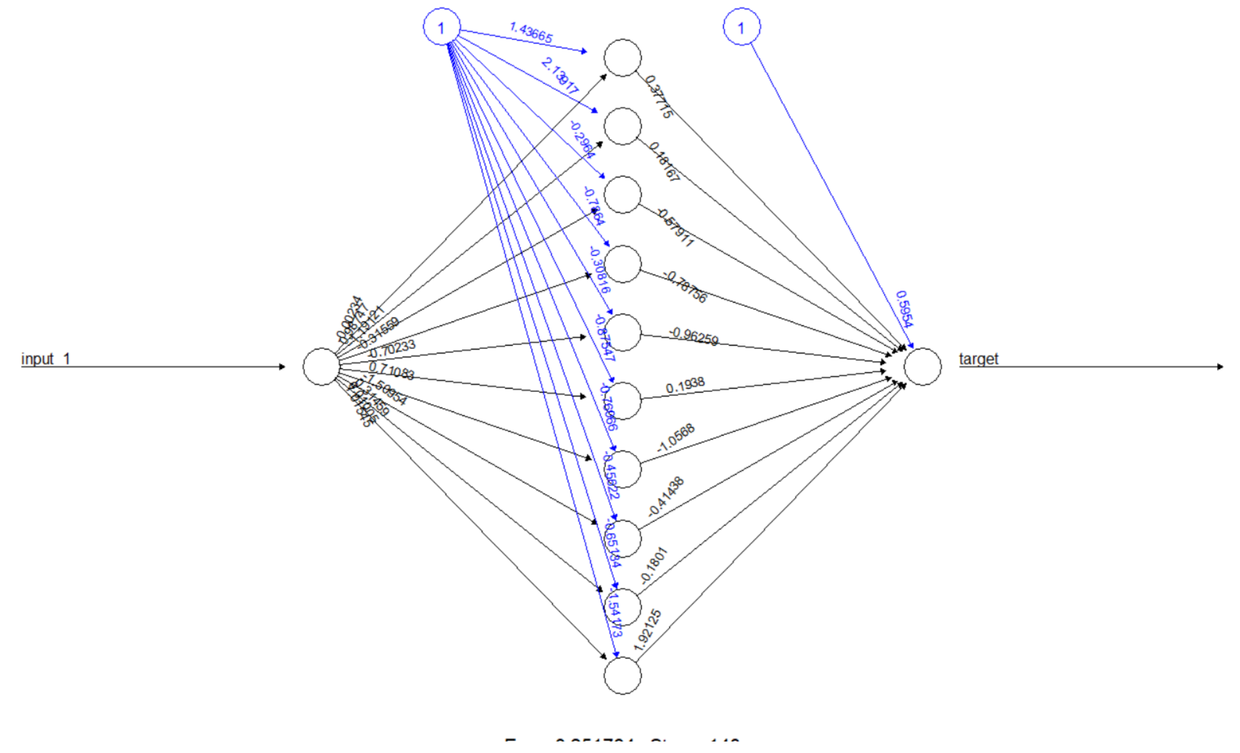


Figure 61: MLP NN model 5

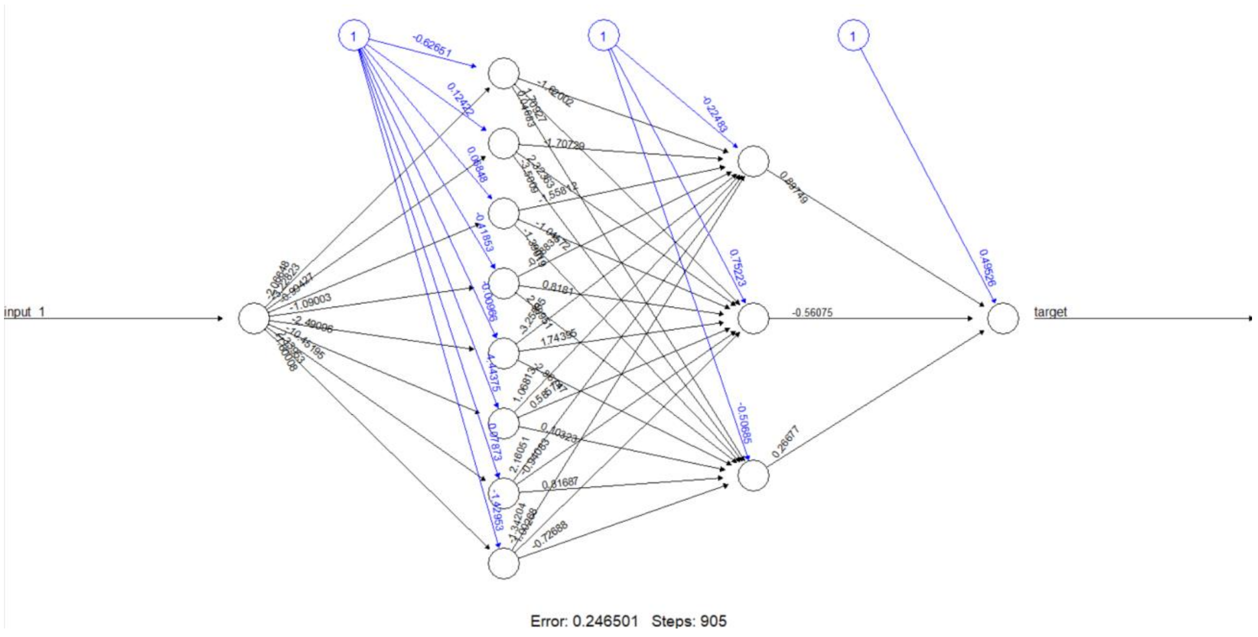


Figure 62: MLP NN model 6

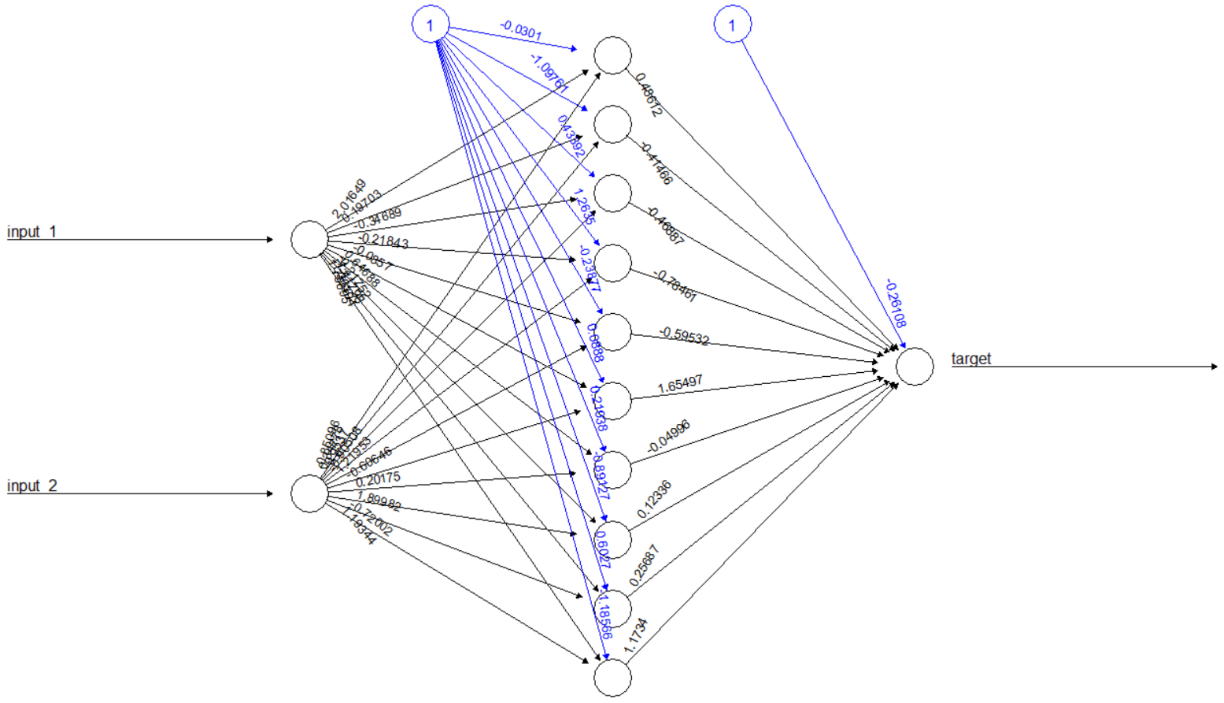


Figure 63: MLP NN model 7

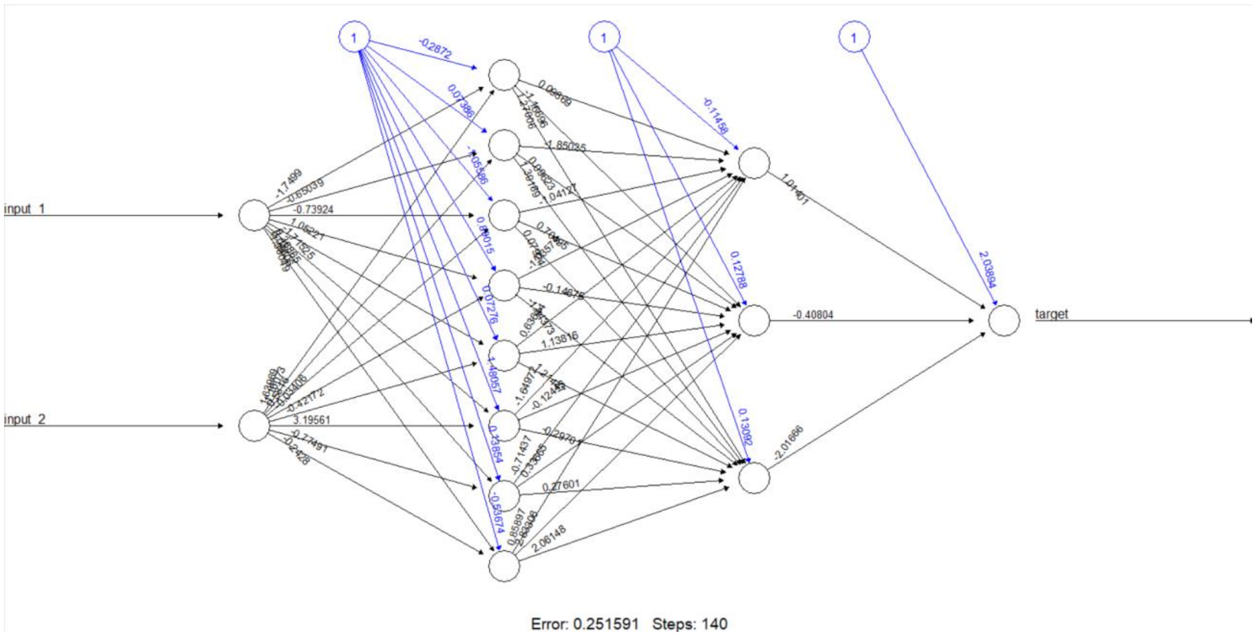


Figure 64: MLP NN model 8

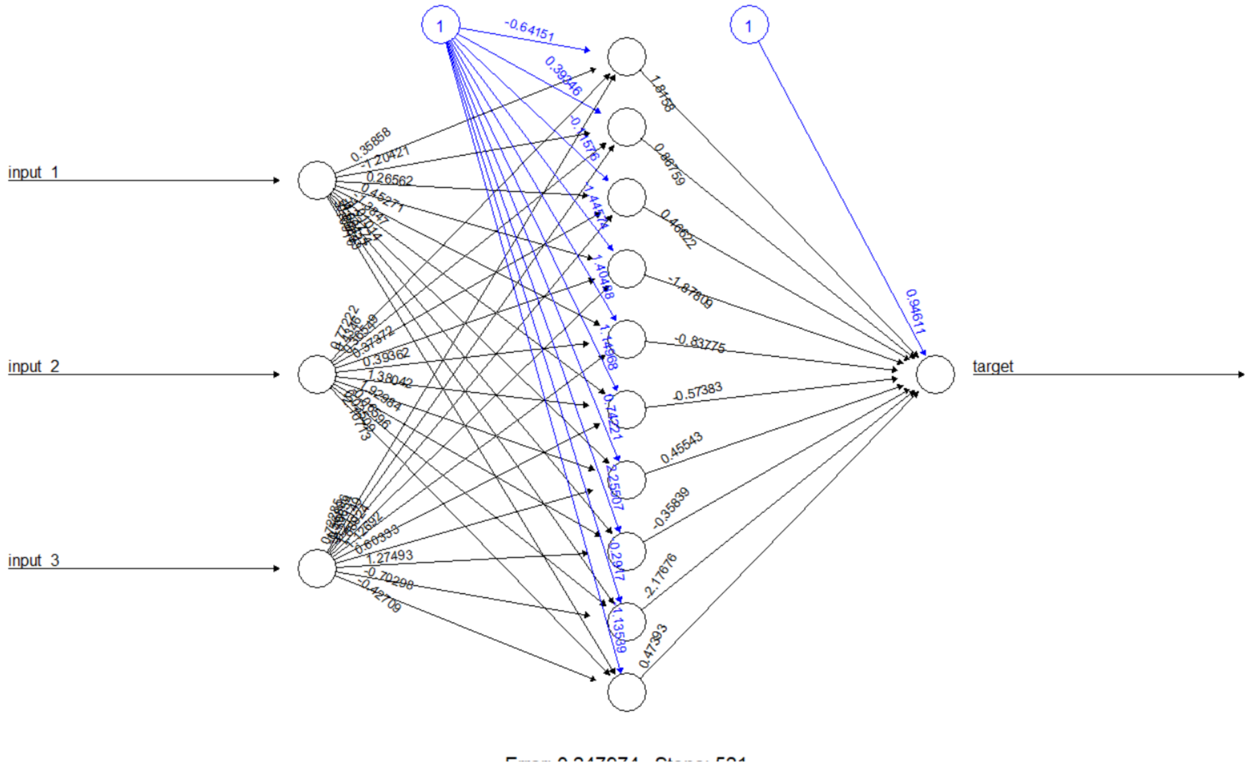


Figure 65: MLP NN model 9

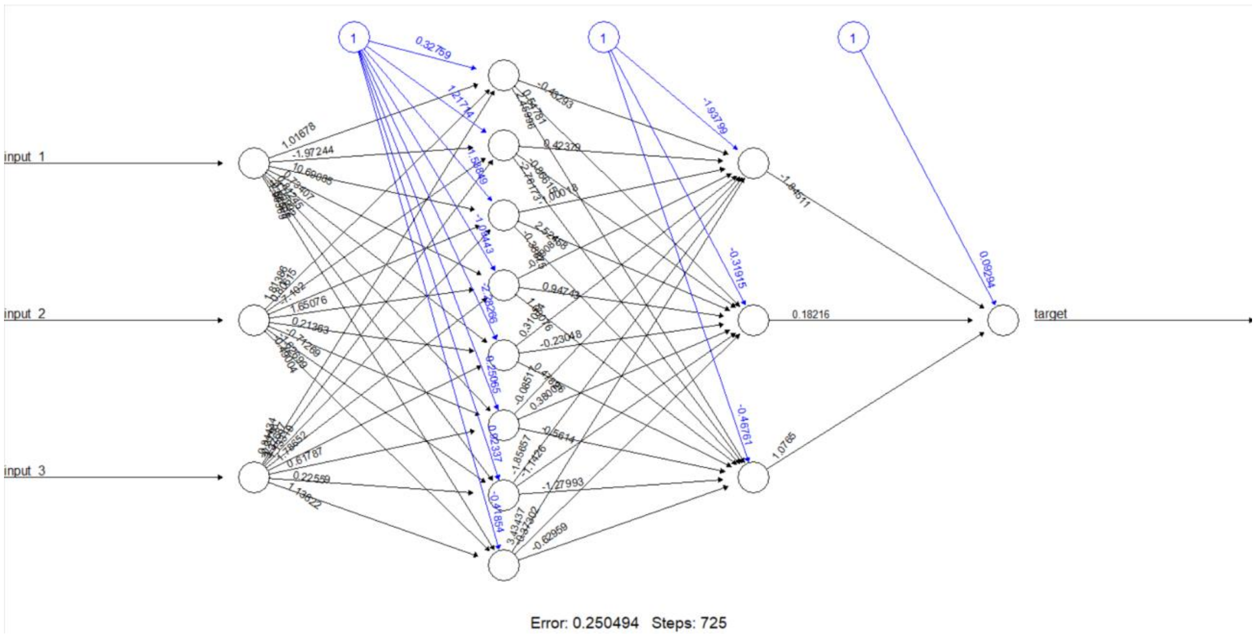


Figure 66: MLP NN model 10

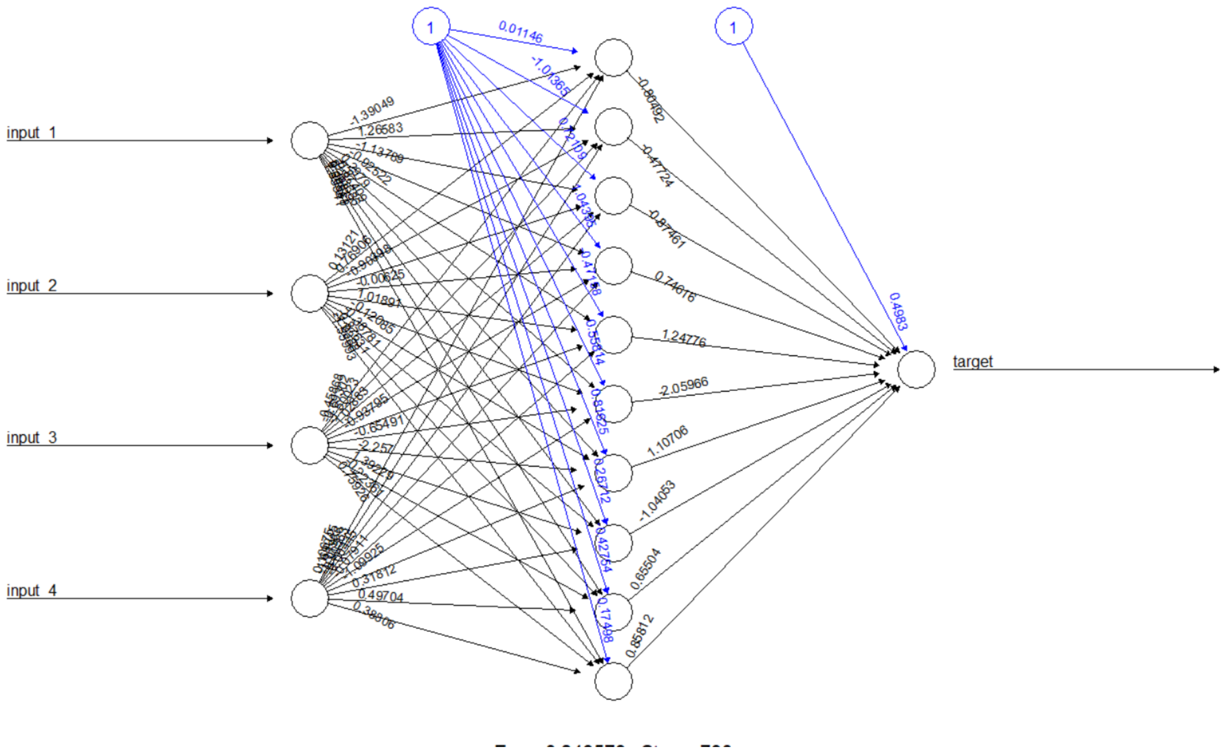


Figure 67: MLP NN model 11

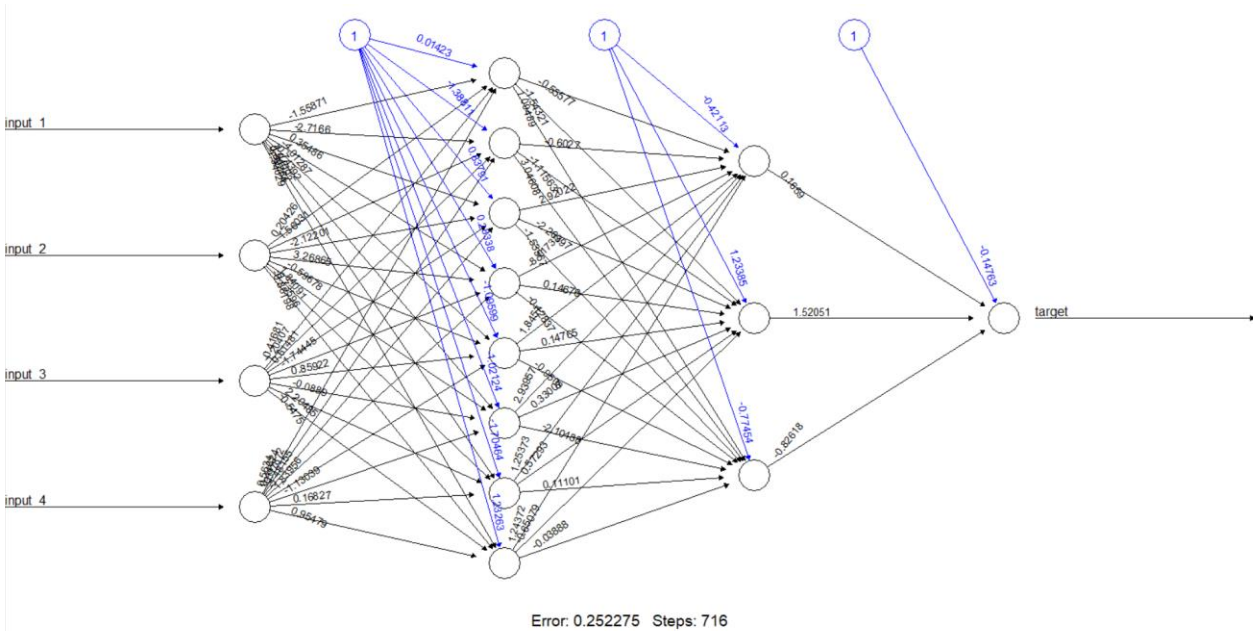
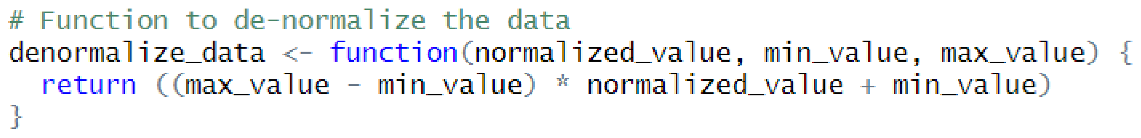


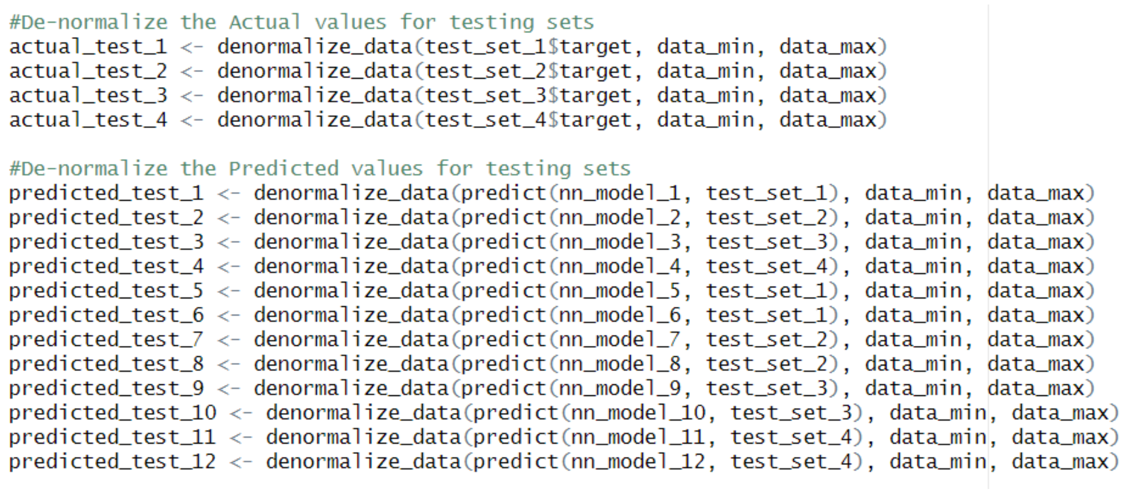
Figure 68: MLP NN model 12

Testing the various MLPs that were previously trained is displayed in the code block below. It has a function that denormalizes the predicted and actual values in the test set. Using the minimum and maximum input values, the predicted and actual values are denormalized.



```
# Function to de-normalize the data
denormalize_data <- function(normalized_value, min_value, max_value) {
  return ((max_value - min_value) * normalized_value + min_value)
}
```

Figure 69: denormalized function



```
#De-normalize the Actual values for testing sets
actual_test_1 <- denormalize_data(test_set_1$target, data_min, data_max)
actual_test_2 <- denormalize_data(test_set_2$target, data_min, data_max)
actual_test_3 <- denormalize_data(test_set_3$target, data_min, data_max)
actual_test_4 <- denormalize_data(test_set_4$target, data_min, data_max)

#De-normalize the Predicted values for testing sets
predicted_test_1 <- denormalize_data(predict(nn_model_1, test_set_1), data_min, data_max)
predicted_test_2 <- denormalize_data(predict(nn_model_2, test_set_2), data_min, data_max)
predicted_test_3 <- denormalize_data(predict(nn_model_3, test_set_3), data_min, data_max)
predicted_test_4 <- denormalize_data(predict(nn_model_4, test_set_4), data_min, data_max)
predicted_test_5 <- denormalize_data(predict(nn_model_5, test_set_1), data_min, data_max)
predicted_test_6 <- denormalize_data(predict(nn_model_6, test_set_1), data_min, data_max)
predicted_test_7 <- denormalize_data(predict(nn_model_7, test_set_2), data_min, data_max)
predicted_test_8 <- denormalize_data(predict(nn_model_8, test_set_2), data_min, data_max)
predicted_test_9 <- denormalize_data(predict(nn_model_9, test_set_3), data_min, data_max)
predicted_test_10 <- denormalize_data(predict(nn_model_10, test_set_3), data_min, data_max)
predicted_test_11 <- denormalize_data(predict(nn_model_11, test_set_4), data_min, data_max)
predicted_test_12 <- denormalize_data(predict(nn_model_12, test_set_4), data_min, data_max)
```

Figure 70: denormalized predicted and actual value

e) Discussion of the Four Statistical Indices.

1) RMSE (Root Mean Squared Error):

The square root of the average squared differences between the predicted and actual values is the value that this metric calculates. Better model performance is shown by a lower RMSE.

2) MAE (Mean Absolute Error):

The average absolute difference between the actual and predicted values is calculated by this metric. A lower MAE denotes higher model performance, similar to RMSE.

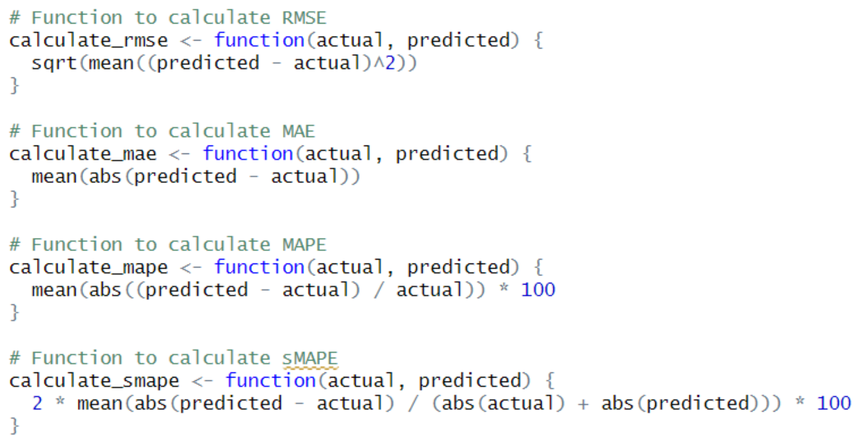
3) MAPE (Mean Absolute Percentage Error):

The average absolute percentage difference between the predicted and actual values is determined using this statistic. Lower numbers denote greater model performance, and it is frequently given as a percentage.

4) sMAPE (Symmetric Mean Absolute Percentage Error):

Certain issues with MAPE, such as its incapacity to handle values that are 0 or very close to zero, are addressed by this metric, which is a modification of MAPE. Better model performance is indicated by lower numbers, which are also represented as a percentage.

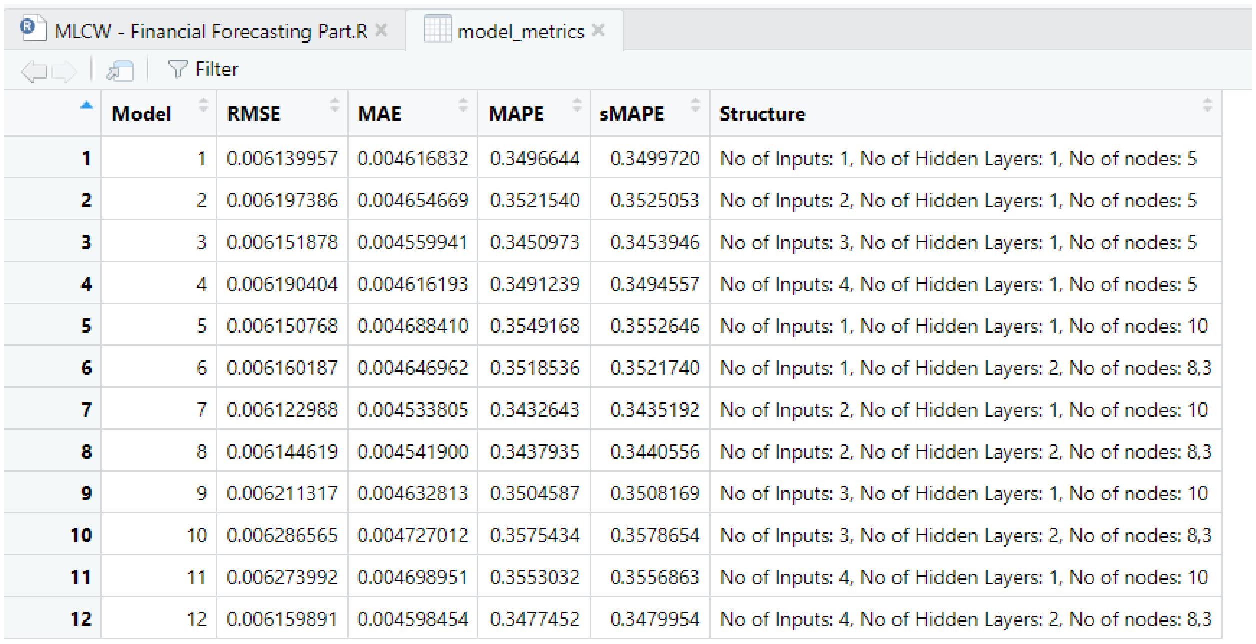
Below code block shows the functions for those,



```r
# Function to calculate RMSE
calculate_rmse <- function(actual, predicted) {
  sqrt(mean((predicted - actual)^2))
}

# Function to calculate MAE
calculate_mae <- function(actual, predicted) {
  mean(abs(predicted - actual))
}

# Function to calculate MAPE
calculate_mape <- function(actual, predicted) {
  mean(abs((predicted - actual) / actual)) * 100
}

# Function to calculate sMAPE
calculate_smape <- function(actual, predicted) {
  2 * mean(abs(predicted - actual) / (abs(actual) + abs(predicted))) * 100
}
```

Figure 71: Four Statistical Indices functions

f) Comparison table of MLP(AR) testing performances.

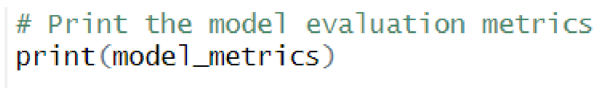


	Model	RMSE	MAE	MAPE	sMAPE	Structure
1	1	0.006139957	0.004616832	0.3496644	0.3499720	No of Inputs: 1, No of Hidden Layers: 1, No of nodes: 5
2	2	0.006197386	0.004654669	0.3521540	0.3525053	No of Inputs: 2, No of Hidden Layers: 1, No of nodes: 5
3	3	0.006151878	0.004559941	0.3450973	0.3453946	No of Inputs: 3, No of Hidden Layers: 1, No of nodes: 5
4	4	0.006190404	0.004616193	0.3491239	0.3494557	No of Inputs: 4, No of Hidden Layers: 1, No of nodes: 5
5	5	0.006150768	0.004688410	0.3549168	0.3552646	No of Inputs: 1, No of Hidden Layers: 1, No of nodes: 10
6	6	0.006160187	0.004646962	0.3518536	0.3521740	No of Inputs: 1, No of Hidden Layers: 2, No of nodes: 8,3
7	7	0.006122988	0.004533805	0.3432643	0.3435192	No of Inputs: 2, No of Hidden Layers: 1, No of nodes: 10
8	8	0.006144619	0.004541900	0.3437935	0.3440556	No of Inputs: 2, No of Hidden Layers: 2, No of nodes: 8,3
9	9	0.006211317	0.004632813	0.3504587	0.3508169	No of Inputs: 3, No of Hidden Layers: 1, No of nodes: 10
10	10	0.006286565	0.004727012	0.3575434	0.3578654	No of Inputs: 3, No of Hidden Layers: 2, No of nodes: 8,3
11	11	0.006273992	0.004698951	0.3553032	0.3556863	No of Inputs: 4, No of Hidden Layers: 1, No of nodes: 10
12	12	0.006159891	0.004598454	0.3477452	0.3479954	No of Inputs: 4, No of Hidden Layers: 2, No of nodes: 8,3

Figure 72: Comparison table

The best model is the one with the lowest value. Thus, in that order, Model 7, Model 3, and Model 12 are the top 3 models based on the table above. The finest model for one hidden layer model is the Model 7, while the best model for two hidden layer models is the Model 12.

g) Checking the Efficiency of Best Models.



```
# Print the model evaluation metrics
print(model_metrics)
```

Figure 73: Print Comparison table

The evaluation metrics (RMSE, MAE, MAPE, and sMAPE) for each of the 12 MLP models are included in the model_metrics data frame, along with an overview of the model structure.

Examine the amount of weight parameters in each of the most one- and two-hidden-layer models to determine their effectiveness. In general, it is advantageous to have models with fewer weight parameters because they may generalize better to unpredictable data and are less likely to overfit the training set.

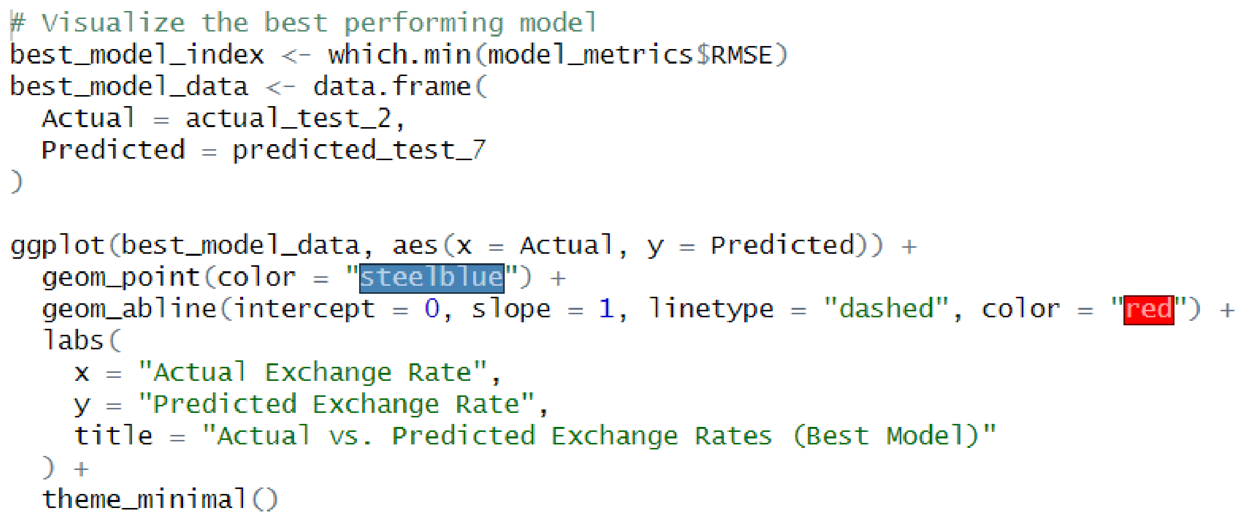
An MLP model's weight parameters are determined by the total number of input variables, hidden layers, and nodes in each hidden layer. Use the following formula to determine the number of weight parameters for a particular model structure:

Number of weight parameters = (Number of input variables * Number of nodes in the first hidden layer) + (Number of nodes in the first hidden layer * Number of nodes in the second hidden layer) + ... + (Number of nodes in the last hidden layer * Number of output variables) + (Number of bias parameters)

h) Visualizing the Best Model's Results.

The first step in this section is to determine the best-performing model's index using the lowest RMSE value. After that, you make a new data frame called *best_model_data* and fill it with the best model's actual and anticipated exchange rates.

Lastly, a scatter plot comparing the actual and expected exchange rates for the best model is produced using *ggplot2*. Plotting the actual data points is done with the *geom_point* function, and the ideal prediction line is added by *geom_abline* as a dashed red line with a slope of 1. A title and the corresponding axis titles are used to label the plot.



```
# Visualize the best performing model
best_model_index <- which.min(model_metrics$RMSE)
best_model_data <- data.frame(
  Actual = actual_test_2,
  Predicted = predicted_test_7
)

ggplot(best_model_data, aes(x = Actual, y = Predicted)) +
  geom_point(color = "steelblue") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
  labs(
    x = "Actual Exchange Rate",
    y = "Predicted Exchange Rate",
    title = "Actual vs. Predicted Exchange Rates (Best Model)"
  ) +
  theme_minimal()
```

Figure 74: Visualizing the Best Model Code Blok

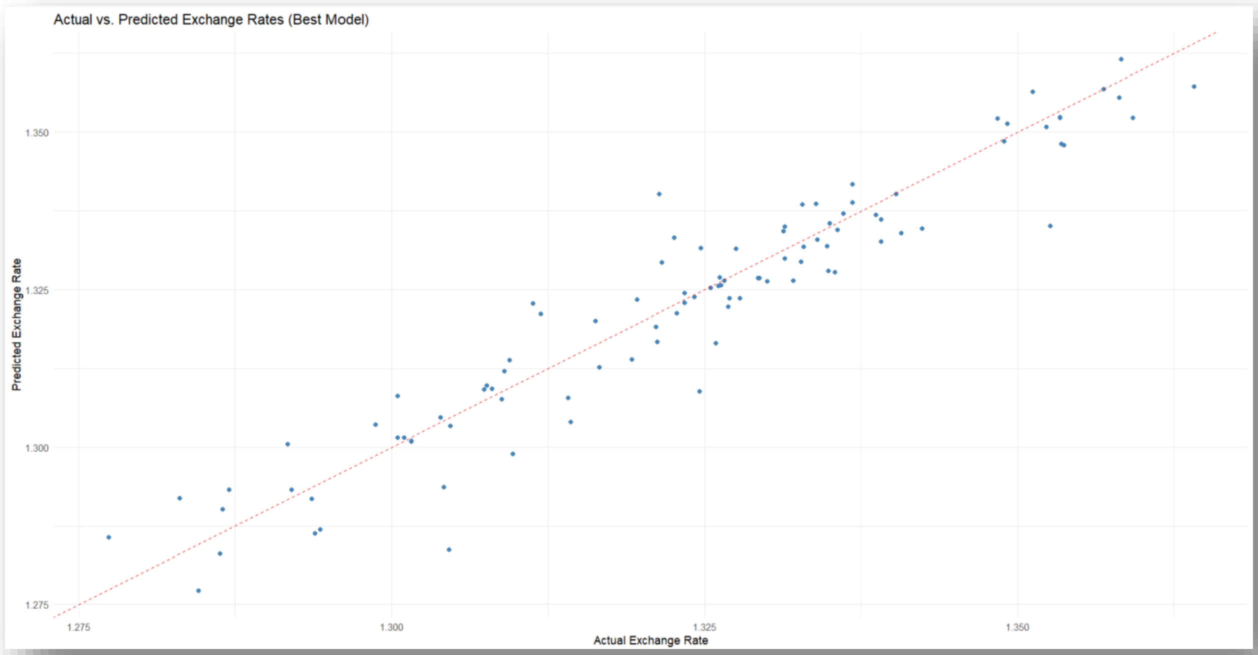


Figure 75: Visualizing the Best Model

REFERENCES.

Basnayake, B. R. P. M. and Chandrasekara, N. V. , Utilization of Various Optimization Algorithms in Neural Network Models for Forecasting Exchange Rates, 2024 4th International Conference on Advanced Research in Computing (ICARC), Belihuloya, Sri Lanka, 2024, pp. 55-60, Available from https://doi.org/10.1109/ICARC61713.2024.10499767 [Accessed 30 April 2024].

R. Ghazali, A. J. Hussain and M. N. M. Salleh, "Application of Polynomial Neural Networks to Exchange Rate Forecasting," 2008 Eighth International Conference on Intelligent Systems Design and Applications, Kaohsuing, Taiwan, 2008, pp. 90-95, Available from https://doi.org/10.1109/ISDA.2008.244 [Accessed 30 April 2024].

APPENDIX

Partitioning Clustering Part R Code.

```
#Load the libraries

library(readxl)

library(NbClust)

library(factoextra)

library(cluster)

library(fpc)


#Reading the WhiteWineFile

WhitewineData <- read_xlsx("Whitewine_v6.xlsx")


#Part a

####################################### Pre-proccesing
#################################################


# Subset the dataset to include only the first 11 attributes

First11Attributes_data <- WhitewineData[, 1:11]


# Check for missing data

sum(is.na(First11Attributes_data))


# show the box plot with outliers

boxplot(First11Attributes_data)


# Outlier Removal

outlierRemoval = apply(First11Attributes_data, 2, function(x) {
  median_value = median(x)
  mad = median(abs(x - median_value))
  return(x < (median_value - 3 * mad) | x > (median_value + 3 * mad))
})

First11Attributes_data_without_outliers = First11Attributes_data[!apply(outlierRemoval, 1,
any),]

# show the box plot without outliers
boxplot(First11Attributes_data_without_outliers)

# Normalization
First11Attributes_data_scaled = scale(First11Attributes_data_without_outliers)
#show the box plot
boxplot(First11Attributes_data_scaled )



#Part b
# Compute the optimal number of clusters using NbClust Method
NbClust(First11Attributes_data_scaled, distance = "euclidean", min.nc = 2, max.nc = 10, method
= "kmeans", index="all")

# Compute the gap statistic for each value of
set.seed(123)
gap_stat <- clusGap(First11Attributes_data_scaled, FUN = kmeans, nstart = 25, K.max = 10, B =
50)

fviz_gap_stat(gap_stat)
```

```
# Compute the optimal number of clusters using Elbow Method

fviz_nbclust(First11Attributes_data_scaled, kmeans, method = "wss") + geom_vline(xintercept =
2, linetype = 2) + labs(title = "Elbow Method")


# Compute the optimal number of clusters using Silhouette Method

fviz_nbclust(First11Attributes_data_scaled, kmeans, method = "silhouette")


#Part c

# Define the number of clusters to be formed

k = 2


# Perform k-means clustering on the scaled vehicle dataset using the specified number of clusters
and 10 random starts

kmeans_Data2 <- kmeans(First11Attributes_data_scaled, centers = k, nstart = 10)


# Print the k-means clustering results, including the cluster centers and assignments

kmeans_Data2


# Visualize the clustering results using the factoextra package, which creates a scatter plot with
the observations colored according to their cluster assignments

fviz_cluster(kmeans_Data2, First11Attributes_data_scaled, palette = c("#ff7f00","#836fff"),
geom = "point", ellipse.type = "convex", ggtheme = theme_bw())


# Extract the total within-cluster sum of squares (WSS), which is a measure of the compactness
of the clusters

wss <- kmeans_Data2$tot.withinss

wss


# Extract the between-cluster sum of squares (BSS), which is a measure of the separation
between the clusters
```

```
bss <- kmeans_Data2$betweenss
bss


# Compute the total sum of squares (TSS), which is the sum of the WSS and BSS
tss <- kmeans_Data2$totss
tss


# Compute the ratio of the BSS to the TSS, which is a measure of the proportion of the total
variance explained by the clustering
ratio <- (bss/tss)*100
ratio


#Part d
# Compute the silhouette widths for each observation,Applying silhouette for K=2
sil <- silhouette(kmeans_Data2$cluster, dist(First11Attributes_data_scaled))
fviz_silhouette(sil)




######################################## PCA
##################################################
#Part e
#View the scaled wine dataset
View(First11Attributes_data_scaled)


#Perform principal component analysis (PCA) on the scaled wine dataset using the prcomp
function Center and scale the dataset as well
pcaData <- prcomp(First11Attributes_data_scaled, center = TRUE, scale = TRUE)
summary(pcaData)
```

```
#Extract the eigenvalues and eigenvectors from the PCA results and view
eigenvalues <- pcaData$sdev^2
eigenvector <- pcaData$rotation
eigenvalues
eigenvector

#Calculate the cumulative score of the eigenvalues
cumulativeScore <- cumsum(eigenvalues/sum(eigenvalues))

#Select the eigenvalues whose cumulative score is less than or equal to 0.85
selectedValues <- which(cumulativeScore <= 0.85)
summary(selectedValues)

#Transform the original dataset using the selected eigenvectors
transformDataset <- as.data.frame(pcaData$x[,selectedValues])
summary(transformDataset)



#Part f
# Determine the optimal number of clusters using the NbClust method for PCA
nb_pca <- NbClust(transformDataset, distance = "euclidean", min.nc = 2, max.nc = 10, method =
"kmeans", index = "all")

# Determine the optimal number of clusters using the elbow method for PCA
fviz_nbclust(transformDataset, kmeans, method = "wss")+ geom_vline(xintercept = 2, linetype =
2) + labs(title = "Elbow Method")

# Compute the optimal number of clusters using Silhouette Method for PCA
```

```
fviz_nbclust(transformDataset, kmeans, method = "silhouette")

# Determine the optimal number of clusters using the gap statistic method for PCA
gap_stat <- clusGap(transformDataset, kmeans, K.max = 10, nstart = 10, B=50)
fviz_gap_stat(gap_stat)

#Part g
# Set the number of clusters based on the most favored k value
k <- 2

# Perform k-means analysis on the transformed PCA dataset
kmeans_pca_data <- kmeans(transformDataset, centers = k, nstart = 10)

# Show the related kmeans output
kmeans_pca_data

# Create a scatter plot of the clustering results
fviz_cluster(kmeans_pca_data, transformDataset, palette = c("#ff7f00","#836fff"), geom =
"point", ellipse.type = "convex", ggtheme = theme_bw())

# Extract the total within-cluster sum of squares (WSS), which is a measure of the compactness
of the clusters
wss<- kmeans_pca_data$tot.withinss
wss

# Extract the between-cluster sum of squares (BSS), which is a measure of the separation
between the clusters
bss <- kmeans_pca_data$betweenss
bss
```

```
# Compute the total sum of squares (TSS), which is the sum of the WSS and BSS

tss <- kmeans_pca_data$totss

tss


# Compute the ratio of the BSS to the TSS, which is a measure of the proportion of the total
variance explained by the clustering

ratio <- (bss/tss)*100

ratio


#Part h

# Compute the silhouette widths for each observation,Applying silhouette for K=2

sil_pca <- silhouette(kmeans_pca_data$cluster, dist(transformDataset))

fviz_silhouette(sil_pca)


#Part i

#calculate the Calinski-Harabaz index k=2

ch_index <- calinhara(transformDataset, kmeans_pca_data$cluster)

print(ch_index)
```

Financial Forecasting Part

```
# Load required libraries
library(neuralnet)
library(readxl)
library(dplyr)
library(Metrics)
library(ggplot2)

# Read the Excel file
currency_data <- read_excel("ExchangeUSD.xlsx")

# Extract the relevant column (USD/EUR exchange rate)
exchange_rates <- as.data.frame(currency_data[3])
exchange_rates
str(exchange_rates)
summary(exchange_rates)

# Get the minimum and maximum values of the original data
data_min <- min(exchange_rates)
data_max <- max(exchange_rates)

# Function to Normalize the data
normalize_data <- function(data) {
  return ((data - min(data)) / (max(data) - min(data)))
}

# Function to de-normalize the data
denormalize_data <- function(normalized_value, min_value, max_value) {
  return ((max_value - min_value) * normalized_value + min_value)
}



# Create lagged variables
lag_1 <- lag(exchange_rates, 1)
lag_2 <- lag(exchange_rates, 2)
lag_3 <- lag(exchange_rates, 3)
lag_4 <- lag(exchange_rates, 4)

# Combine the original and lagged variables into different datasets(I/O Matrix)
dataset_1 <- cbind(exchange_rates, lag_1)
dataset_2 <- cbind(dataset_1, lag_2)
dataset_3 <- cbind(dataset_2, lag_3)
dataset_4 <- cbind(dataset_3, lag_4)

# Remove rows with missing values
dataset_1 <- na.omit(dataset_1)
dataset_2 <- na.omit(dataset_2)
dataset_3 <- na.omit(dataset_3)
dataset_4 <- na.omit(dataset_4)

# Rename columns
colnames(dataset_1) <- c('target', 'input_1')
colnames(dataset_2) <- c('target', 'input_1', 'input_2')
colnames(dataset_3) <- c('target', 'input_1', 'input_2', 'input_3')
colnames(dataset_4) <- c('target', 'input_1', 'input_2', 'input_3', 'input_4')
```

```
#Normalizing the data
normalized_data_1 <- as.data.frame(lapply(dataset_1, normalize_data))
normalized_data_2 <- as.data.frame(lapply(dataset_2, normalize_data))
normalized_data_3 <- as.data.frame(lapply(dataset_3, normalize_data))
normalized_data_4 <- as.data.frame(lapply(dataset_4, normalize_data))

# Create a box plot of the normalized data
boxplot(normalized_data_1)
boxplot(normalized_data_2)
boxplot(normalized_data_3)
boxplot(normalized_data_4)

# Split data into training and testing sets
train_set_1 <- normalized_data_1[1:400, ]
test_set_1 <- normalized_data_1[401:nrow(normalized_data_1), ]

train_set_2 <- normalized_data_2[1:400, ]
test_set_2 <- normalized_data_2[401:nrow(normalized_data_2), ]

train_set_3 <- normalized_data_3[1:400, ]
test_set_3 <- normalized_data_3[401:nrow(normalized_data_3), ]

train_set_4 <- normalized_data_4[1:400, ]
test_set_4 <- normalized_data_4[401:nrow(normalized_data_4), ]

# Set seed for reproducibility
set.seed(123)
```

```
# Define neural network models

# 1 hidden layer with 5 nodes from t-1 to t-4

nn_model_1 <- neuralnet(target ~ input_1, data = train_set_1, hidden = c(5), linear.output =
TRUE)

nn_model_2 <- neuralnet(target ~ input_1 + input_2, data = train_set_2, hidden = c(5),
linear.output = TRUE)

nn_model_3 <- neuralnet(target ~ input_1 + input_2 + input_3, data = train_set_3, hidden = c(5),
linear.output = TRUE)

nn_model_4 <- neuralnet(target ~ input_1 + input_2 + input_3 + input_4, data = train_set_4,
hidden = c(5), linear.output = TRUE)


# 1 hidden layer with 10 nodes and 2 hidden layers with 5 and 3 nodes from t-1 to t-4

nn_model_5 <- neuralnet(target ~ input_1, data = train_set_1, hidden = c(10), linear.output =
TRUE)

nn_model_6 <- neuralnet(target ~ input_1, data = train_set_1, hidden = c(8, 3), linear.output =
TRUE)

nn_model_7 <- neuralnet(target ~ input_1 + input_2, data = train_set_2, hidden = c(10),
linear.output = TRUE)

nn_model_8 <- neuralnet(target ~ input_1 + input_2, data = train_set_2, hidden = c(8, 3),
linear.output = TRUE)

nn_model_9 <- neuralnet(target ~ input_1 + input_2 + input_3, data = train_set_3, hidden =
c(10), linear.output = TRUE)

nn_model_10 <- neuralnet(target ~ input_1 + input_2 + input_3, data = train_set_3, hidden =
c(8, 3), linear.output = TRUE)

nn_model_11 <- neuralnet(target ~ input_1 + input_2 + input_3 + input_4, data = train_set_4,
hidden = c(10), linear.output = TRUE)

nn_model_12 <- neuralnet(target ~ input_1 + input_2 + input_3 + input_4, data = train_set_4,
hidden = c(8, 3), linear.output = TRUE)


#plot neural network models

plot(nn_model_1)
plot(nn_model_2)
plot(nn_model_3)
plot(nn_model_4)
plot(nn_model_5)
plot(nn_model_6)
plot(nn_model_7)
plot(nn_model_8)
plot(nn_model_9)
plot(nn_model_10)
plot(nn_model_11)
plot(nn_model_12)

#De-normalize the Actual values for testing sets
actual_test_1 <- denormalize_data(test_set_1$target, data_min, data_max)
actual_test_2 <- denormalize_data(test_set_2$target, data_min, data_max)
actual_test_3 <- denormalize_data(test_set_3$target, data_min, data_max)
actual_test_4 <- denormalize_data(test_set_4$target, data_min, data_max)

#De-normalize the Predicted values for testing sets
predicted_test_1 <- denormalize_data(predict(nn_model_1, test_set_1), data_min, data_max)
predicted_test_2 <- denormalize_data(predict(nn_model_2, test_set_2), data_min, data_max)
predicted_test_3 <- denormalize_data(predict(nn_model_3, test_set_3), data_min, data_max)
predicted_test_4 <- denormalize_data(predict(nn_model_4, test_set_4), data_min, data_max)
predicted_test_5 <- denormalize_data(predict(nn_model_5, test_set_1), data_min, data_max)
predicted_test_6 <- denormalize_data(predict(nn_model_6, test_set_1), data_min, data_max)
predicted_test_7 <- denormalize_data(predict(nn_model_7, test_set_2), data_min, data_max)
predicted_test_8 <- denormalize_data(predict(nn_model_8, test_set_2), data_min, data_max)
predicted_test_9 <- denormalize_data(predict(nn_model_9, test_set_3), data_min, data_max)
```

```
predicted_test_10 <- denormalize_data(predict(nn_model_10, test_set_3), data_min, data_max)
predicted_test_11 <- denormalize_data(predict(nn_model_11, test_set_4), data_min, data_max)
predicted_test_12 <- denormalize_data(predict(nn_model_12, test_set_4), data_min, data_max)

# Function to calculate RMSE
calculate_rmse <- function(actual, predicted) {
  sqrt(mean((predicted - actual)^2))
}

# Function to calculate MAE
calculate_mae <- function(actual, predicted) {
  mean(abs(predicted - actual))
}

# Function to calculate MAPE
calculate_mape <- function(actual, predicted) {
  mean(abs((predicted - actual) / actual)) * 100
}

# Function to calculate sMAPE
calculate_smape <- function(actual, predicted) {
  2 * mean(abs(predicted - actual) / (abs(actual) + abs(predicted))) * 100
}

# Calculate evaluation metrics for each model
model_metrics <- data.frame(
  Model = 1:12,
  RMSE = c(
  calculate_rmse(actual_test_1, predicted_test_1),
  calculate_rmse(actual_test_2, predicted_test_2),
  calculate_rmse(actual_test_3, predicted_test_3),
  calculate_rmse(actual_test_4, predicted_test_4),
  calculate_rmse(actual_test_1, predicted_test_5),
  calculate_rmse(actual_test_1, predicted_test_6),
  calculate_rmse(actual_test_2, predicted_test_7),
  calculate_rmse(actual_test_2, predicted_test_8),
  calculate_rmse(actual_test_3, predicted_test_9),
  calculate_rmse(actual_test_3, predicted_test_10),
  calculate_rmse(actual_test_4, predicted_test_11),
  calculate_rmse(actual_test_4, predicted_test_12)
),
MAE = c(
  calculate_mae(actual_test_1, predicted_test_1),
  calculate_mae(actual_test_2, predicted_test_2),
  calculate_mae(actual_test_3, predicted_test_3),
  calculate_mae(actual_test_4, predicted_test_4),
  calculate_mae(actual_test_1, predicted_test_5),
  calculate_mae(actual_test_1, predicted_test_6),
  calculate_mae(actual_test_2, predicted_test_7),
  calculate_mae(actual_test_2, predicted_test_8),
  calculate_mae(actual_test_3, predicted_test_9),
  calculate_mae(actual_test_3, predicted_test_10),
  calculate_mae(actual_test_4, predicted_test_11),
  calculate_mae(actual_test_4, predicted_test_12)
),
MAPE = c(
  calculate_mape(actual_test_1, predicted_test_1),
  calculate_mape(actual_test_2, predicted_test_2),
  calculate_mape(actual_test_3, predicted_test_3),
  calculate_mape(actual_test_4, predicted_test_4),
  calculate_mape(actual_test_1, predicted_test_5),
  calculate_mape(actual_test_1, predicted_test_6),
  calculate_mape(actual_test_2, predicted_test_7),
  calculate_mape(actual_test_2, predicted_test_8),
  calculate_mape(actual_test_3, predicted_test_9),
  calculate_mape(actual_test_3, predicted_test_10),
  calculate_mape(actual_test_4, predicted_test_11),
  calculate_mape(actual_test_4, predicted_test_12)
),
sMAPE = c(
  calculate_smape(actual_test_1, predicted_test_1),
  calculate_smape(actual_test_2, predicted_test_2),
  calculate_smape(actual_test_3, predicted_test_3),
  calculate_smape(actual_test_4, predicted_test_4),
  calculate_smape(actual_test_1, predicted_test_5),
  calculate_smape(actual_test_1, predicted_test_6),
  calculate_smape(actual_test_2, predicted_test_7),
  calculate_smape(actual_test_2, predicted_test_8),
  calculate_smape(actual_test_3, predicted_test_9),
  calculate_smape(actual_test_3, predicted_test_10),
  calculate_smape(actual_test_4, predicted_test_11),
  calculate_smape(actual_test_4, predicted_test_12)
),
Structure = c(
    "No of Inputs: 1, No of Hidden Layers: 1, No of nodes: 5",
    "No of Inputs: 2, No of Hidden Layers: 1, No of nodes: 5",
    "No of Inputs: 3, No of Hidden Layers: 1, No of nodes: 5",
    "No of Inputs: 4, No of Hidden Layers: 1, No of nodes: 5",
    "No of Inputs: 1, No of Hidden Layers: 1, No of nodes: 10",
    "No of Inputs: 1, No of Hidden Layers: 2, No of nodes: 8,3",
    "No of Inputs: 2, No of Hidden Layers: 1, No of nodes: 10",
    "No of Inputs: 2, No of Hidden Layers: 2, No of nodes: 8,3",
    "No of Inputs: 3, No of Hidden Layers: 1, No of nodes: 10",
    "No of Inputs: 3, No of Hidden Layers: 2, No of nodes: 8,3",
    "No of Inputs: 4, No of Hidden Layers: 1, No of nodes: 10",
    "No of Inputs: 4, No of Hidden Layers: 2, No of nodes: 8,3"
  )
)

# Print the model evaluation metrics
print(model_metrics)

# Visualize the best performing model
best_model_index <- which.min(model_metrics$RMSE)
best_model_data <- data.frame(
  Actual = actual_test_2,
  Predicted = predicted_test_7
)

ggplot(best_model_data, aes(x = Actual, y = Predicted)) +
  geom_point(color = "steelblue") +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "red") +
labs(
  x = "Actual Exchange Rate",
  y = "Predicted Exchange Rate",
  title = "Actual vs. Predicted Exchange Rates (Best Model)"
) +
theme_minimal()
```