INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology (IIT) Sri Lanka, Affiliated with The University of Westminster, UK Module

5DATA001C.2

Machine Learning and Data Mining

Name: Sanuda Damboragama

UOW no: W1998852

IIT no: 20220533

Table of Contents

Partitioning Clustering Part

1.1 Subtask Objectives

```r
# Load the required libraries
library(dplyr)        # For data manipulation
library(ggplot2)      # For data visualization
library(cluster)      # For clustering algorithms
library(readxl)       # For reading Excel files
library(NbClust)      # For cluster number determination
library(clusterCrit)  # For cluster validation
library(fpc)          # For gap statistic calculation
library(factoextra)   # For visualizing clustering results and more
library(reshape2)     # For reshaping data frames
```

The **dplyr** library is used for data manipulation. It provides a set of functions (verbs) to handle common data manipulation tasks efficiently, such as filtering, selecting, mutating, summarizing, and arranging data.

The **ggplot2** library is employed for data visualization. It implements the grammar of graphics, enabling users to create detailed and customizable plots. This package allows for creating complex and multi-layered graphics.

The **cluster** library provides various clustering algorithms. These algorithms group similar observations into clusters, and the package includes methods for hierarchical and partitioning clustering, such as agnes, diana, pam, and clara.

The **readxl** library is used for reading Excel files into R. It supports both **.xls** and **.xlsx** formats, with the primary function being **read_excel()**, which imports Excel sheets as data frames.
The **NbClust** library provides tools for determining the optimal number of clusters in a dataset. It computes and compares 30 different indices to suggest the best number of clusters, with the key function being **NbClust()**.

The **clusterCrit** library is used for validating clustering results. It offers internal and external clustering validation indices, allowing users to evaluate the quality of their clustering solutions.
The **fpc** library provides various tools for clustering and cluster validation, including the calculation of the gap statistic, which helps determine the optimal number of clusters. It includes functions for computing cluster statistics and visualizing clustering results.

The **factoextra** library is used for visualizing clustering results and performing exploratory factor analysis. It enhances the interpretability of clustering outputs by providing visualization tools for clustering solutions.

The **reshape2** library is used for reshaping data frames. It makes it easy to transform data from wide to long format and vice versa, with the main functions being **melt()** and **dcast()**, which convert data frames between these formats.

1.1.a

```r
#a
# Load the dataset
wine_data <- read_excel("Whitewine_v6.xlsx")

# Scaling the data
scaled_data <- scale(wine_data[,1:11])

# Convert scaled data to a data frame
scaled_data_df <- as.data.frame(scaled_data)

# Outlier detection and removal
# Using boxplot and filtering outliers
outliers_removed <- scaled_data_df %>%
  filter_all(all_vars(between(., quantile(., 0.25) - 1.5 * IQR(.), quantile(., 0.75) + 1.5 * IQR(.

# Plot the data before and after outlier removal
# Melt the data for easier plotting with ggplot2
melted_scaled_data <- melt(scaled_data_df)
melted_outliers_removed <- melt(outliers_removed)

# Create boxplots before outlier removal
ggplot(melted_scaled_data, aes(x = variable, y = value)) +
  geom_boxplot() +
  ggtitle("Boxplot of Scaled Data Before Outlier Removal") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Create boxplots after outlier removal
ggplot(melted_outliers_removed, aes(x = variable, y = value)) +
  geom_boxplot() +
  ggtitle("Boxplot of Scaled Data After Outlier Removal") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
# Justification for outlier removal:
# Outliers can significantly affect clustering results by skewing centroids and increasing intra-c
# Removing outliers helps improve clustering quality by reducing noise and ensuring more represent
```

Loading the Dataset:

Loads data from an Excel file named "Whitewine_v6.xlsx" into a data frame called wine_data.

Scaling the Data:

Scales the first 11 columns of wine_data to have a mean of 0 and a standard deviation of 1, storing the result in scaled_data.

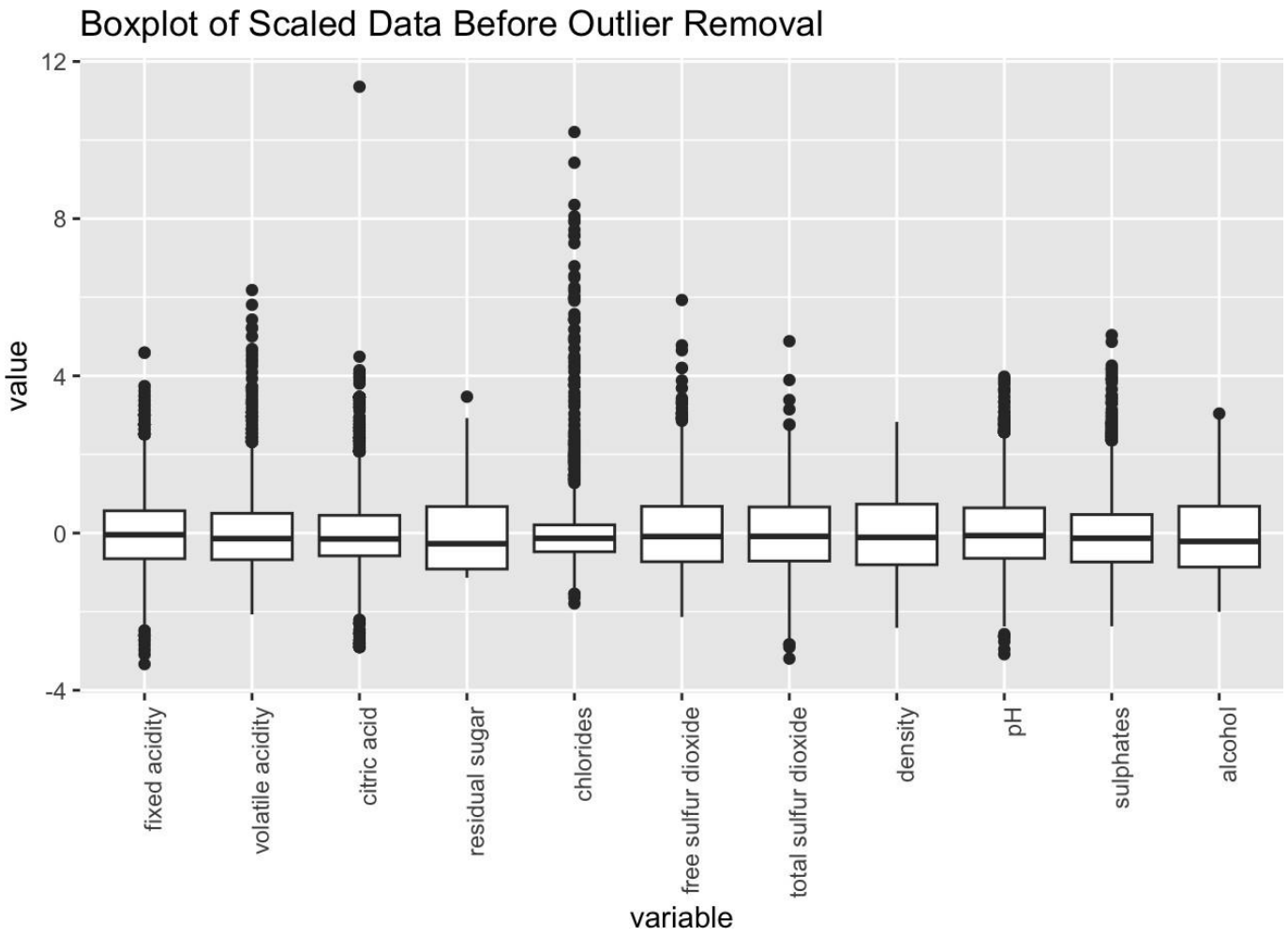
Converting to Data Frame:
Converts the scaled data matrix into a data frame named scaled_data_df.

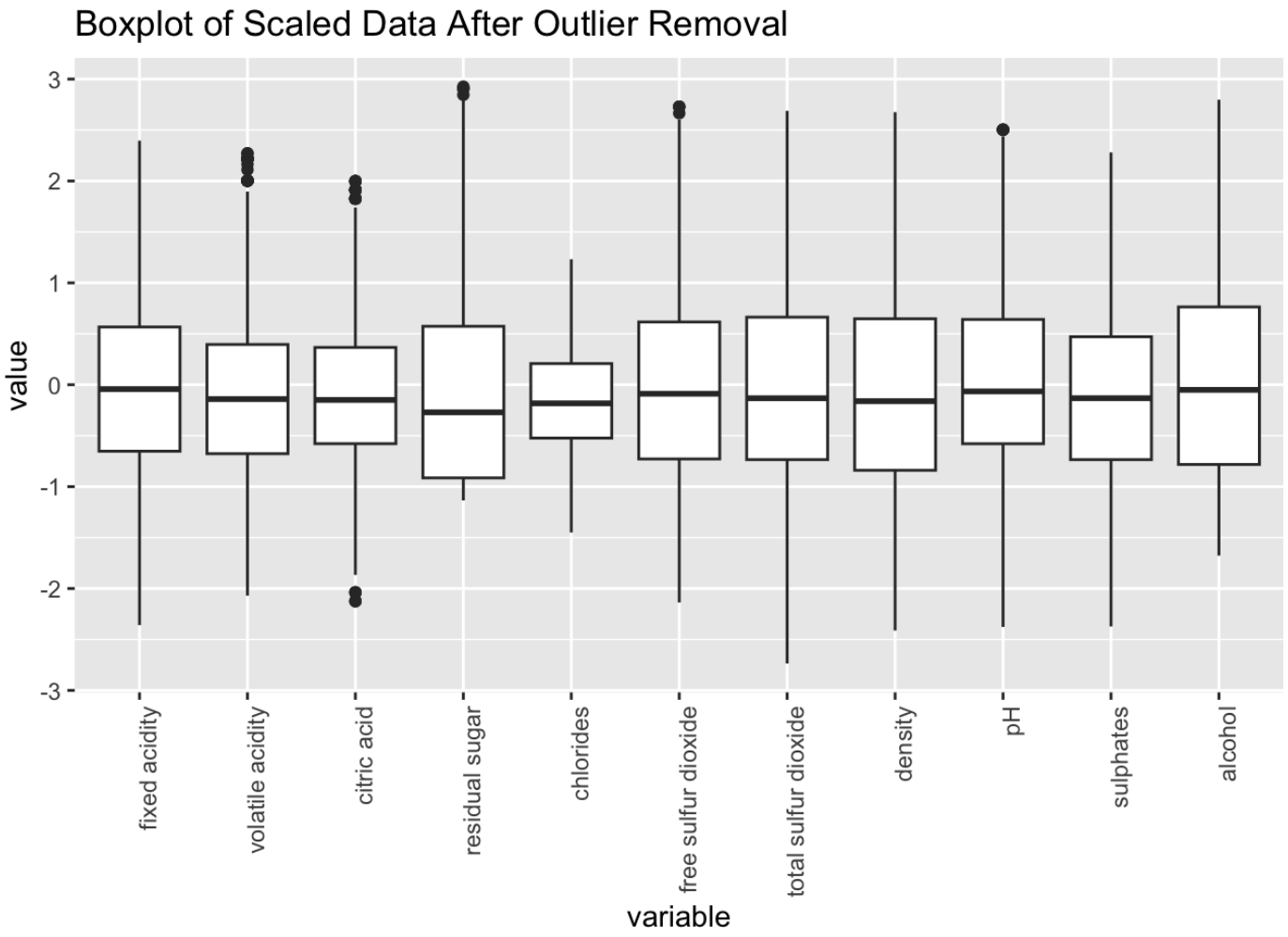
Outlier Detection and Removal:
Removes outliers from scaled_data_df by keeping values within the range of the first quartile minus 1.5 times the interquartile range (IQR) to the third quartile plus 1.5 times the IQR. The cleaned data is stored in outliers_removed.

Data Preparation for Plotting:
Reshapes scaled_data_df and outliers_removed from wide format to long format for easier plotting with ggplot2, storing the results in melted_scaled_data and melted_outliers_removed, respectively.

Plotting Data:
Creates a boxplot of the scaled data before outlier removal using ggplot2.
Creates a boxplot of the scaled data after outlier removal using ggplot2.

Justification for Outlier Removal:
Outliers can skew clustering results and increase intra-cluster variability.
Removing outliers helps improve clustering quality by reducing noise and ensuring more representative clusters.


Boxplot of Scaled Data Before Outlier Removal
value
12
8
4
0
-4
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
variable

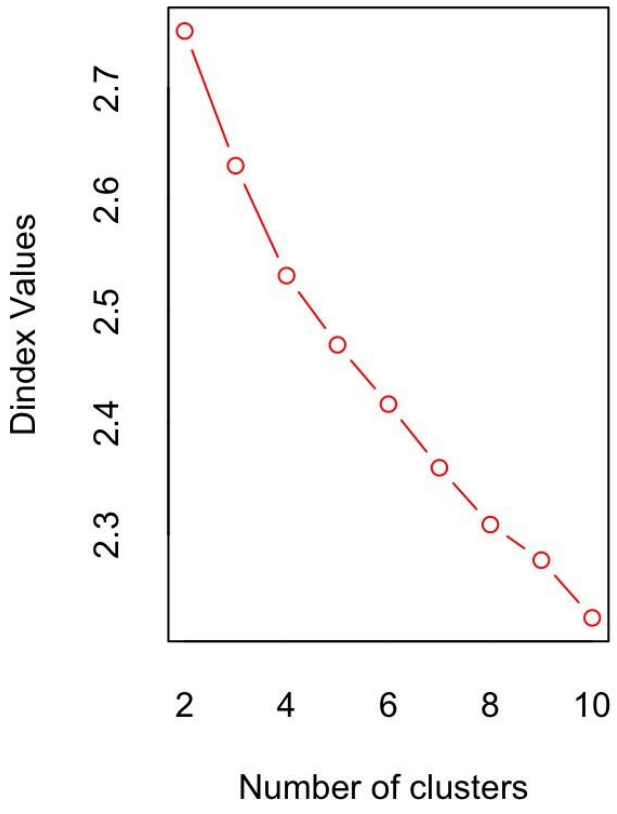

Boxplot of Scaled Data After Outlier Removal
value
3
2
1
0
-1
-2
-3
fixed acidity
volatile acidity
citric acid
residual sugar
chlorides
free sulfur dioxide
total sulfur dioxide
density
pH
sulphates
alcohol
variable

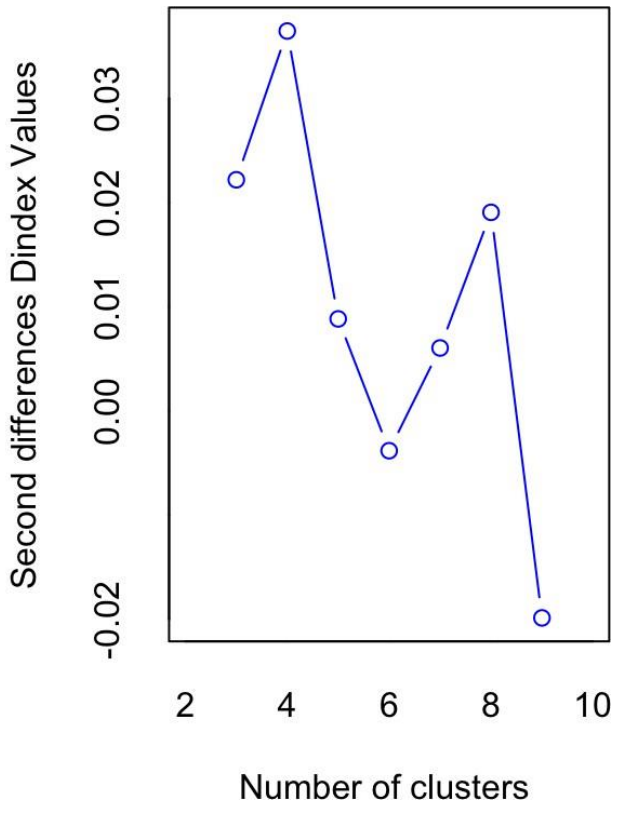
1.1 b

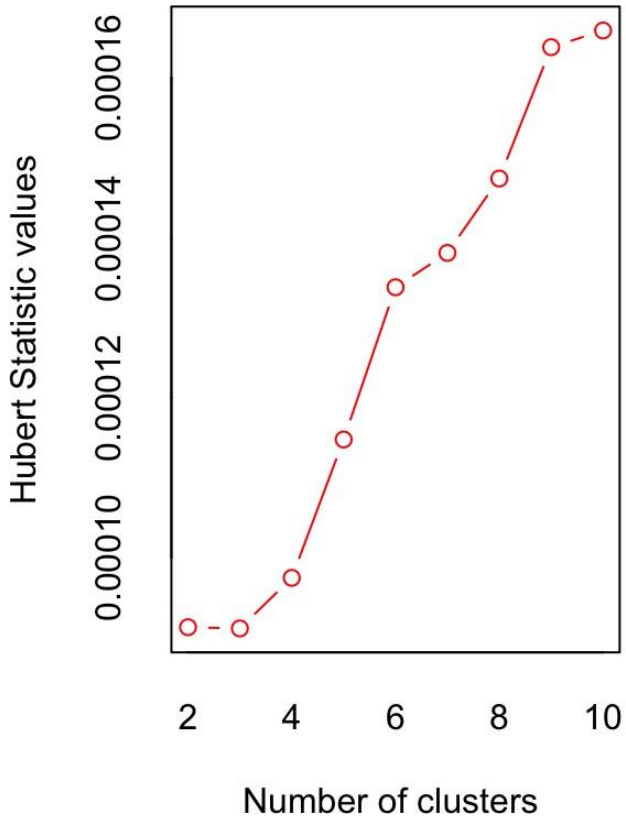
```
#b
# Determine the number of clusters using the NbClust method
set.seed(123)
clusterNo=NbClust(scaled_data,distance="euclidean",min.nc=2,max.nc=10,method="kmeans",index="all")

# Determine the number of clusters using the Elbow method
fviz_nbclust(scaled_data, kmeans, method = "wss")

# Determine the number of clusters using the Gap statistics method
fviz_nbclust(scaled_data, kmeans, method = 'gap_stat')

# Determine the number of clusters using the silhouette method
fviz_nbclust(scaled_data, kmeans, method = 'silhouette')
```

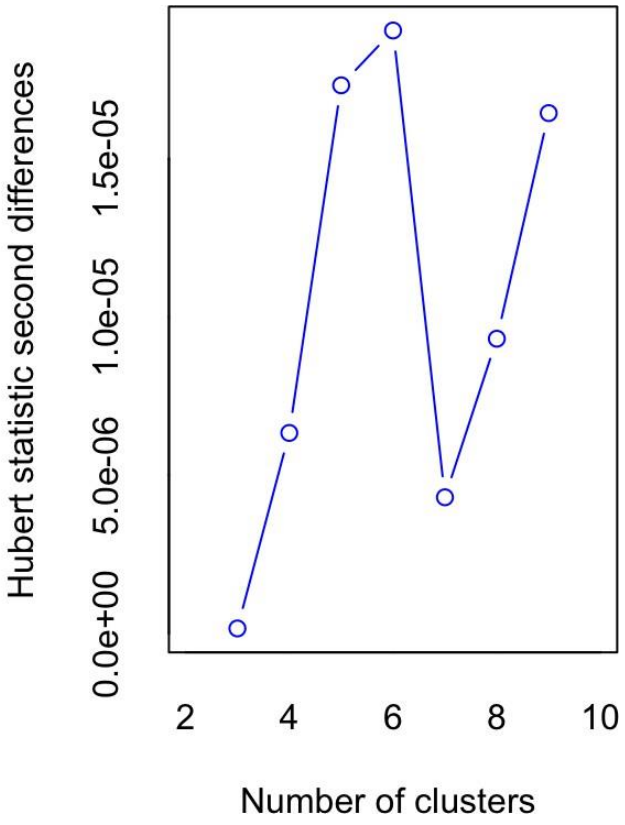
NbClust for Determining Number of Clusters (nb_clusters):

The NbClust function from the NbClust package is used to assess the optimal number of clusters (k) based on various indices. In this case, the function is applied to outliers_removeddata using Euclidean distance. min.nc and max.nc specify the range of cluster numbers to consider. This method helps in selecting the most appropriate k by evaluating different clustering criteria







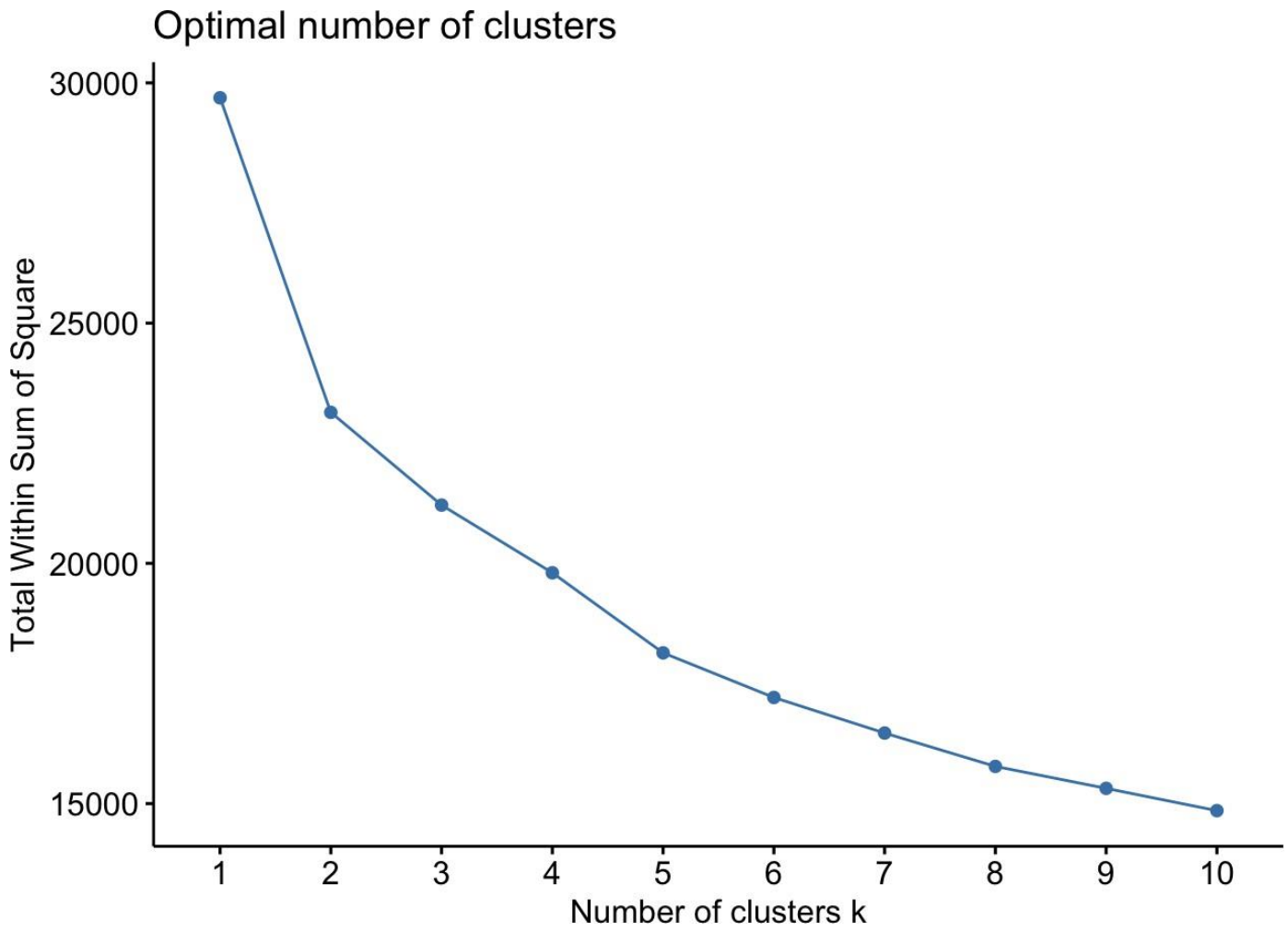



```
                ***** Conclusion *****

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

According to the above no.of clusters is 2

Elbow Method to Determine Optimal k:

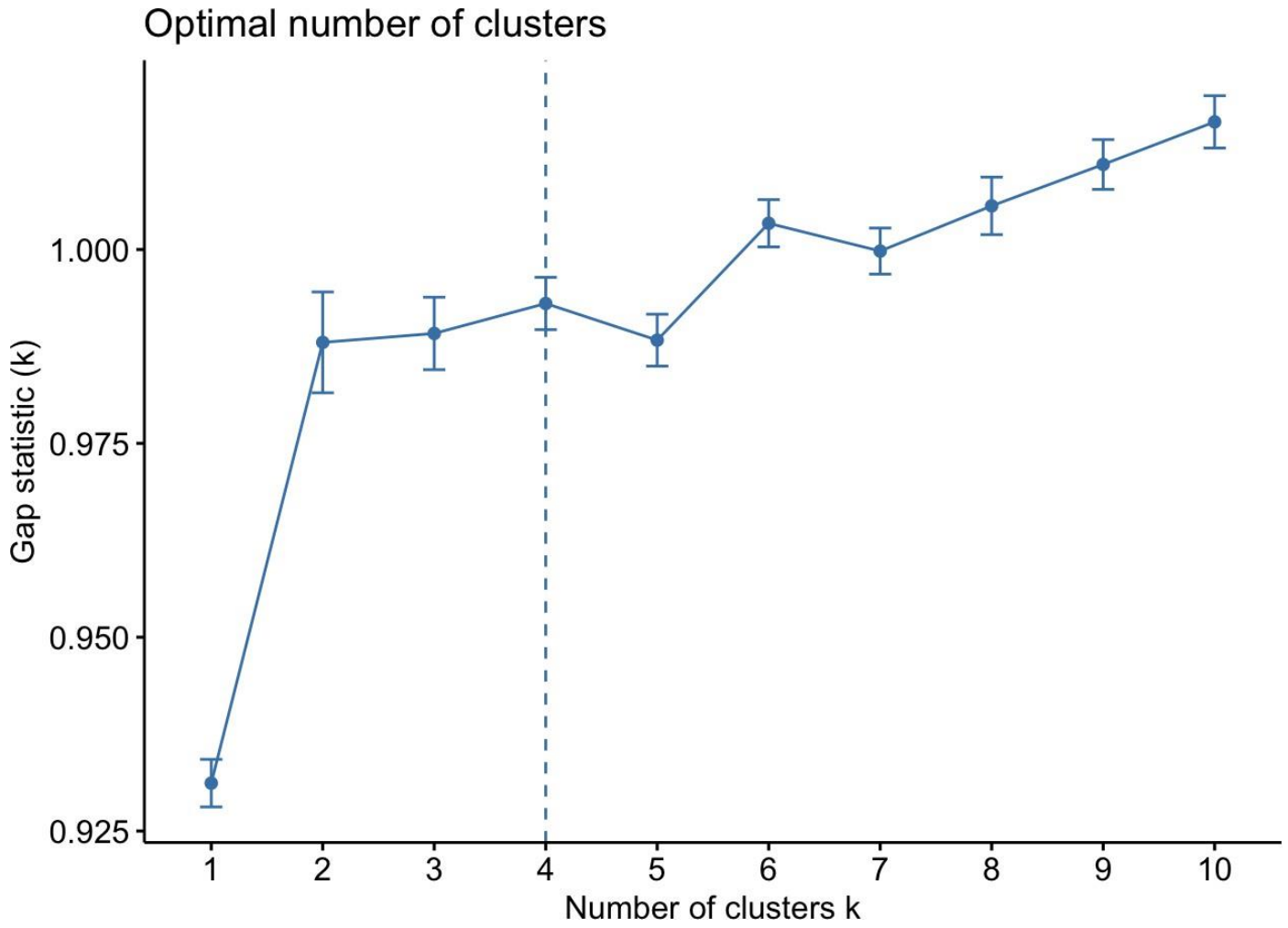
The elbow method is employed to identify the optimal number of clusters (k) based on the within-cluster sum of squares (WSS). The code iterates over different values of k (from 1 to10) and computes the total within-cluster sum of squares (tot.withinss) for each k using k- means clustering (kmeans). A plot (elbow_plot) of k versus WSS is generated to visually identify the "elbow" point where the rate of decrease in WSS slows down, suggesting the optimal k.



According to the above no.of clusters is 2

Gap Statistic:

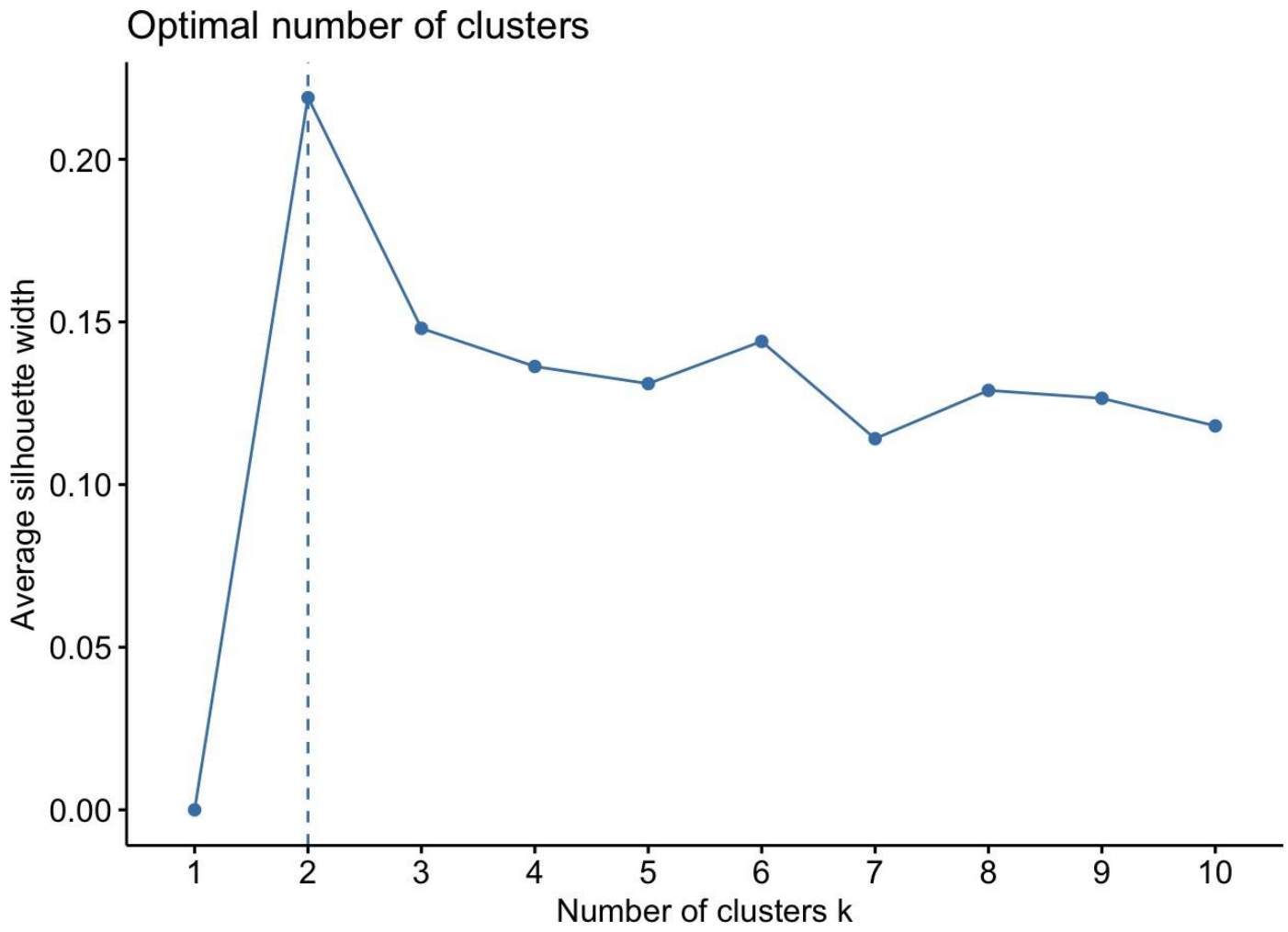
The clusGap function calculates the gap statistic to assess the optimal number of clusters (k)based on k-means clustering. It compares the within-cluster dispersion of the observed data with that of randomly generated reference datasets. The goal is to find k where the gap statistic is maximized, indicating a significant improvement in clustering structure comparedto random data.



According to the above no.of clusters is 4

Silhouette Method for Clustering Quality:

The silhouette method evaluates the quality of clustering by assessing how well each object fits into its assigned cluster. The code computes silhouette scores (silhouette_scores) for different values of k (ranging from 2 to 10) using k-means clustering. A higher average silhouette width (avg.width) indicates better-defined and distinct clusters. This method helpsin determining the k that maximizes clustering cohesion and separation.



According to the above no.of clusters is 2

So most of Automated tools show the no of clusters are 2.Therefor k=2.

1.1 c

```r
#c
# Apply k-means clustering with k=2
k <- 2
set.seed(123)
kmeans_output <- kmeans(scaled_data, centers = k)
kmeans_output

# Visualize clustering results
fviz_cluster(kmeans_output, data = scaled_data)

# Calculate BSS and WSS
BSS <- sum(kmeans_output$betweenss)
WSS <- kmeans_output$tot.withinss
TSS <- BSS + WSS
BSS_ratio <- BSS / TSS

# Output kmeans results
print(kmeans_output)
print(BSS_ratio)
```

Set Number of Clusters and Seed:

k <- 2: Sets the number of clusters *k* to 2.

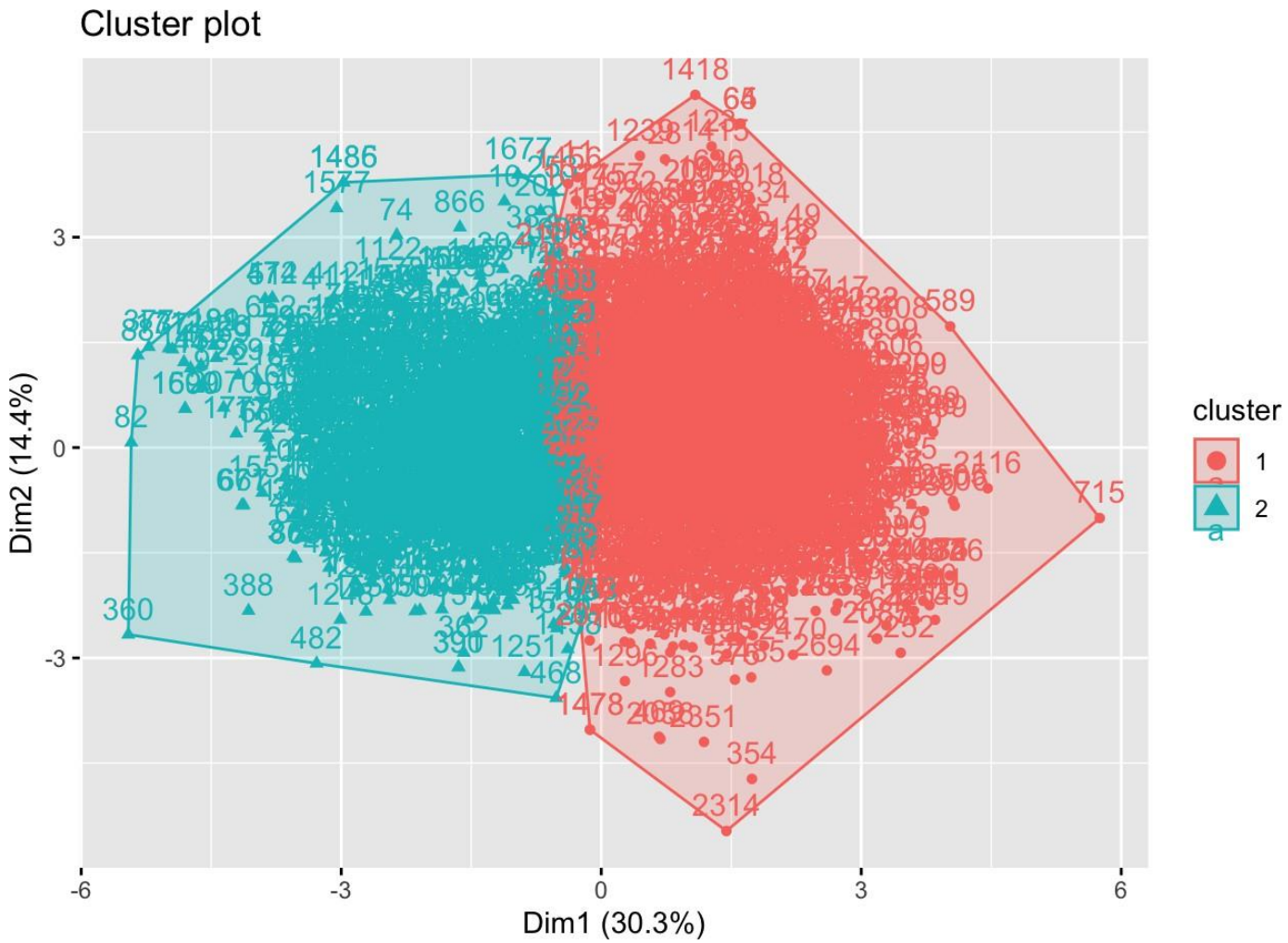
set.seed(123): Sets the seed for random number generation to ensure reproducibility of the k-means results.

Apply k-Means Clustering:

kmeans_output <- kmeans(scaled_data, centers = k): Applies k-means clustering to **scaled_data** with 2 clusters, storing the result in **kmeans_output**.

Visualize Clustering Results:

viz_cluster(kmeans_output, data = scaled_data): Uses the **fviz_cluster** function from the **factoextra** package to visualize the clustering results. This function plots the data points colored by their assigned clusters and displays cluster centers.



Calculate BSS and WSS:

BSS <- sum(kmeans_output$betweenss): Calculates the Between Sum of Squares (BSS), which measures the variation between different clusters.

WSS <- kmeans_output$tot.withinss: Retrieves the Total Within Sum of Squares (WSS), which measures the variation within each cluster.

TSS <- BSS + WSS: Calculates the Total Sum of Squares (TSS) by summing BSS and WSS. TSS represents the total variation in the data.

BSS_ratio <- BSS / TSS: Calculates the ratio of BSS to TSS, indicating the proportion of the total variance that is explained by the clustering.

Output k-Means Results:

print(kmeans_output): Prints the complete k-means clustering output, including cluster centers, cluster assignment of each data point, and clustering statistics.

print(BSS_ratio): Prints the BSS ratio, showing the effectiveness of the clustering in terms of explained variance.

```
Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
> print(BSS_ratio)
[1] 0.220511
```

According to the above BSS ratio is 0.220511

1.1.d

```
#d
# Silhouette plot for visualizing clustering quality
silhouette_plot <- silhouette(kmeans_output$cluster, dist(scaled_data))  # Use scaled_data instead
plot(silhouette_plot)

# Calculate average silhouette width score
avg_silhouette <- mean(silhouette_plot[, 3])
print(avg_silhouette)
# Discussion on silhouette plot
# The silhouette plot illustrates individual point fitting within clusters and cluster distinctnes
```

Silhouette Plot for Visualizing Clustering Quality:

silhouette_plot <- silhouette(kmeans_output$cluster, dist(scaled_data)):
Calculates the silhouette information for each data point based on the k-means cluster assignments.
kmeans_output$cluster provides the cluster assignments for each point.
dist(scaled_data) computes the distance matrix for the scaled data.

plot(silhouette_plot):
Plots the silhouette values, visualizing how well each data point fits within its cluster and how distinct each cluster is from the others

.

Calculate Average Silhouette Width Score:

avg_silhouette <- mean(silhouette_plot[, 3]):
Computes the average silhouette width score by taking the mean of the silhouette values for all data points (the third column of the silhouette plot object contains these values).

print(avg_silhouette):
Prints the average silhouette width score, which is a measure of clustering quality.

Discussion on Silhouette Plot:

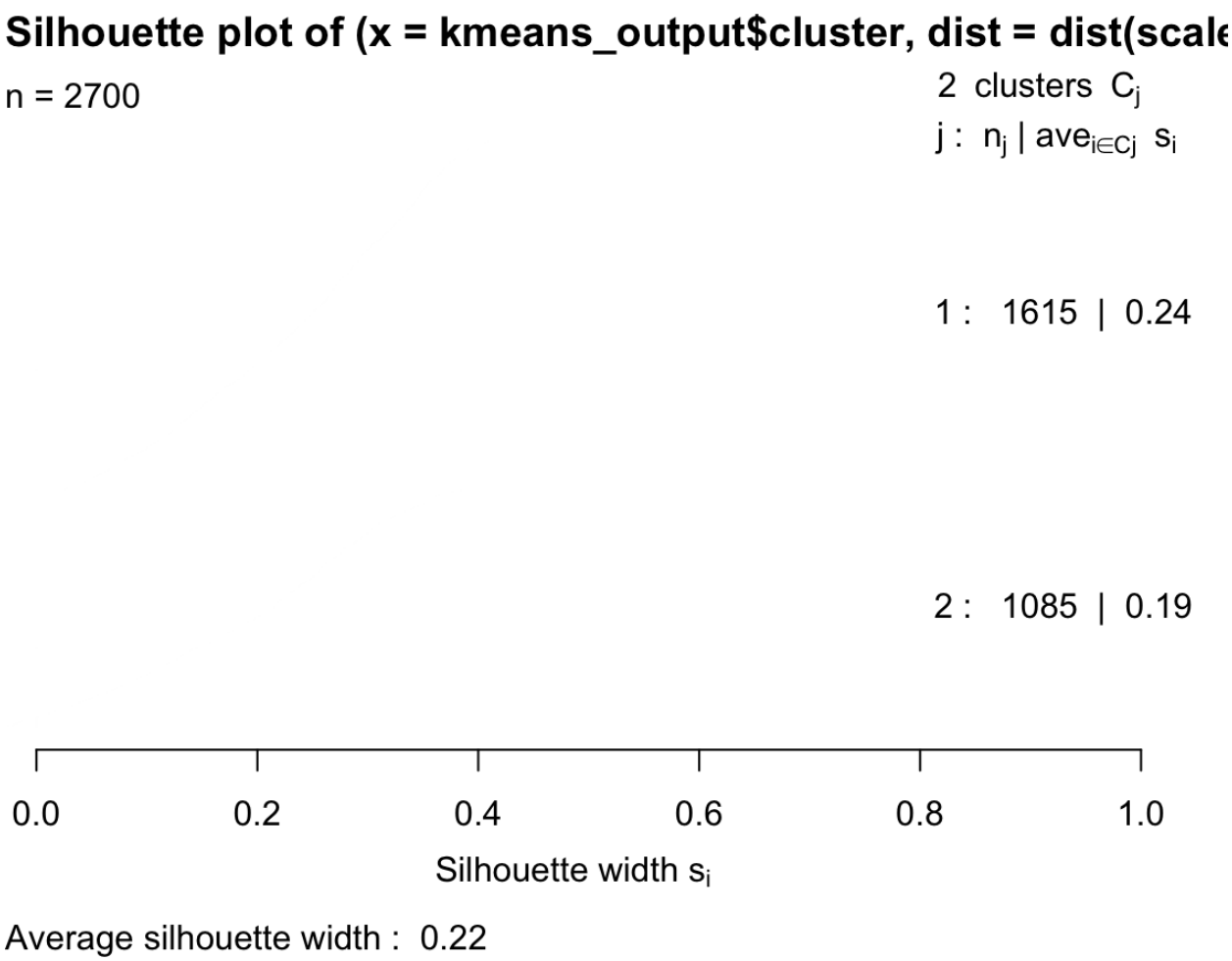
- The silhouette plot illustrates how well each data point fits within its assigned cluster and how distinct the clusters are from each other.
- Higher silhouette values indicate better-defined clusters:

Values close to 1 suggest well-separated clusters.

Values around 0 indicate overlapping clusters.

Negative values suggest points might have been assigned to the wrong cluster.

- The average silhouette width provides an overall measure of clustering quality, with higher values indicating better clustering.




```
> print(avg_silhouette)
[1] 0.2189387
```

According to the above average silhotte width is 0.22

1.2 Subtask Objective

1.2.e

```r
#e
# Perform Principal Component Analysis (PCA)
pca_result <- prcomp(scaled_data,center= TRUE,scale = TRUE)

# Eigenvalues and eigenvectors
eigenvalues <- pca_result$sdev^2
eigenvectors <- pca_result$rotation

# Cumulative score per principal components
cumulative_score <- cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2))

# Select principal components with cumulative score > 85%
selected_PCs <- which(cumulative_score > 0.85)[1]

# Transform the dataset using selected principal components
transformed_data <- as.data.frame(predict(pca_result, newdata = wine_data)[, 1:selected_PCs])

# Print the number of selected principal components
print(selected_PCs)
```

Perform Principal Component Analysis (PCA):

- **pca_result <- prcomp(scaled_data, center = TRUE, scale = TRUE)**:

Conducts PCA on the scaled data.

center = TRUE ensures that the data is centered to have a mean of 0.

scale = TRUE ensures that the data is scaled to have a standard deviation of 1.

Eigenvalues and Eigenvectors:

- **eigenvalues <- pca_result$sdev^2**:

Calculates the eigenvalues of the covariance matrix. These represent the variance explained by each principal component.

- **eigenvectors <- pca_result$rotation**:

Extracts the eigenvectors (principal component loadings), which indicate the direction of the principal components in the feature space.

Cumulative Score per Principal Components:

- **cumulative_score <- cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2))**:

Computes the cumulative proportion of variance explained by the principal components.

cumsum() calculates the cumulative sum of the variance proportions.

Select Principal Components with Cumulative Score > 85%:

- **selected_PCs <- which(cumulative_score > 0.85)[1]**:

Identifies the number of principal components needed to explain at least 85% of the total variance.
which(cumulative_score > 0.85)[1] finds the first principal component where the cumulative score exceeds 85%.

Transform the Dataset Using Selected Principal Components:

- **transformed_data <- as.data.frame(predict(pca_result, newdata = wine_data)[, 1:selected_PCs])**:

Projects the original data onto the selected principal components, reducing its dimensionality.
predict(pca_result, newdata = wine_data) computes the PCA scores for the original dataset.
[, 1:selected_PCs] selects the columns corresponding to the selected principal components.
as.data.frame() converts the result to a data frame.

Print the Number of Selected Principal Components:

- **print(selected_PCs)**:

Prints the number of principal components selected based on the 85% cumulative variance threshold.

```
> print(selected_PCs)
[1] 7
```

According to the above no of Principal Components used is 7.

1.2.f

```r
#f
# Determine the number of clusters using the NbClust method for PCA-transformed data
set.seed(123)
clusterNo_pca <- NbClust(transformed_data, distance = "euclidean", min.nc = 2, max.nc = 10, method

# Determine the number of clusters using the Elbow method for PCA-transformed data
fviz_nbclust(transformed_data, kmeans, method = "wss")

# Determine the number of clusters using the Gap statistics method for PCA-transformed data
fviz_nbclust(transformed_data, kmeans, method = 'gap_stat')

# Determine the number of clusters using the silhouette method for PCA-transformed data
fviz_nbclust(transformed_data, kmeans, method = 'silhouette')
```

Set Seed for Reproducibility:

- **set.seed(123)**:

Sets the seed for random number generation to ensure the reproducibility of the clustering results.

Determine Number of Clusters Using NbClust Method:

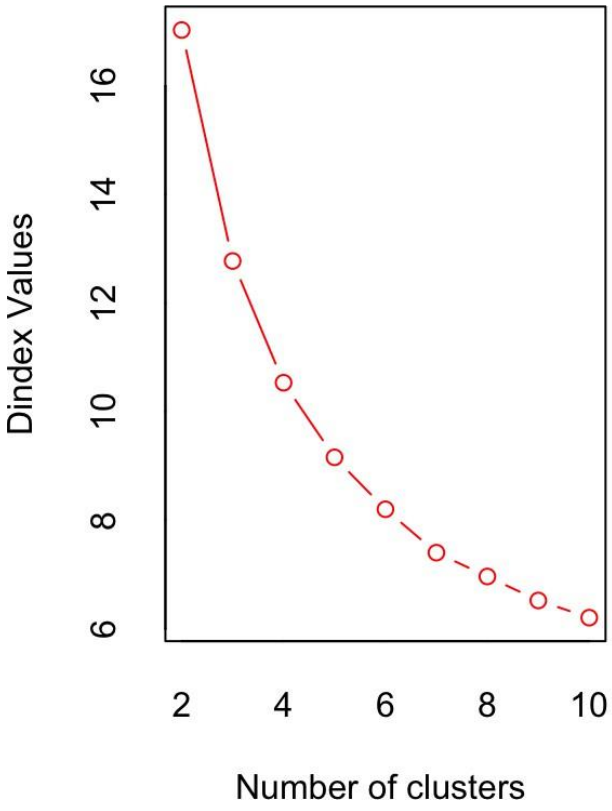
- **clusterNo_pca <- NbClust(transformed_data, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")**:

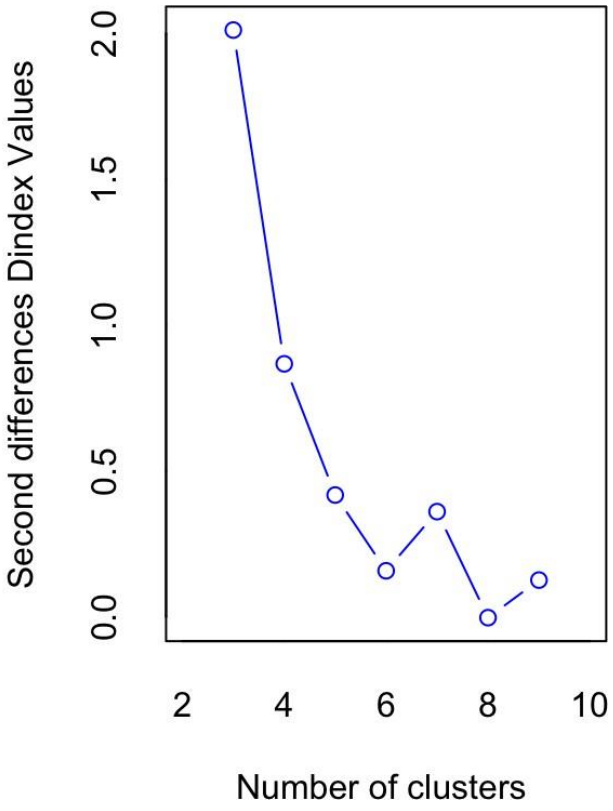
Uses the **NbClust** package to determine the optimal number of clusters in the PCA-transformed data.

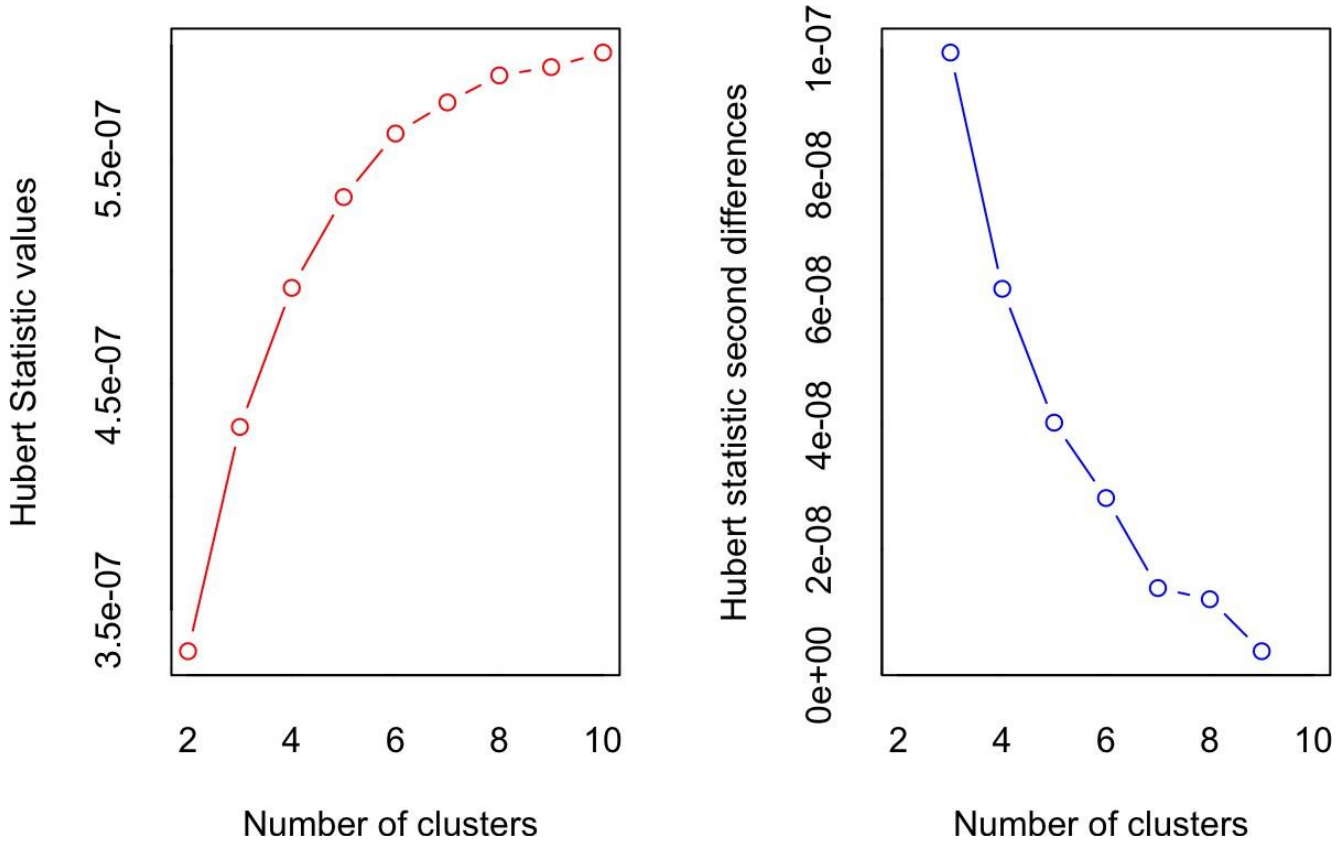
distance = "euclidean" specifies the distance measure to be Euclidean.

min.nc = 2 and **max.nc = 10** set the range for the number of clusters to be considered.

method = "kmeans" specifies the clustering method to be k-means.


Dindex Values
6
8
10
12
14
16
2
4
6
8
10
Number of clusters


Second differences Dindex Values
0.0
0.5
1.0
1.5
2.0
2
4
6
8
10
Number of clusters




```
                ***** Conclusion *****

* According to the majority rule, the best number of clusters is  3
```

According to the above no.of clusters is 3

Determine Number of Clusters Using Elbow Method:

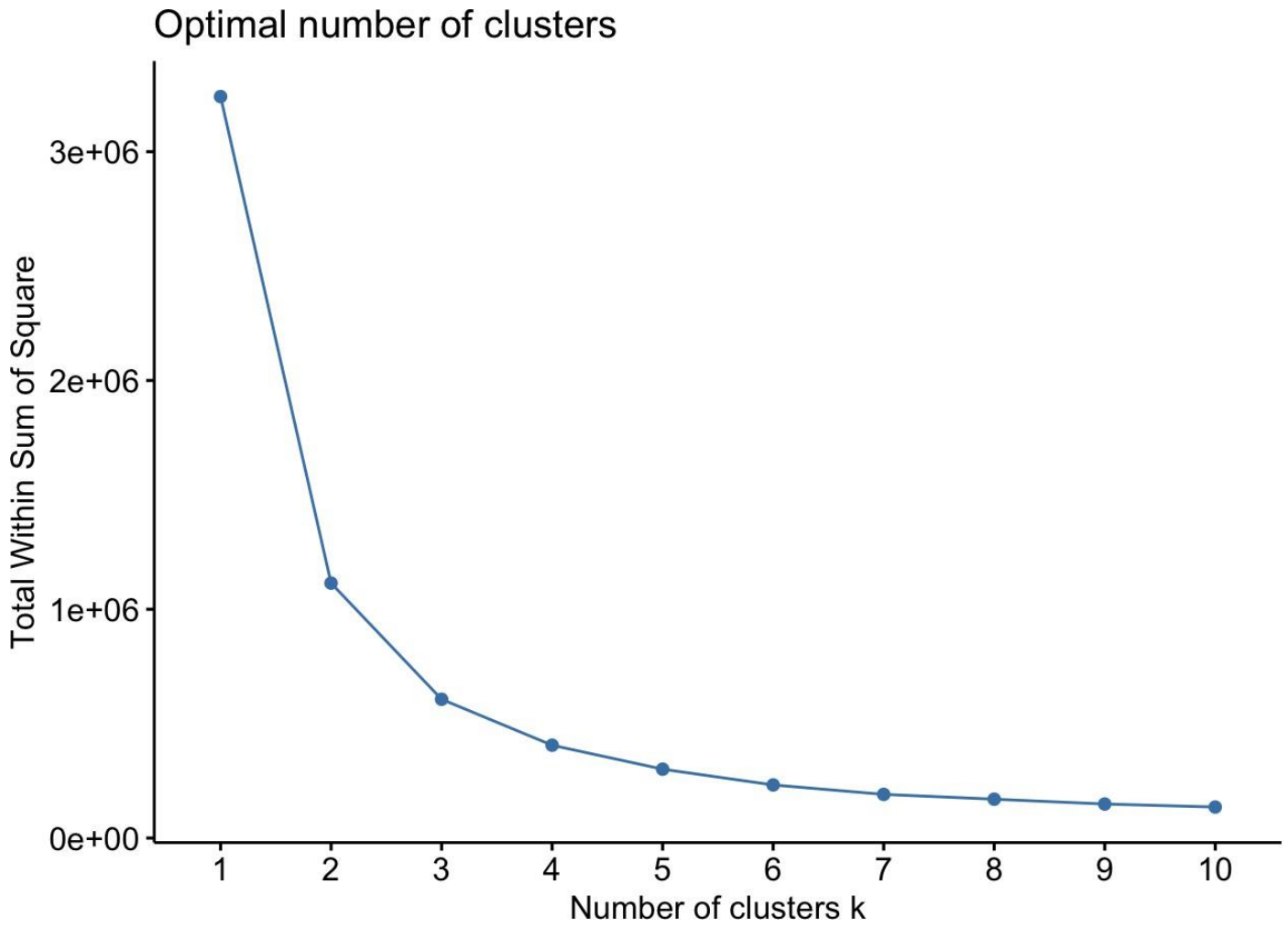
- **fviz_nbclust(transformed_data, kmeans, method = "wss")**:

Uses the **fviz_nbclust** function from the **factoextra** package to determine the optimal number of clusters based on the Elbow method.

method = "wss" stands for "within-cluster sum of squares", which is the measure used by the Elbow method to assess the variance within clusters.

Plots the within-cluster sum of squares against the number of clusters, with the optimal number indicated by the "elbow" point where the curve starts to flatten.

- According to the above no.of clusters is 2
- According to the above no.of clusters is 2

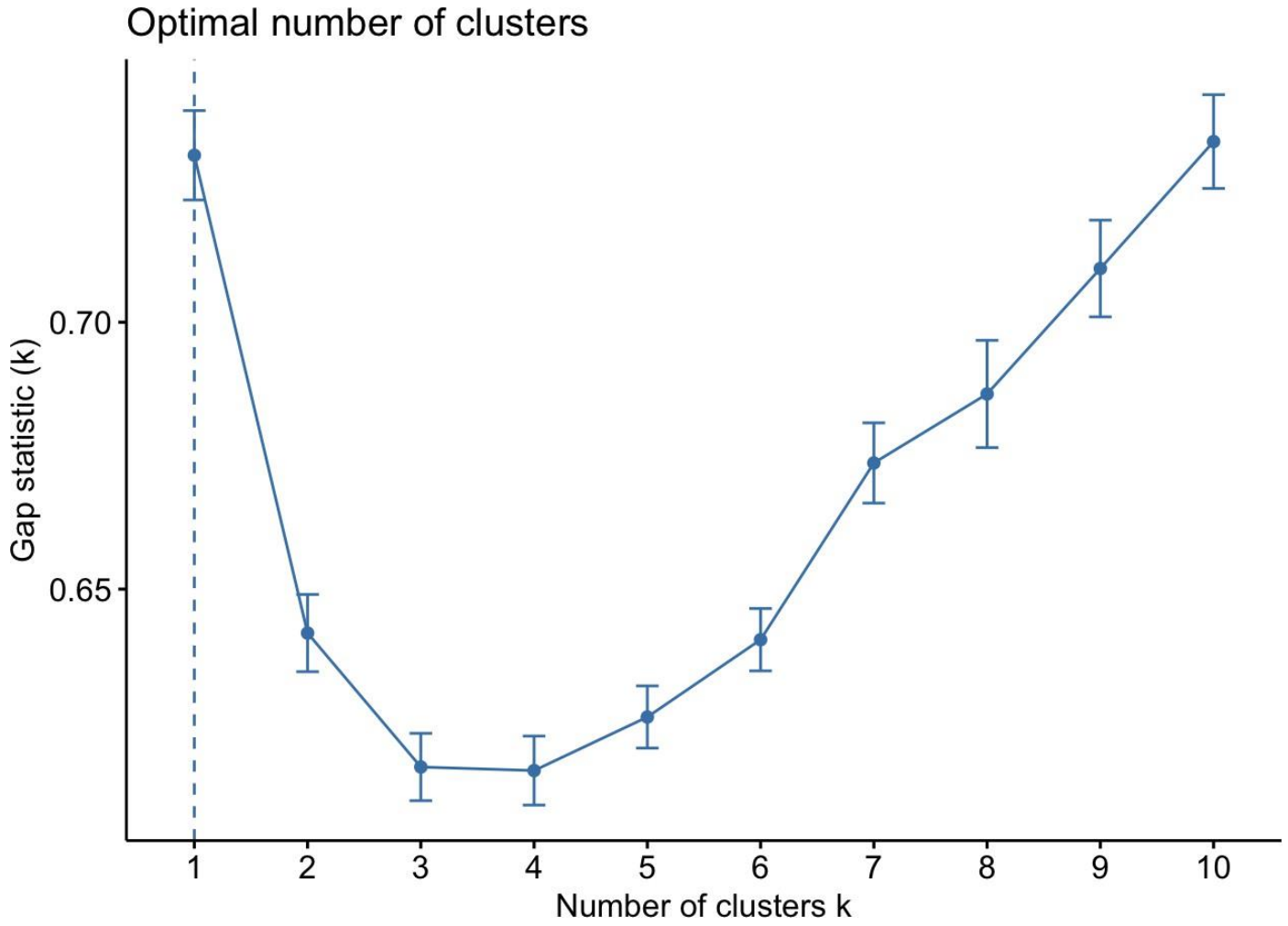


According to the above no.of clusters is 3

Determine Number of Clusters Using Gap Statistics Method:

- **fviz_nbclust(transformed_data, kmeans, method = 'gap_stat')**:

Uses the **fviz_nbclust** function to determine the optimal number of clusters based on Gap statistics. **method = "gap_stat"** uses the gap statistic method, which compares the total within intra-cluster variation for different numbers of clusters with their expected values under null reference distribution of the data. Plots the gap statistic values, with the optimal number of clusters indicated by the maximum gap value.



According to the above no.of clusters is 1

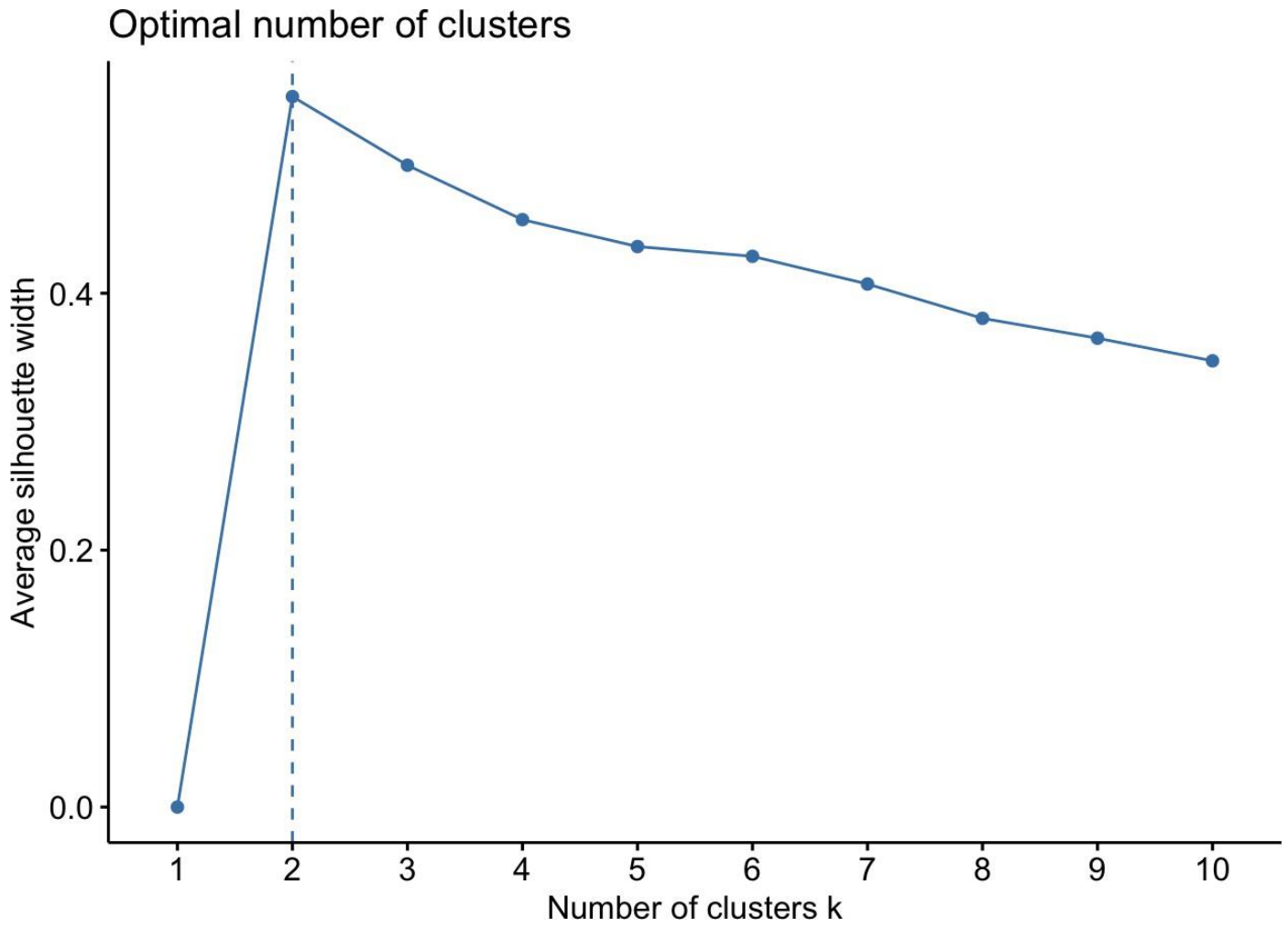
Determine Number of Clusters Using Silhouette Method:

- **fviz_nbclust(transformed_data, kmeans, method = 'silhouette')**:

Uses the **fviz_nbclust** function to determine the optimal number of clusters based on the Silhouette method.

method = "silhouette" uses the average silhouette width to evaluate the quality of clustering.

Plots the silhouette values, with the optimal number of clusters indicated by the maximum average silhouette width.



According to the above no.of clusters is 2

So most of Automated tools show the no of clusters are 2.Therefor k=3

1.2.g

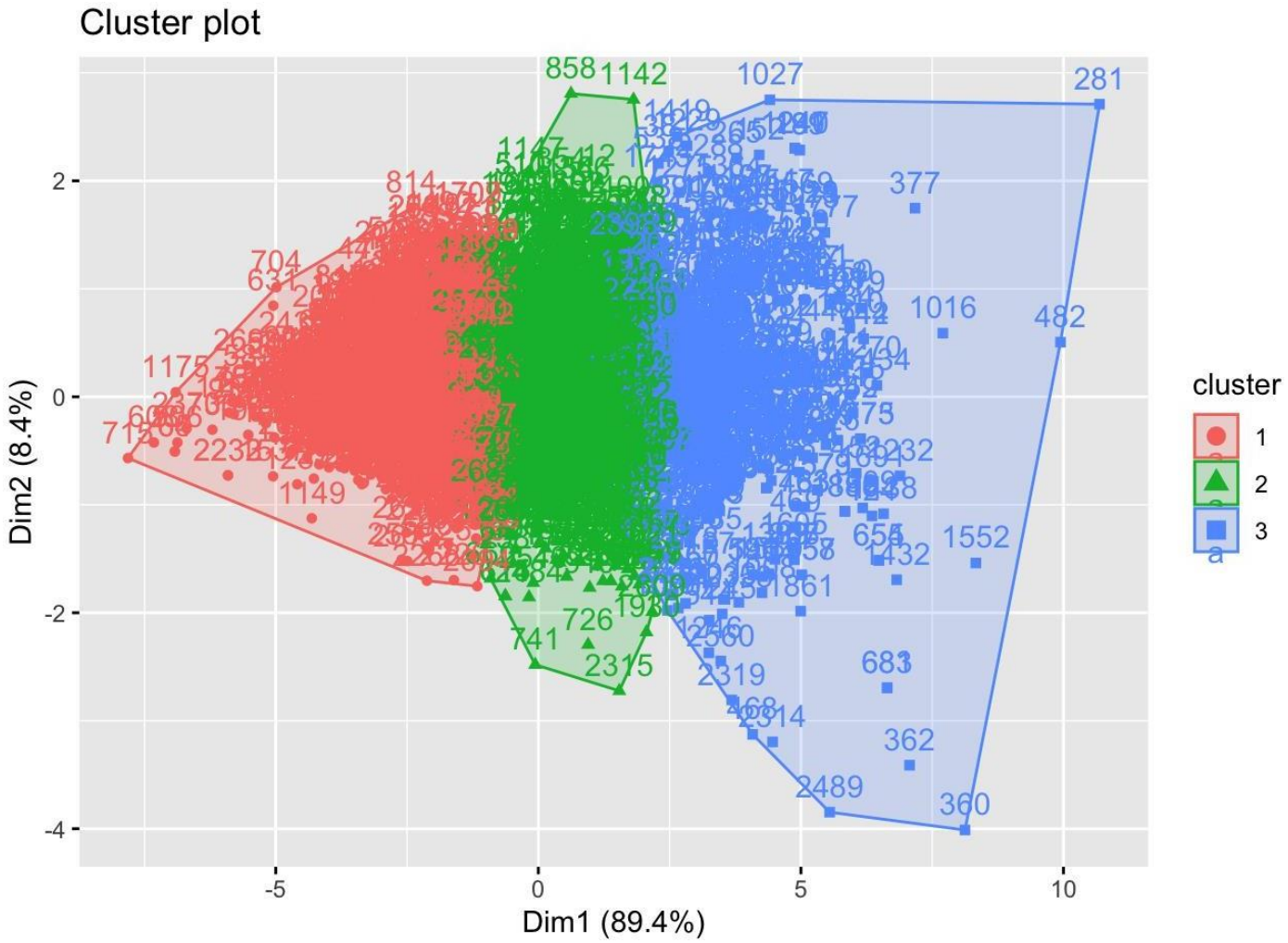
```r
#g
# Apply k-means clustering with k=3 on the PCA-transformed data
k <- 3
set.seed(123)
kmeans_model_pca <- kmeans(transformed_data, centers = k)
kmeans_model_pca

# Visualize clustering results
fviz_cluster(kmeans_model_pca, data = transformed_data)

# Calculate BSS, WSS, and TSS for PCA-based k-means
BSS_pca <- sum(kmeans_model_pca$betweenss)
WSS_pca <- kmeans_model_pca$tot.withinss
TSS_pca <- BSS_pca + WSS_pca
BSS_ratio_pca <- BSS_pca / TSS_pca

# Print k-means output
print(kmeans_model_pca)
print(BSS_ratio_pca)
```

Set Number of Clusters and Seed:

- **k <- 3**: Sets the number of clusters *k* to 3 for k-means clustering.
- **set.seed(123)**: Sets the seed for random number generation to ensure reproducibility of the clustering results.
-

Apply k-Means Clustering on PCA-transformed Data:

- **kmeans_model_pca <- kmeans(transformed_data, centers = k)**: Applies k-means clustering to the PCA-transformed data with 3 clusters.
- **kmeans_model_pca**: Stores the k-means clustering model.
- **Visualize Clustering Results**:
- **fviz_cluster(kmeans_model_pca, data = transformed_data)**: Visualizes the clustering results using the **fviz_cluster** function from the **factoextra** package. This function plots the data points colored by their assigned clusters and displays cluster centers.



Calculate BSS, WSS, and TSS for PCA-based k-Means:

- **BSS_pca <- sum(kmeans_model_pca$betweenss)**: Calculates the Between Sum of Squares (BSS), which measures the variation between different clusters.
- **WSS_pca <- kmeans_model_pca$tot.withinss**: Retrieves the Total Within Sum of Squares (WSS), which measures the variation within each cluster.
- **TSS_pca <- BSS_pca + WSS_pca**: Calculates the Total Sum of Squares (TSS) by summing BSS and WSS. TSS represents the total variation in the data.
- **BSS_ratio_pca <- BSS_pca / TSS_pca**: Calculates the ratio of BSS to TSS, indicating the proportion of the total variance that is explained by the clustering.

Print k-Means Output:

- **print(kmeans_model_pca)**: Prints the complete k-means clustering output, including cluster centers, cluster assignment of each data point, and clustering statistics.
- **print(BSS_ratio_pca)**: Prints the BSS ratio, showing the effectiveness of the clustering in terms of explained variance.

```
[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
> print(BSS_ratio_pca)
[1] 0.812834
```

According to the above BSS ratio is 0.812834

1.2.h

```
150 #h
151 # Silhouette plot for PCA-based k-means
152 silhouette_plot_pca <- silhouette(kmeans_model_pca$cluster, dist(transformed_data))
153 plot(silhouette_plot_pca)
154
155 # Calculate average silhouette width score for PCA-based k-means
156 avg_silhouette_pca <- mean(silhouette_plot_pca[, 3])
157 print(avg_silhouette_pca)
```

Silhouette Plot for PCA-based k-Means:

- **silhouette_plot_pca <- silhouette(kmeans_model_pca$cluster, dist(transformed_data))**:

Calculates the silhouette information for each data point based on the cluster assignments obtained from PCA-based k-means clustering.

kmeans_model_pca$cluster provides the cluster assignments for each data point.

dist(transformed_data) computes the distance matrix for the PCA-transformed data.

- **plot(silhouette_plot_pca)**: Plots the silhouette values, visualizing how well each data point fits within its cluster and how distinct each cluster is from the others.

Calculate Average Silhouette Width Score for PCA-based k-Means:

- **avg_silhouette_pca <- mean(silhouette_plot_pca[, 3])**:

Computes the average silhouette width score for the PCA-based k-means clustering.

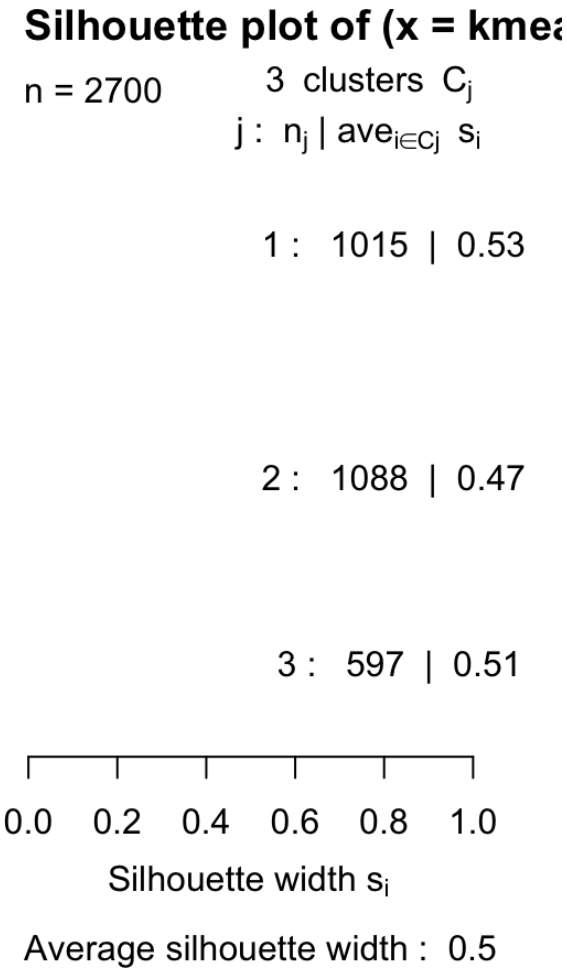
silhouette_plot_pca[, 3] extracts the silhouette widths for each data point.

mean() calculates the average silhouette width across all data points.

Print Average Silhouette Width Score:

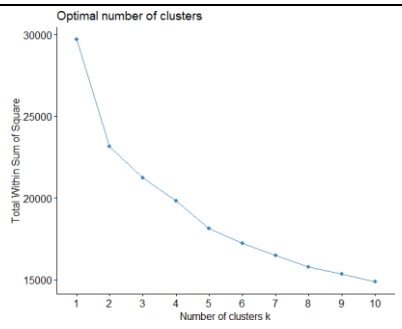
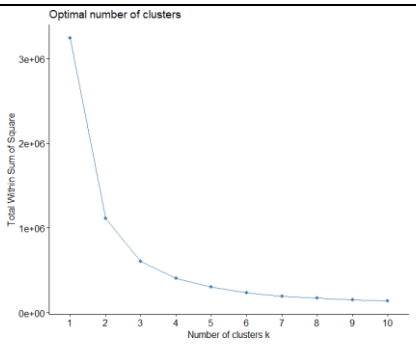
- **print(avg_silhouette_pca)**: Prints the calculated average silhouette width score, which is a measure of clustering quality.

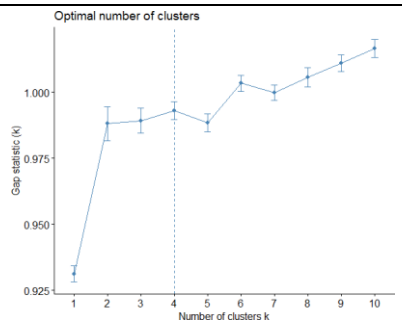
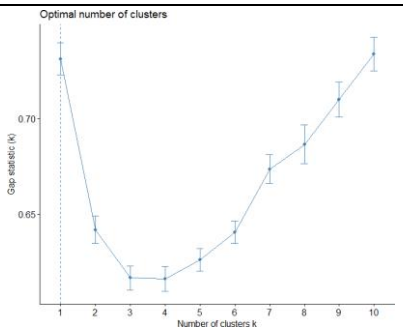
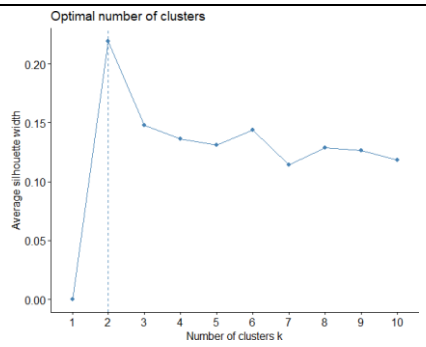
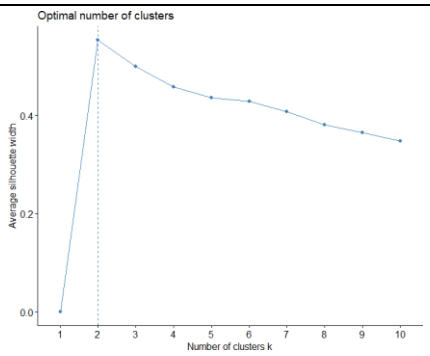
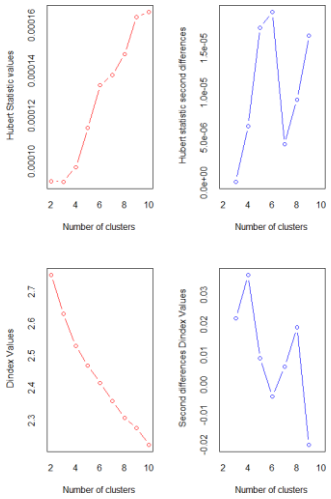
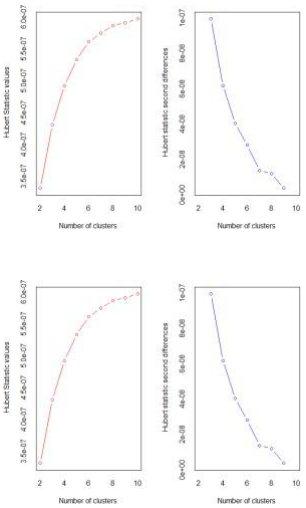
```
> print(avg_silhouette_pca)
[1] 0.4996241
> #i
```

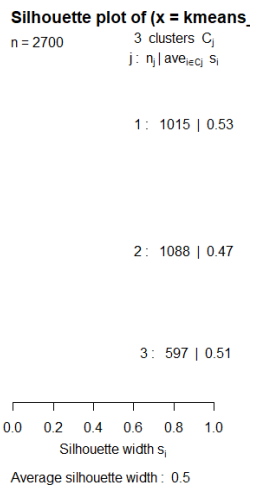
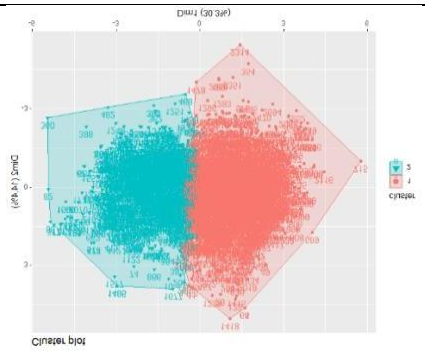
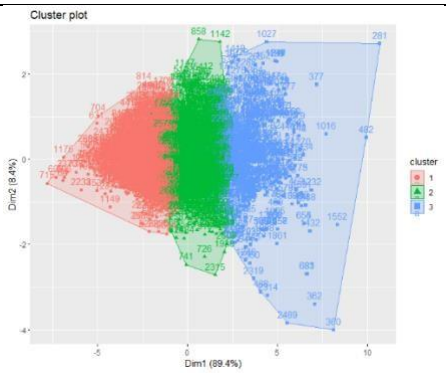


According to the above average silhotte width is 0.5

Comparison

	Before PCA	After PCA
Elbow Method After PCA: **well defined output**		

Gap Static Method After PCA : **Maybe Higher. In this case same.**		
Silhouette Method After PCA: **Sometimes It will be higher. In this case same.**		
NB Cluster Method After PCA: **Low Value**		

Silhouette Plot After PCA: **Plot Increases/ Average width increases.**		
Number Of Clusters After PCA: **Number of clusters maybe increases. In this case It is.**		

1.2.i

```r
#i
# Calculate the Calinski-Harabasz Index
calinski_harabasz_pca <- function(cluster_result, data) {
  k <- length(unique(cluster_result$cluster))
  n <- nrow(data)
  BSS <- cluster_result$betweenss
  WSS <- cluster_result$tot.withinss
  ch_index <- ((n - k) / (k - 1)) * (BSS / WSS)
  return(ch_index)
}

ch_index_pca <- calinski_harabasz_pca(kmeans_model_pca, transformed_data)
ch_index_pca
```

Define Calinski-Harabasz Index Calculation Function:

- Defines a function (**calinski_harabasz_pca**) that calculates the index using the number of clusters, the number of data points, and clustering statistics.

Calculate Calinski-Harabasz Index for PCA-based k-means:

- Calls the function with the PCA-based k-means clustering result and the transformed data as input.

Print Calinski-Harabasz Index:

- Displays the calculated index, which indicates the clustering quality. Higher values imply better-defined clusters.

formula –[(n-k)/BSS]/[(k-1)/WSS]

```
> ch_index_pca
[1] 5856.336
```

According to the above CH Index is 5856.336

Financial Forecasting Part

Discussion on MLP models

The process of forecasting, which is an essential function for business and finance organisations, involves the use of time series analysis to anticipate changes in currency exchange rates. A number of economic factors influence exchange rates, which indicate the value of one currency compared to another. It is hard to predict currency exchange rates, because of their complexity, instability and inherent volatility. The historical exchange rate data are the basis of financial forecasting models, which capture past trends and fluctuations in currency values over time. The use of NNs to predict exchange rates is becoming increasingly popular because they are capable of identifying complex patterns and relationships in the datasets.

For the forecasting of exchange rates based on delayed exchange rate values, the AR method is widely applied to determine input vectors in NN models. Other approaches include:

1. Moving Average Approach: as input variables, the moving average of the exchange rate series shall be used in this approach.
2. Hybrid Approach: the combination of an AAR and MA approach with other technical indicators such as Relative Strength IndexRSI, Moving Average Convergence Divergence MACD or Bollinger Bands is also a hybrid strategy. These indicators may provide more detailed information on the state of the market and can improve the accuracy of the model.
3. Economic and Fundamental Approach: Macroeconomic and fundamental variables that affect exchange rates shall be used as input variables in this approach. These variables include interest rates, inflation rate, GDP growth and balance of trade data.
4. Non-linear Autoregressive (NAR) Approach with Exogenous Inputs (NARX): The NARX method expands on the AR approach by incorporating external inputs such as Economic Statistics, News Opinion or any relevant factors which may have an impact on exchange rates. The input vector in this approach consists of the lag values of the exchange rate series and of the exogenous inputs.

The best way to feed vectors into MLP models for forecasting exchange rates relies on the particular requirements and the accessibility of data. Every technique has its advantages and

disadvantages, so it is important to carefully consider the pros and cons of each method in order to determine which one will suit you best. The accuracy of their exchange rate forecasts may be improved by financial institutions that use MLP models and carefully select input factors.

Input/Output Metrics

The "I/O Matrix" is a fundamental tool for neural network analysis, connecting input variables to output predictions. This matrix is needed for training and forecasting in the network so as to allow a deeper understanding of its dynamics, which will make it easier to improve computing intelligence.

Case (T-4)

```
# A tibble: 496 × 5
   input_1 input_2 input_3 input_4  rate
     <dbl>   <dbl>   <dbl>   <dbl> <dbl>
 1    1.37    1.38    1.39    1.37  1.38
 2    1.38    1.37    1.38    1.39  1.37
 3    1.37    1.38    1.37    1.38  1.39
 4    1.39    1.37    1.38    1.37  1.39
 5    1.39    1.39    1.37    1.38  1.39
 6    1.39    1.39    1.39    1.37  1.38
 7    1.38    1.39    1.39    1.39  1.42
 8    1.42    1.38    1.39    1.39  1.42
 9    1.42    1.42    1.38    1.39  1.39
10    1.39    1.42    1.42    1.38  1.37
```

Case(T-3)

```
# A tibble: 497 × 4
   input_1 input_2 input_3  rate
     <dbl>   <dbl>   <dbl> <dbl>
 1    1.38    1.39    1.37  1.37
 2    1.37    1.38    1.39  1.38
 3    1.38    1.37    1.38  1.37
 4    1.37    1.38    1.37  1.39
 5    1.39    1.37    1.38  1.39
 6    1.39    1.39    1.37  1.39
 7    1.39    1.39    1.39  1.38
 8    1.38    1.39    1.39  1.42
 9    1.42    1.38    1.39  1.42
10    1.42    1.42    1.38  1.39
# A tibble: 6 × 4
  input_1 input_2 input_3  rate
    <dbl>   <dbl>   <dbl> <dbl>
1    1.38    1.39    1.37  1.37
2    1.37    1.38    1.39  1.38
3    1.38    1.37    1.38  1.37
4    1.37    1.38    1.37  1.39
5    1.39    1.37    1.38  1.39
6    1.39    1.39    1.37  1.39
```

Case(T-2)

```
# A tibble: 498 × 3
   input_1 input_2  rate
     <dbl>   <dbl> <dbl>
 1    1.39    1.37  1.38
 2    1.38    1.39  1.37
 3    1.37    1.38  1.38
 4    1.38    1.37  1.37
 5    1.37    1.38  1.39
 6    1.39    1.37  1.39
 7    1.39    1.39  1.39
 8    1.39    1.39  1.38
 9    1.38    1.39  1.42
10    1.42    1.38  1.42
```

Case(T-4)

```
# A tibble: 499 × 2
   input_1  rate
     <dbl> <dbl>
 1    1.37  1.39
 2    1.39  1.38
 3    1.38  1.37
 4    1.37  1.38
 5    1.38  1.37
 6    1.37  1.39
 7    1.39  1.39
 8    1.39  1.39
 9    1.39  1.38
10    1.38  1.42
```

Normalisation

Input normalisation is essential for the MultiLayer PerceptronMLP structure, prior to use. As MLPs are sensitive to the scale of input variables, which could lead to potential performance

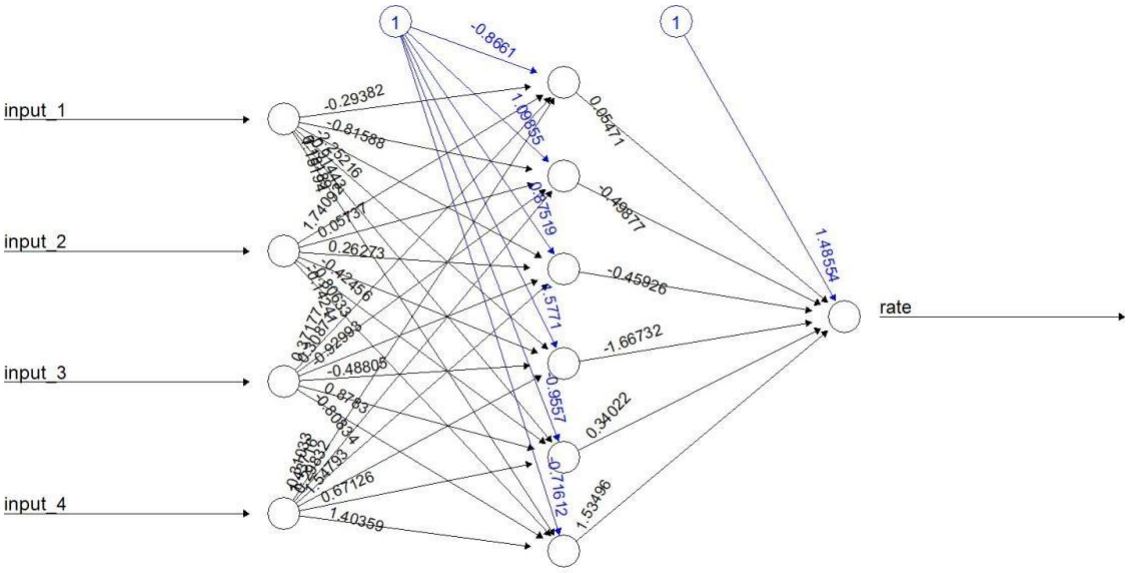
problems if not normalised, this step is crucial for training the neural network. Normalisation is to scale the data to a uniform range, typically between 0 and 1 or -1 and 1, to ensure that each input variable contributes proportionately without being dominated by variables of a larger size. In addition, normalisation mitigates the risk that neurons will be trapped within a localised minimum by accelerating training. Data normalisation contributes to a more efficient convergence of the network by preventing excessive growth or contraction of the curve. For normalisation purposes, the minimax formula was used in this study.

```r
#Function to normalize data
normalize<-function(x){
  return((x-min(x))/(max(x)-min(x)))
}

# Function to renormalize data
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}
```

Implementation of different Models.

MODEL-1 (4 INPUTS)

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 1 using 4 variables with one hidden layer
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

exchange_model_T4_1<-neuralnet(formula_T4 ,hidden = 6 ,
                               data=exchangeRate_train_T4,linear.output=TRUE)
plot(exchange_model_T4_1)
model_results_T4_1<-predict(exchange_model_T4_1,exchangeRate_test_T4)

# Unnormalize the predicted values
exchangeRate_pred_T4_1 <- unnormalize(model_results_T4_1, exchangeRate_min_T4,
                                      exchangeRate_max_T4)
head(exchangeRate_pred_T4_1)

#Calculate performance metrics
rmse_value_T4_1 <- rmse(exchangeRate_test_T4$rate, exchangeRate_pred_T4_1)
mae_value_T4_1 <- mae(exchangeRate_test_T4$rate, exchangeRate_pred_T4_1)
mape_value_T4_1 <- mape(exchangeRate_test_T4$rate, exchangeRate_pred_T4_1)
smape_value_T4_1<-smape(exchangeRate_test_T4$rate,exchangeRate_pred_T4_1)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T4_1))
print(paste("MAE: ", mae_value_T4_1))
print(paste("MAPE: ", mape_value_T4_1))
print(paste("SMAPE: ", smape_value_T4_1))
```

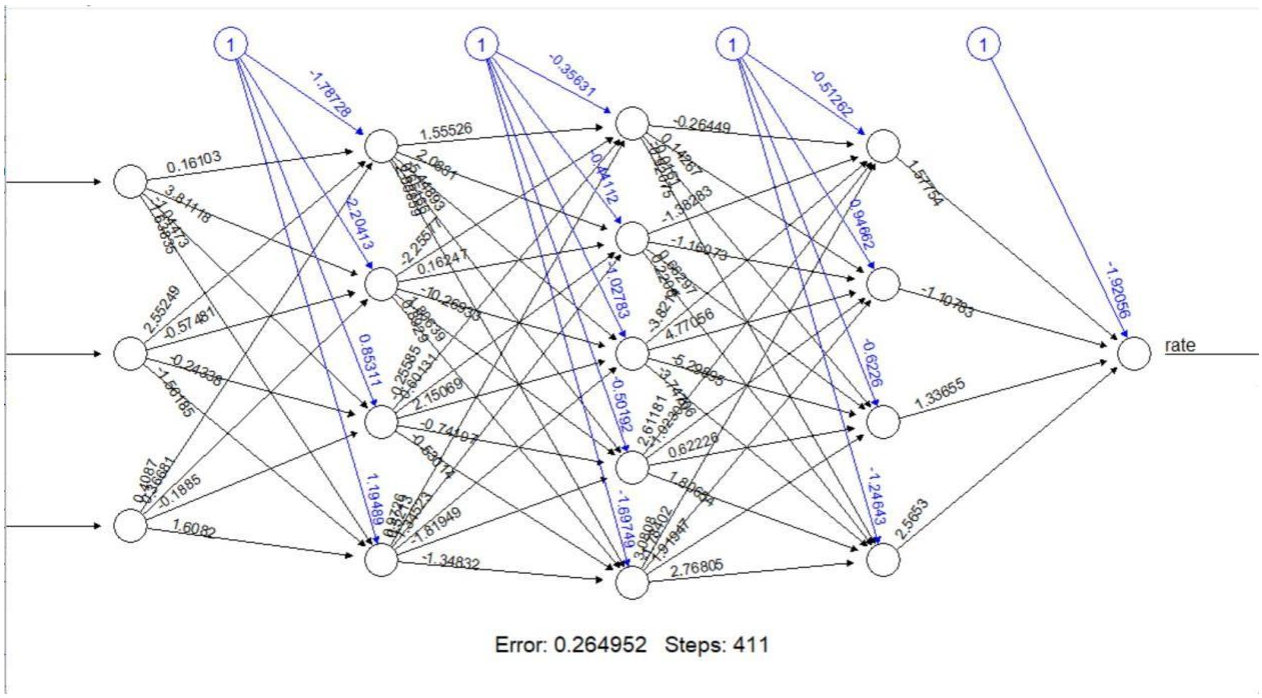


Error: 0.251002 Steps: 580

Similarly, different numbers of nodes were used to create two further models in Model 2,3, which have been concealed. layers and linear/nonlinear output with 4 inputs.

MODEL-6 (3 INPUTS)

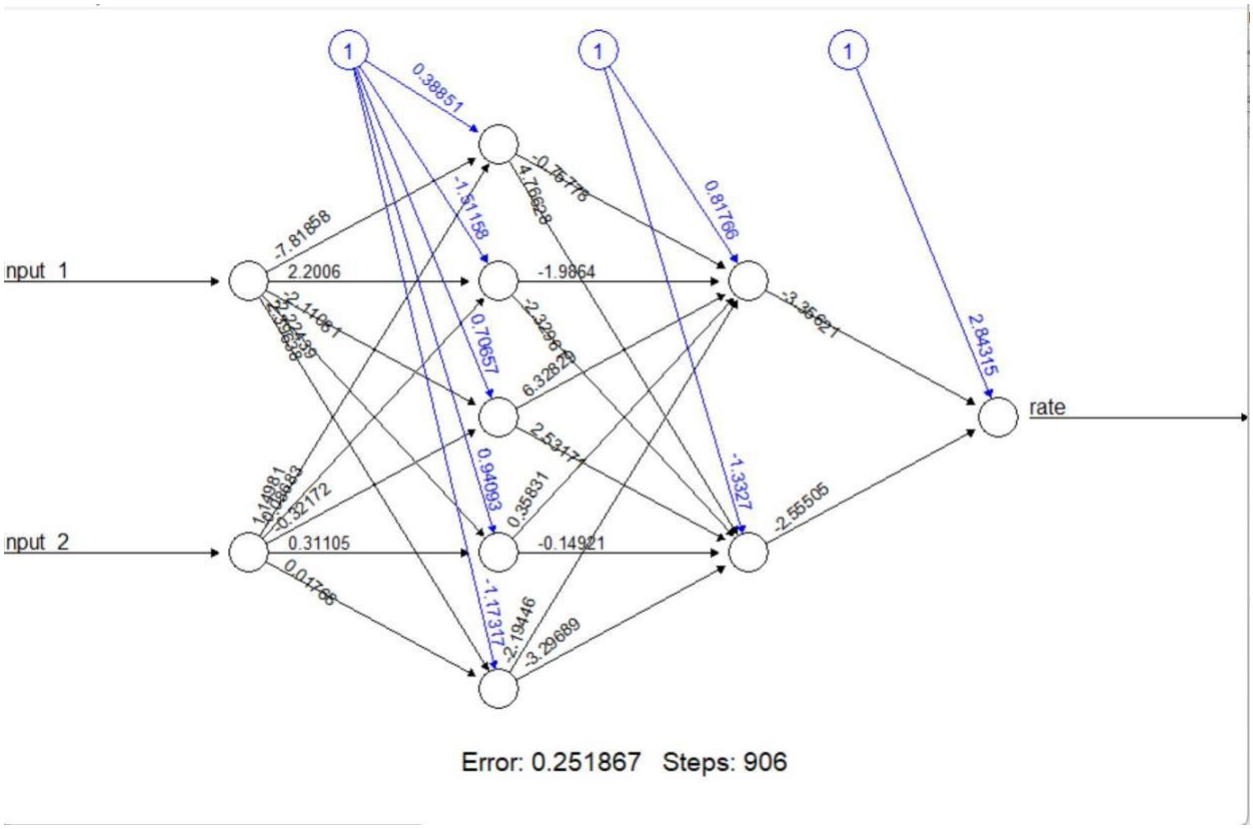
```r
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 6 using 3 variables with Three hidden layer
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

exchange_model_T3_3<-neuralnet(formula_T3 ,hidden = c(4,5,4) ,
                               data=exchangeRate_train_T3,linear.output=FALSE)
plot(exchange_model_T3_3)
model_results_T3_3<-predict(exchange_model_T3_3,exchangeRate_test_T3)

# Unnormalize the predicted values
exchangeRate_pred_T3_3 <- unnormalize(model_results_T3_3, exchangeRate_min_T3,
                                      exchangeRate_max_T3)
head(exchangeRate_pred_T3_3)

#Calculate performance metrics
rmse_value_T3_3 <- rmse(exchangeRate_test_T3$rate, exchangeRate_pred_T3_3)
mae_value_T3_3 <- mae(exchangeRate_test_T3$rate, exchangeRate_pred_T3_3)
mape_value_T3_3 <- mape(exchangeRate_test_T3$rate, exchangeRate_pred_T3_3)
smape_value_T3_3<-smape(exchangeRate_test_T3$rate,exchangeRate_pred_T3_3)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T3_3))
print(paste("MAE: ", mae_value_T3_3))
print(paste("MAPE: ", mape_value_T3_3))
print(paste("SMAPE: ", smape_value_T3_3))
```



Similarly, different numbers of nodes were used to create two further models in Model 4,5, which have been concealed. layers and linear/nonlinear output with 3 inputs.

MODEL-9 (2 INPUTS)

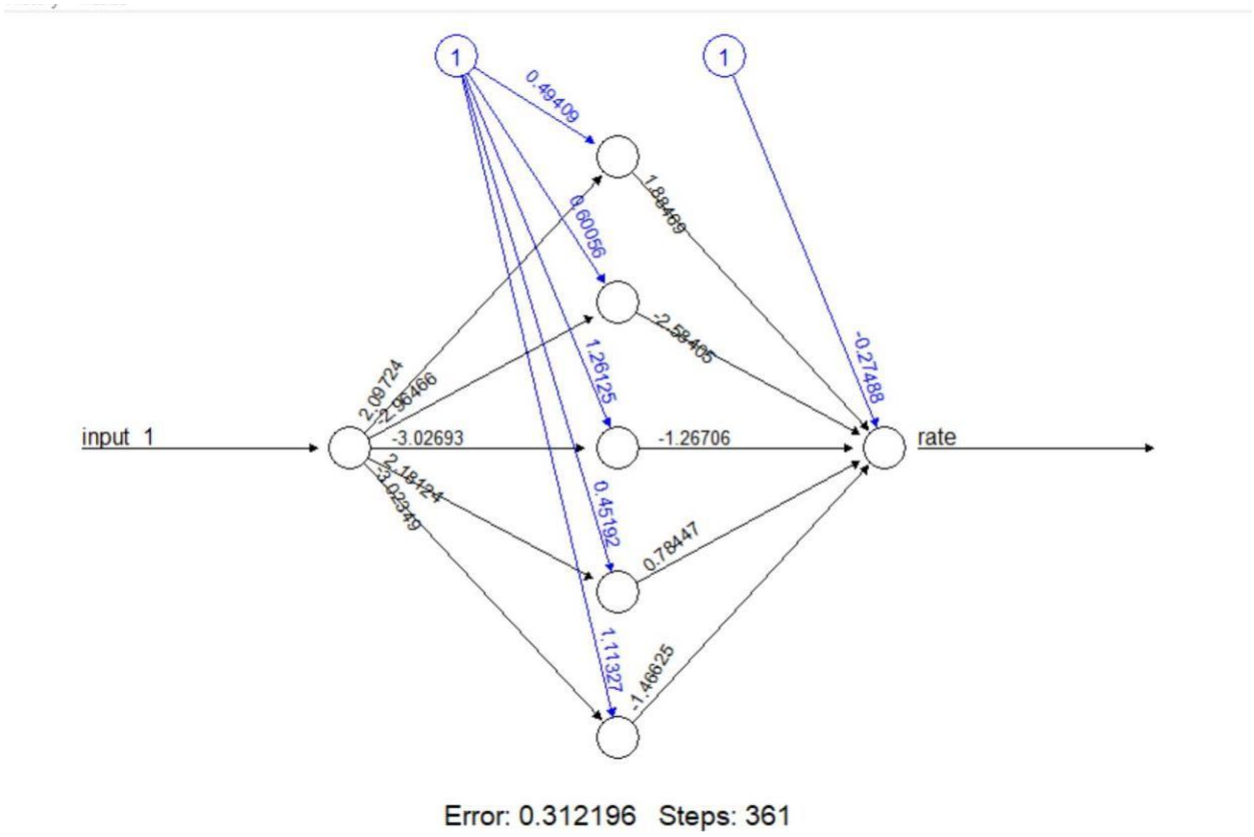
```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 8 using 2 variables with one hidden layer
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

exchange_model_T2_2<-neuralnet(formula_T2 ,hidden = 10 ,
                               data=exchangeRate_train_T2,linear.output=TRUE)
plot(exchange_model_T2_2)
model_results_T2_2<-predict(exchange_model_T2_2,exchangeRate_test_T2)

# Unnormalize the predicted values
exchangeRate_pred_T2_2 <- unnormalize(model_results_T2_2, exchangeRate_min_T2,
                                      exchangeRate_max_T2)
head(exchangeRate_pred_T2_2)

#Calculate performance metrics
rmse_value_T2_2 <- rmse(exchangeRate_test_T2$rate, exchangeRate_pred_T2_2)
mae_value_T2_2 <- mae(exchangeRate_test_T2$rate, exchangeRate_pred_T2_2)
mape_value_T2_2 <- mape(exchangeRate_test_T2$rate, exchangeRate_pred_T2_2)
smape_value_T2_2<-smape(exchangeRate_test_T2$rate,exchangeRate_pred_T2_2)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T2_2))
print(paste("MAE: ", mae_value_T2_2))
print(paste("MAPE: ", mape_value_T2_2))
print(paste("SMAPE: ", smape_value_T2_2))

```



Similarly, different numbers of nodes were used to create two further models in Model 7,8, which have been concealed. layers and linear/nonlinear output with 2 inputs.

MODEL-12 (1 INPUTS)

```
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 12 using 1 variables with one hidden layer
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
exchange_model_T1_3<-neuralnet(formula_T1 ,hidden = 5,
                               data=exchangeRate_train_T1,linear.output=FALSE)
plot(exchange_model_T1_3)
model_results_T1_3<-predict(exchange_model_T1_3,exchangeRate_test_T1)

# Unnormalize the predicted values
exchangeRate_pred_T1_3 <- unnormalize(model_results_T1_3, exchangeRate_min_T1, exchangeRate_max_T1)
head(exchangeRate_pred_T1_3)

#Calculate performance metrics
rmse_value_T1_3 <- rmse(exchangeRate_test_T1$rate, exchangeRate_pred_T1_3)
mae_value_T1_3 <- mae(exchangeRate_test_T1$rate, exchangeRate_pred_T1_3)
mape_value_T1_3 <- mape(exchangeRate_test_T1$rate, exchangeRate_pred_T1_3)
smape_value_T1_3<-smape(exchangeRate_test_T1$rate,exchangeRate_pred_T1_3)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T1_3))
print(paste("MAE: ", mae_value_T1_3))
print(paste("MAPE: ", mape_value_T1_3))
print(paste("SMAPE: ", smape_value_T1_3))
```



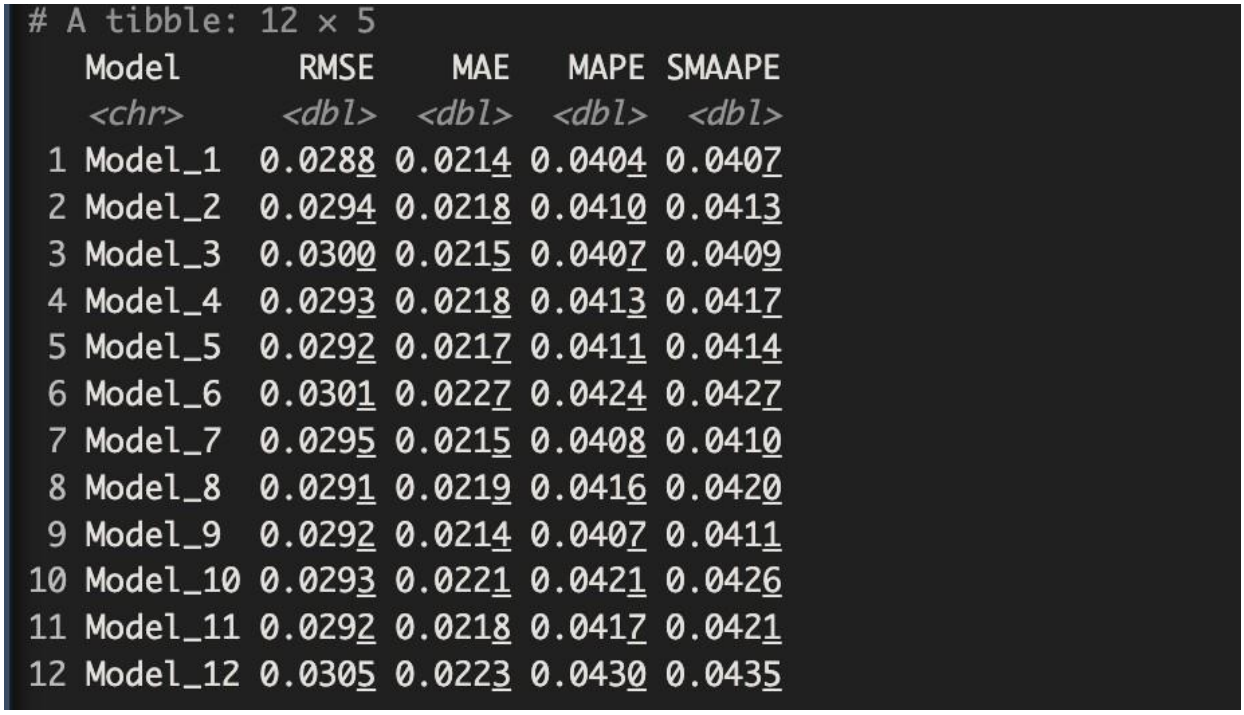
Similarly, different numbers of nodes were used to create two further models in Model 10 ,11, which have been concealed. layers and linear/nonlinear output with 1 inputs.

Comparison Table

Model	Number of inputs	Number of hidden layers	Number of neurons	Linear output
Model 1	4	1	6	TRUE
Model 2		2	c(5,3)	TRUE
Model 3		1	12	FALSE
Model 4	3	1	10	TRUE

Model 5		2	C(5,6)	TRUE

Model 6		3	c(4,5,4)	FALSE
Model 7	2	1	15	FALSE
Model 8		1	10	TRUE
Model 9		2	C(5,2)	FALSE
Model 10	1	1	12	TRUE
Model 11		2	C(5,4)	TRUE
Model 12		1	5	FALSE



```
# A tibble: 12 × 5
   Model      RMSE    MAE   MAPE SMAAPE
   <chr>     <dbl>  <dbl>  <dbl>  <dbl>
 1 Model_1  0.0288 0.0214 0.0404 0.0407
 2 Model_2  0.0294 0.0218 0.0410 0.0413
 3 Model_3  0.0300 0.0215 0.0407 0.0409
 4 Model_4  0.0293 0.0218 0.0413 0.0417
 5 Model_5  0.0292 0.0217 0.0411 0.0414
 6 Model_6  0.0301 0.0227 0.0424 0.0427
 7 Model_7  0.0295 0.0215 0.0408 0.0410
 8 Model_8  0.0291 0.0219 0.0416 0.0420
 9 Model_9  0.0292 0.0214 0.0407 0.0411
10 Model_10 0.0293 0.0221 0.0421 0.0426
11 Model_11 0.0292 0.0218 0.0417 0.0421
12 Model_12 0.0305 0.0223 0.0430 0.0435
```

Comparison of efficiency between one layer and two layers of hidden networks.

Lower values of RMSE, MAE, MAPE, and SMAPE indicate more accurate predictions by a model. Therefore, it suggests that the model is more accurate in predicting values and achieves a better balance between expected and actual values by showing lower RMSE, MAE, MAPE or SMAPE values compared to other models.The best MLP model with one hidden layer and four inputs is determined in the comparison

table as Model 3. Model 3, due to its better performance, has been chosen as the optimum choice despite a lower level of statistically favourable indicethan other MLP models with different concealment layer configurations. This MLP network, consisting of four inputs and a single hidden layer of 12 neurons, was the most efficient of the alternatives, given the lower statistical indices.

Analysis of the Statistical Indices

The relationship between subsets of components may be examined by statistical indices. The purpose of these indices is twofold: to confirm or refute the existence of the rules of law and the correlation that governs their behaviour.

Root Mean Square Error (RMSE)

The RMSE is a widely used metric to assess the performance of regression models. It can measure the average magnitude of errors by calculating a square root for variance between forecast and observed values. It is sensitive to large discrepancies because of square errors before averaging, implying that significant mistakes contribute more than small ones to the overall RMSE. This attribute allows RMSE to benefit in situations where precision is of paramount importance and important errors are unacceptable, as it imposes a more severe penalty on major mistakes compared with the precision-oriented nature of these settings.

Mean Absolute Error (MAE)

Another commonly used statistic for the estimation of regression models is MAE. Averaging the absolute differences decomposes the mean absolute difference of predicted and measured values. Contrary to RMSE, MAE is resistant to outliers because it doesn't square errors before averaging them. Therefore, all errors are equally contributing to the overall MAE value regardless of their size. In cases where a more balanced assessment of model performance is sought, the MAE will prove to be effective, and outliers should not play an undue role in assessing models.

Mean Absolute Percentage Error (MAPE)

By averaging the absolute percentage errors, MAPE, or the mean absolute percentage error, computes the average percentage difference between projected and observed values. It provides a relative measure of accuracy, making it easier to compare data sets and scales. MAPE, when considering the relative magnitude of errors in terms of percentages, which are particularly relevant for forecasting and demand planning efforts, shows that it is advantageous.

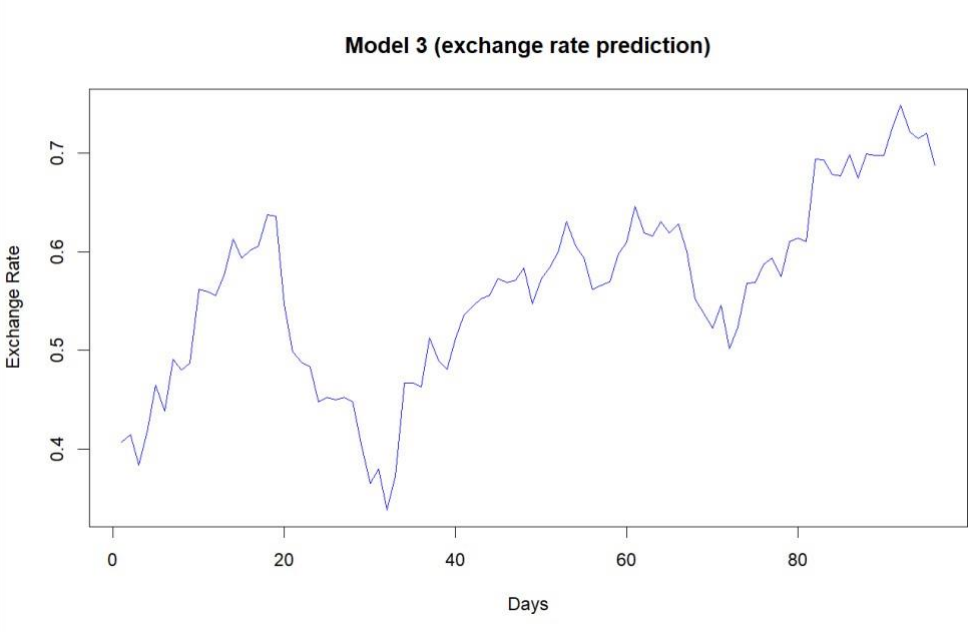
Symmetric Mean Absolute Percentage Error (SMAPE)

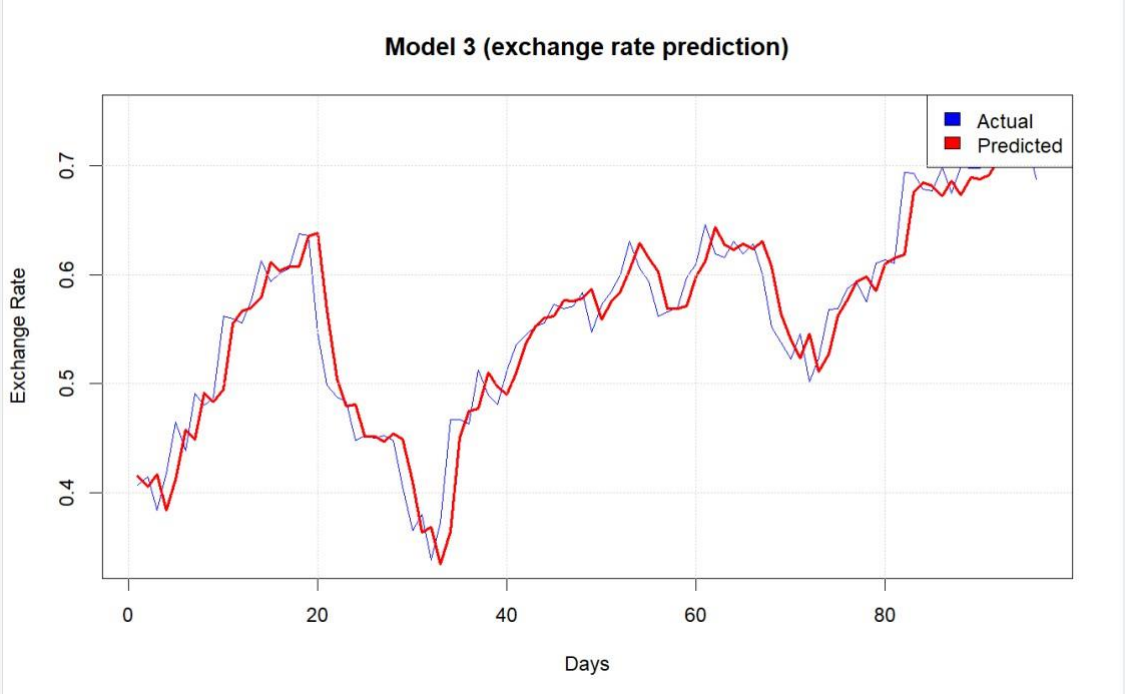
SMAPE, or symmetric Mean Absolute Percentage Error, presents itself as a refined version of MAPE, addressing various shortcomings such as handling zero values and improving interpretability. SMAPE mitigates this problem by including a small number in the denominator to avoid division of zero, unlike MAPE, which can produce undefined or infinite values when actual values are nil. SMAPE can be useful in cases where both overestimation and underestimation of data have equal weight, e.g. forecasting and inventory management, when a symmetric percentage-based error metric is needed.

The results of the best MLP network: Graphical and Statistical Analysis

The optimal MLP model is selected on the basis of the lowest RMSE value in this code block. Normalized predictions of the best model are returned to their original scale by the denormalise function. Subsequently, a line chart is utilised to visualise the predictions against the actual results for the "USD/EUR" exchange rate consumption values in the test set. The actual values are shown in blue, while the denorNormalisedmalized estimates of the best MLP model appear to be a red line. To distinguish between actual and forecast lines, a description is added to the chart. This code block effectively demonstrates the performance of the best MLP model in graphically forecasting the "USD/EUR" exchange rate consumption figures.

```r
# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T4$rate)
plot(x,exchangeRate_test_T4$rate, col = "blue", type = "l",main = "Model 3 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T4_3, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.

grid()
cleanoutput_T4_3<-cbind(exchangeRate_test_T4$rate,exchangeRate_pred_T4_3)
head(cleanoutput_T4_3)
```





Full code for part 1

```
# Load the required libraries
library(dplyr)        # For data manipulation
library(ggplot2)      # For data visualization
library(cluster)      # For clustering algorithms
library(readxl)       # For reading Excel files
library(NbClust)      # For cluster number determination
library(clusterCrit)  # For cluster validation
library(fpc)          # For gap statistic calculation
library(factoextra)   # For visualizing clustering results and more
library(reshape2)     # For reshaping data frames

#a
# Load the dataset
wine_data <- read_excel("Whitewine_v6.xlsx")

# Scaling the data
scaled_data <- scale(wine_data[,1:11])

# Convert scaled data to a data frame
scaled_data_df <- as.data.frame(scaled_data)

# Outlier detection and removal
# Using boxplot and filtering outliers
outliers_removed <- scaled_data_df %>%
  filter_all(all_vars(between(., quantile(., 0.25) - 1.5 * IQR(.), quantile(., 0.75) + 1.5 * IQR(.))))

# Plot the data before and after outlier removal
# Melt the data for easier plotting with ggplot2
melted_scaled_data <- melt(scaled_data_df)
melted_outliers_removed <- melt(outliers_removed)

# Create boxplots before outlier removal
ggplot(melted_scaled_data, aes(x = variable, y = value)) +
geom_boxplot() +
  ggtitle("Boxplot of Scaled Data Before Outlier Removal") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Create boxplots after outlier removal
ggplot(melted_outliers_removed, aes(x = variable, y = value)) +
geom_boxplot() +
  ggtitle("Boxplot of Scaled Data After Outlier Removal") +
theme(axis.text.x = element_text(angle = 90, hjust = 1))
# Justification for outlier removal:
# Outliers can significantly affect clustering results by skewing centroids and increasing intra-cluster variability.
# Removing outliers helps improve clustering quality by reducing noise and ensuring more representative clusters.

#b
# Determine the number of clusters using the NbClust method
```

```
set.seed(123)
clusterNo=NbClust(scaled_data,distance="euclidean",min.nc=2,max.nc=10,method="kmeans",index="all")

# Determine the number of clusters using the Elbow method
fviz_nbclust(scaled_data, kmeans, method = "wss")

# Determine the number of clusters using the Gap statistics method
fviz_nbclust(scaled_data, kmeans, method = 'gap_stat')

# Determine the number of clusters using the silhouette method
fviz_nbclust(scaled_data, kmeans, method = 'silhouette')


#c
# Apply k-means clustering with k=2
k <- 2
set.seed(123)
kmeans_output <- kmeans(scaled_data, centers = k)
kmeans_output

# Visualize clustering results
fviz_cluster(kmeans_output, data = scaled_data)

# Calculate BSS and WSS
BSS <- sum(kmeans_output$betweenss)
WSS <- kmeans_output$tot.withinss
TSS <- BSS + WSS
BSS_ratio <- BSS / TSS

# Output kmeans results
print(kmeans_output)
print(BSS_ratio)


#d
# Silhouette plot for visualizing clustering quality
silhouette_plot <- silhouette(kmeans_output$cluster, dist(scaled_data)) # Use scaled_data instead of
outliers_removed
plot(silhouette_plot)

# Calculate average silhouette width score
avg_silhouette <- mean(silhouette_plot[, 3])
print(avg_silhouette)
# Discussion on silhouette plot
# The silhouette plot illustrates individual point fitting within clusters and cluster distinctness. Higher average
silhouette width indicates better clustering, with values near 1 suggesting well-separated clusters.

#e
# Perform Principal Component Analysis (PCA)
pca_result <- prcomp(scaled_data,center= TRUE,scale = TRUE)
```

```
# Eigenvalues and eigenvectors
eigenvalues <- pca_result$sdev^2
eigenvectors <- pca_result$rotation

# Cumulative score per principal components
cumulative_score <- cumsum(pca_result$sdev^2 / sum(pca_result$sdev^2))

# Select principal components with cumulative score > 85%
selected_PCs <- which(cumulative_score > 0.85)[1]

# Transform the dataset using selected principal components
transformed_data <- as.data.frame(predict(pca_result, newdata = wine_data)[, 1:selected_PCs])

# Print the number of selected principal components
print(selected_PCs)


#f
# Determine the number of clusters using the NbClust method for PCA-transformed data
set.seed(123)
clusterNo_pca <- NbClust(transformed_data, distance = "euclidean", min.nc = 2, max.nc = 10, method =
"kmeans")

# Determine the number of clusters using the Elbow method for PCA-transformed data
fviz_nbclust(transformed_data, kmeans, method = "wss")

# Determine the number of clusters using the Gap statistics method for PCA-transformed data
fviz_nbclust(transformed_data, kmeans, method = 'gap_stat')

# Determine the number of clusters using the silhouette method for PCA-transformed data
fviz_nbclust(transformed_data, kmeans, method = 'silhouette')


#g
# Apply k-means clustering with k=3 on the PCA-transformed data
k <- 3
set.seed(123)
kmeans_model_pca <- kmeans(transformed_data, centers = k)
kmeans_model_pca

# Visualize clustering results
fviz_cluster(kmeans_model_pca, data = transformed_data)

# Calculate BSS, WSS, and TSS for PCA-based k-means
BSS_pca <- sum(kmeans_model_pca$betweenss)
WSS_pca <- kmeans_model_pca$tot.withinss
TSS_pca <- BSS_pca + WSS_pca
BSS_ratio_pca <- BSS_pca / TSS_pca
```

```
# Print k-means output
print(kmeans_model_pca)
print(BSS_ratio_pca)

#h
# Silhouette plot for PCA-based k-means
silhouette_plot_pca <- silhouette(kmeans_model_pca$cluster, dist(transformed_data))
plot(silhouette_plot_pca)

# Calculate average silhouette width score for PCA-based k-means
avg_silhouette_pca <- mean(silhouette_plot_pca[, 3])
print(avg_silhouette_pca)

#i
# Calculate the Calinski-Harabasz Index
calinski_harabasz_pca <- function(cluster_result, data) {
k <- length(unique(cluster_result$cluster))
  n <- nrow(data)
  BSS <- cluster_result$betweenss
  WSS <- cluster_result$tot.withinss
  ch_index <- ((n - k) / (k - 1)) * (BSS / WSS)
  return(ch_index)
}

ch_index_pca <- calinski_harabasz_pca(kmeans_model_pca, transformed_data)
ch_index_pca
```

Full code for part 2

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

#b
#read the data
exchangeData <- read_xlsx("ExchangeUSD.xlsx")
str(exchangeData)
summary(exchangeData)

# Focus on the 'USD/EUR' column
exchange_rate<-exchangeData$`USD/EUR`
summary(exchange_rate)

#Function to normalize data
normalize<-function(x){
  return((x-min(x))/(max(x)-min(x)))
}

# Function to renormalize data
unnormalize <- function(x, min, max) {
return( (max - min)*x + min )
}

#=====================================================#
#create time-lagged input variables up to (t-4) level
#=====================================================#
exchange_T4 <- bind_cols(
  input_1 = lag(exchange_rate, 1),
  input_2 = lag(exchange_rate, 2),
  input_3 = lag(exchange_rate, 3),
  input_4 = lag(exchange_rate, 4),
  rate=exchange_rate)
```

```
exchange_T4<-exchange_T4[complete.cases(exchange_T4),]
head(exchange_T4)

exchange_T4 <- na.omit(exchange_T4)
str(exchange_T4)
exchange_T4

# Normalize the data
exchangeRate_norm_T4<- as.data.frame(lapply(exchange_T4,normalize))
summary(exchangeRate_norm_T4)

#training and testing data
exchangeRate_train_T4<-exchangeRate_norm_T4[1:400,]
exchangeRate_test_T4<-exchangeRate_norm_T4[401:500,]
exchangeRate_test_T4<-exchangeRate_test_T4[complete.cases(exchangeRate_test_T4),]

#maximum & minimum value
exchangeRate_min_T4 <- min(exchangeRate_train_T4)

exchangeRate_max_T4 <- max(exchangeRate_train_T4)
formula_T4 <- as.formula(rate ~ input_1 + input_2 +input_3 + input_4 )

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 1 using 4 variables with one hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

exchange_model_T4_1<-neuralnet(formula_T4 ,hidden = 6 ,
                               data=exchangeRate_train_T4,linear.output=TRUE)
plot(exchange_model_T4_1)
model_results_T4_1<-predict(exchange_model_T4_1,exchangeRate_test_T4)

# Unnormalize the predicted values
exchangeRate_pred_T4_1 <- unnormalize(model_results_T4_1, exchangeRate_min_T4,
                                      exchangeRate_max_T4)
head(exchangeRate_pred_T4_1)
```

```
#Calculate performance metrics
rmse_value_T4_1 <- rmse(exchangeRate_test_T4$rate, exchangeRate_pred_T4_1)
mae_value_T4_1 <- mae(exchangeRate_test_T4$rate, exchangeRate_pred_T4_1)
mape_value_T4_1 <- mape(exchangeRate_test_T4$rate, exchangeRate_pred_T4_1)
smape_value_T4_1<-smape(exchangeRate_test_T4$rate,exchangeRate_pred_T4_1)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T4_1))
print(paste("MAE: ", mae_value_T4_1))
print(paste("MAPE: ", mape_value_T4_1))
print(paste("SMAPE: ", smape_value_T4_1))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T4$rate)
plot(x,exchangeRate_test_T4$rate, col = "blue", type = "l",main = "Model 1 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T4_1, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T4_1<-cbind(exchangeRate_test_T4$rate,exchangeRate_pred_T4_1)
head(cleanoutput_T4_1)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#
# Train the neural network model 2 using 4 variables with two hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#

exchange_model_T4_2<-neuralnet(formula_T4 ,hidden = c(5,3) ,
                    data=exchangeRate_train_T4,linear.output=TRUE)
plot(exchange_model_T4_2)
model_results_T4_2<-predict(exchange_model_T4_2,exchangeRate_test_T4)

# Unnormalize the predicted values
exchangeRate_pred_T4_2 <- unnormalize(model_results_T4_2, exchangeRate_min_T4,
                        exchangeRate_max_T4)
head(exchangeRate_pred_T4_2)
```

```
#Calculate performance metrics

rmse_value_T4_2 <- rmse(exchangeRate_test_T4$rate, exchangeRate_pred_T4_2)
mae_value_T4_2 <- mae(exchangeRate_test_T4$rate, exchangeRate_pred_T4_2)
mape_value_T4_2 <- mape(exchangeRate_test_T4$rate, exchangeRate_pred_T4_2)
smape_value_T4_2<-smape(exchangeRate_test_T4$rate,exchangeRate_pred_T4_2)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T4_2))
print(paste("MAE: ", mae_value_T4_2))
print(paste("MAPE: ", mape_value_T4_2))
print(paste("SMAPE: ", smape_value_T4_2))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T4$rate)
plot(x,exchangeRate_test_T4$rate, col = "blue", type = "l",main = "Model 2 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T4_2, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T4_2<-cbind(exchangeRate_test_T4$rate,exchangeRate_pred_T4_2)
head(cleanoutput_T4_2)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#
# Train the neural network model 3 using 4 variables with one hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#

exchange_model_T4_3<-neuralnet(formula_T4 ,hidden = 12 ,
                               data=exchangeRate_train_T4,linear.output=FALSE)
plot(exchange_model_T4_3)
model_results_T4_3<-predict(exchange_model_T4_3,exchangeRate_test_T4)

# Unnormalize the predicted values
exchangeRate_pred_T4_3 <- unnormalize(model_results_T4_3, exchangeRate_min_T4,
```

```
                                exchangeRate_max_T4)
head(exchangeRate_pred_T4_3)

#Calculate performance metrics
rmse_value_T4_3 <- rmse(exchangeRate_test_T4$rate, exchangeRate_pred_T4_3)
mae_value_T4_3 <- mae(exchangeRate_test_T4$rate, exchangeRate_pred_T4_3)
mape_value_T4_3 <- mape(exchangeRate_test_T4$rate, exchangeRate_pred_T4_3)
smape_value_T4_3<-smape(exchangeRate_test_T4$rate,exchangeRate_pred_T4_3)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T4_3))
print(paste("MAE: ", mae_value_T4_3))
print(paste("MAPE: ", mape_value_T4_3))
print(paste("SMAPE: ", smape_value_T4_3))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T4$rate)
plot(x,exchangeRate_test_T4$rate, col = "blue", type = "l",main = "Model 3 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T4_3, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))

grid()
cleanoutput_T4_3<-cbind(exchangeRate_test_T4$rate,exchangeRate_pred_T4_3)
head(cleanoutput_T4_3)



#=====================================================#
#create time-lagged input variables up to (t-3) level
#=====================================================#

exchange_T3 <- bind_cols(
  input_1 = lag(exchange_rate, 1),
  input_2 = lag(exchange_rate, 2),
  input_3 = lag(exchange_rate, 3),
  rate=exchange_rate)
exchange_T3<-exchange_T3[complete.cases(exchange_T3),]
```

```
head(exchange_T3)
exchange_T3 <- na.omit(exchange_T3)
str(exchange_T3)
exchange_T3

# Normalize the data
exchangeRate_norm_T3<- as.data.frame(lapply(exchange_T3,normalize))
summary(exchangeRate_norm_T3)

#training and testing data
exchangeRate_train_T3<-exchangeRate_norm_T3[1:400,]
exchangeRate_test_T3<-exchangeRate_norm_T3[401:500,]

exchangeRate_test_T3<-exchangeRate_test_T3[complete.cases(exchangeRate_test_T3),]

#maximum & minimum value
exchangeRate_min_T3 <- min(exchangeRate_train_T3)
exchangeRate_max_T3 <- max(exchangeRate_train_T3)

formula_T3 <- as.formula(rate ~ input_1 + input_2 +input_3 )

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#
# Train the neural network model 4 using 3 variables with one hidden layer
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#

exchange_model_T3_1<-neuralnet(formula_T3 ,hidden = 10 ,
                    data=exchangeRate_train_T3,linear.output=TRUE)
plot(exchange_model_T3_1)
model_results_T3_1<-predict(exchange_model_T3_1,exchangeRate_test_T3)

# Unnormalize the predicted values
exchangeRate_pred_T3_1 <- unnormalize(model_results_T3_1, exchangeRate_min_T3,
                         exchangeRate_max_T3)
head(exchangeRate_pred_T3_1)

#Calculate performance metrics
```

```
rmse_value_T3_1 <- rmse(exchangeRate_test_T3$rate, exchangeRate_pred_T3_1)
mae_value_T3_1 <- mae(exchangeRate_test_T3$rate, exchangeRate_pred_T3_1)
mape_value_T3_1 <- mape(exchangeRate_test_T3$rate, exchangeRate_pred_T3_1)
smape_value_T3_1<-smape(exchangeRate_test_T3$rate,exchangeRate_pred_T3_1)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T3_1))
print(paste("MAE: ", mae_value_T3_1))
print(paste("MAPE: ", mape_value_T3_1))
print(paste("SMAPE: ", smape_value_T3_1))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T3$rate)
plot(x,exchangeRate_test_T3$rate, col = "blue", type = "l",main = "Model 4 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T3_1, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T3_1<-cbind(exchangeRate_test_T3$rate,exchangeRate_pred_T3_1)
head(cleanoutput_T3_1)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 5 using 3 variables with two hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

exchange_model_T3_2<-neuralnet(formula_T3 ,hidden = c(5,6) ,
                               data=exchangeRate_train_T3,linear.output=TRUE)
plot(exchange_model_T3_2)
model_results_T3_2<-predict(exchange_model_T3_2,exchangeRate_test_T3)

# Unnormalize the predicted values
exchangeRate_pred_T3_2 <- unnormalize(model_results_T3_2, exchangeRate_min_T3,
                                      exchangeRate_max_T3)
head(exchangeRate_pred_T3_2)
```

```
#Calculate performance metrics
rmse_value_T3_2 <- rmse(exchangeRate_test_T3$rate, exchangeRate_pred_T3_2)
mae_value_T3_2 <- mae(exchangeRate_test_T3$rate, exchangeRate_pred_T3_2)
mape_value_T3_2 <- mape(exchangeRate_test_T3$rate, exchangeRate_pred_T3_2)
smape_value_T3_2<-smape(exchangeRate_test_T3$rate,exchangeRate_pred_T3_2)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T3_2))
print(paste("MAE: ", mae_value_T3_2))
print(paste("MAPE: ", mape_value_T3_2))
print(paste("SMAPE: ", smape_value_T3_2))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T3$rate)
plot(x,exchangeRate_test_T3$rate, col = "blue", type = "l",main = "Model 5 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T3_2, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T3_2<-cbind(exchangeRate_test_T3$rate,exchangeRate_pred_T3_2)
head(cleanoutput_T3_2)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#
# Train the neural network model 6 using 3 variables with Three hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#

exchange_model_T3_3<-neuralnet(formula_T3 ,hidden = c(4,5,4) ,
                               data=exchangeRate_train_T3,linear.output=FALSE)
plot(exchange_model_T3_3)
model_results_T3_3<-predict(exchange_model_T3_3,exchangeRate_test_T3)

# Unnormalize the predicted values
exchangeRate_pred_T3_3 <- unnormalize(model_results_T3_3, exchangeRate_min_T3,
                                      exchangeRate_max_T3)
head(exchangeRate_pred_T3_3)
```

```
#Calculate performance metrics
rmse_value_T3_3 <- rmse(exchangeRate_test_T3$rate, exchangeRate_pred_T3_3)
mae_value_T3_3 <- mae(exchangeRate_test_T3$rate, exchangeRate_pred_T3_3)
mape_value_T3_3 <- mape(exchangeRate_test_T3$rate, exchangeRate_pred_T3_3)
smape_value_T3_3<-smape(exchangeRate_test_T3$rate,exchangeRate_pred_T3_3)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T3_3))
print(paste("MAE: ", mae_value_T3_3))
print(paste("MAPE: ", mape_value_T3_3))
print(paste("SMAPE: ", smape_value_T3_3))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T3$rate)
plot(x,exchangeRate_test_T3$rate, col = "blue", type = "l",main = "Model 6 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T3_3, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T3_3<-cbind(exchangeRate_test_T3$rate,exchangeRate_pred_T3_3)
head(cleanoutput_T3_3)

#=================================================#
#create time-lagged input variables up to (t-2) level
#=================================================#
exchange_T2 <- bind_cols(
  input_1 = lag(exchange_rate, 1),
  input_2 = lag(exchange_rate, 2),
  rate=exchange_rate)
exchange_T2<-exchange_T2[complete.cases(exchange_T2),]
head(exchange_T2)
exchange_T2 <- na.omit(exchange_T2)
str(exchange_T2)
exchange_T2
```

```
# Normalize the data
exchangeRate_norm_T2<- as.data.frame(lapply(exchange_T2,normalize))
summary(exchangeRate_norm_T2)

#training and testing data
exchangeRate_train_T2<-exchangeRate_norm_T2[1:400,]
exchangeRate_test_T2<-exchangeRate_norm_T2[401:500,]
exchangeRate_test_T2<-exchangeRate_test_T2[complete.cases(exchangeRate_test_T2),]

#maximum & minimum value
exchangeRate_min_T2 <- min(exchangeRate_train_T2)
exchangeRate_max_T2 <- max(exchangeRate_train_T2)

formula_T2 <- as.formula(rate ~ input_1 + input_2 )

#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#
# Train the neural network model 7 using 2 variables with one hidden layer
#+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#

exchange_model_T2_1<-neuralnet(formula_T2 ,hidden = 15 ,
                         data=exchangeRate_train_T2,linear.output=FALSE)
plot(exchange_model_T2_1)
model_results_T2_1<-predict(exchange_model_T2_1,exchangeRate_test_T2)

# Unnormalize the predicted values
exchangeRate_pred_T2_1 <- unnormalize(model_results_T2_1, exchangeRate_min_T2,
                              exchangeRate_max_T2)
head(exchangeRate_pred_T2_1)

#Calculate performance metrics
rmse_value_T2_1 <- rmse(exchangeRate_test_T2$rate, exchangeRate_pred_T2_1)
mae_value_T2_1 <- mae(exchangeRate_test_T2$rate, exchangeRate_pred_T2_1)
mape_value_T2_1 <- mape(exchangeRate_test_T2$rate, exchangeRate_pred_T2_1)
smape_value_T2_1<-smape(exchangeRate_test_T2$rate,exchangeRate_pred_T2_1)

# Print performance metrics
```

```
print(paste("RMSE: ", rmse_value_T2_1))
print(paste("MAE: ", mae_value_T2_1))
print(paste("MAPE: ", mape_value_T2_1))
print(paste("SMAPE: ", smape_value_T2_1))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T2$rate)
plot(x,exchangeRate_test_T2$rate, col = "blue", type = "l",main = "Model 7 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T2_1, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T2_1<-cbind(exchangeRate_test_T2$rate,exchangeRate_pred_T2_1)
head(cleanoutput_T2_1)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#
# Train the neural network model 8 using 2 variables with one hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#

exchange_model_T2_2<-neuralnet(formula_T2 ,hidden = 10 ,
                               data=exchangeRate_train_T2,linear.output=TRUE)
plot(exchange_model_T2_2)
model_results_T2_2<-predict(exchange_model_T2_2,exchangeRate_test_T2)

# Unnormalize the predicted values
exchangeRate_pred_T2_2 <- unnormalize(model_results_T2_2, exchangeRate_min_T2,
                                      exchangeRate_max_T2)
head(exchangeRate_pred_T2_2)

#Calculate performance metrics
rmse_value_T2_2 <- rmse(exchangeRate_test_T2$rate, exchangeRate_pred_T2_2)
mae_value_T2_2 <- mae(exchangeRate_test_T2$rate, exchangeRate_pred_T2_2)
mape_value_T2_2 <- mape(exchangeRate_test_T2$rate, exchangeRate_pred_T2_2)
smape_value_T2_2<-smape(exchangeRate_test_T2$rate,exchangeRate_pred_T2_2)
```

```
# Print performance metrics
print(paste("RMSE: ", rmse_value_T2_2))
print(paste("MAE: ", mae_value_T2_2))
print(paste("MAPE: ", mape_value_T2_2))
print(paste("SMAPE: ", smape_value_T2_2))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T2$rate)
plot(x,exchangeRate_test_T2$rate, col = "blue", type = "l",main = "Model 8 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T2_2, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))

grid()
cleanoutput_T2_2<-cbind(exchangeRate_test_T2$rate,exchangeRate_pred_T2_2)
head(cleanoutput_T2_2)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#
# Train the neural network model 9 using 2 variables with one hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++#

exchange_model_T2_3<-neuralnet(formula_T2 ,hidden = c(5,2) ,
                        data=exchangeRate_train_T2,linear.output=FALSE)
plot(exchange_model_T2_3)
model_results_T2_3<-predict(exchange_model_T2_3,exchangeRate_test_T2)

# Unnormalize the predicted values
exchangeRate_pred_T2_3 <- unnormalize(model_results_T2_3, exchangeRate_min_T2,
                              exchangeRate_max_T2)
head(exchangeRate_pred_T2_3)

#Calculate performance metrics
rmse_value_T2_3 <- rmse(exchangeRate_test_T2$rate, exchangeRate_pred_T2_3)
mae_value_T2_3 <- mae(exchangeRate_test_T2$rate, exchangeRate_pred_T2_3)
mape_value_T2_3 <- mape(exchangeRate_test_T2$rate, exchangeRate_pred_T2_3)
smape_value_T2_3<-smape(exchangeRate_test_T2$rate,exchangeRate_pred_T2_3)
```

```
# Print performance metrics
print(paste("RMSE: ", rmse_value_T2_3))
print(paste("MAE: ", mae_value_T2_3))
print(paste("MAPE: ", mape_value_T2_3))
print(paste("SMAPE: ", smape_value_T2_3))

#Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T2$rate)
plot(x,exchangeRate_test_T2$rate, col = "blue", type = "l",main = "Model 9 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T2_3, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T2_3<-cbind(exchangeRate_test_T2$rate,exchangeRate_pred_T2_3)
head(cleanoutput_T2_3)

#=====================================================#
#create time-lagged input variables up to (t-1) level
#=====================================================#
exchange_T1 <- bind_cols(
  input_1 = lag(exchange_rate, 1),
  rate=exchange_rate)

exchange_T1<-exchange_T1[complete.cases(exchange_T1),]
head(exchange_T1)
exchange_T1 <- na.omit(exchange_T1)

str(exchange_T1)
exchange_T1

# Normalize the data
exchangeRate_norm_T1<- as.data.frame(lapply(exchange_T1,normalize))
summary(exchangeRate_norm_T1)

#training and testing data
```

```
exchangeRate_train_T1<-exchangeRate_norm_T1[1:400,]
exchangeRate_test_T1<-exchangeRate_norm_T1[401:500,]
exchangeRate_test_T1<-exchangeRate_test_T1[complete.cases(exchangeRate_test_T1),]

#maximum & minimum value
exchangeRate_min_T1 <- min(exchangeRate_train_T1)
exchangeRate_max_T1 <- max(exchangeRate_train_T1)
formula_T1 <- as.formula(rate ~ input_1 )

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 10 using 1 variables with one hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

exchange_model_T1_1<-neuralnet(formula_T1 ,hidden = 12 ,
                               data=exchangeRate_train_T1,linear.output=TRUE)
plot(exchange_model_T1_1)
model_results_T1_1<-predict(exchange_model_T1_1,exchangeRate_test_T1)

# Unnormalize the predicted values

exchangeRate_pred_T1_1 <- unnormalize(model_results_T1_1, exchangeRate_min_T1,
                                      exchangeRate_max_T1)
head(exchangeRate_pred_T1_1)

#Calculate performance metrics
rmse_value_T1_1 <- rmse(exchangeRate_test_T1$rate, exchangeRate_pred_T1_1)
mae_value_T1_1 <- mae(exchangeRate_test_T1$rate, exchangeRate_pred_T1_1)
mape_value_T1_1 <- mape(exchangeRate_test_T1$rate, exchangeRate_pred_T1_1)
smape_value_T1_1<-smape(exchangeRate_test_T1$rate,exchangeRate_pred_T1_1)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T1_1))
print(paste("MAE: ", mae_value_T1_1))
print(paste("MAPE: ", mape_value_T1_1))
print(paste("SMAPE: ", smape_value_T1_1))
```

```
# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T1$rate)
plot(x,exchangeRate_test_T1$rate, col = "blue", type = "l",main = "Model 10 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T1_1, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T1_1<-cbind(exchangeRate_test_T1$rate,exchangeRate_pred_T1_1)
head(cleanoutput_T1_1)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 11 using 1 variables with two hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#

exchange_model_T1_2<-neuralnet(formula_T1 ,hidden = c(5,4) ,
                    data=exchangeRate_train_T1,linear.output=TRUE)
plot(exchange_model_T1_2)
model_results_T1_2<-predict(exchange_model_T1_2,exchangeRate_test_T1)

# Unnormalize the predicted values
exchangeRate_pred_T1_2 <- unnormalize(model_results_T1_2, exchangeRate_min_T1,
                        exchangeRate_max_T1)
head(exchangeRate_pred_T1_2)

#Calculate performance metrics
rmse_value_T1_2 <- rmse(exchangeRate_test_T1$rate, exchangeRate_pred_T1_2)
mae_value_T1_2 <- mae(exchangeRate_test_T1$rate, exchangeRate_pred_T1_2)
mape_value_T1_2 <- mape(exchangeRate_test_T1$rate, exchangeRate_pred_T1_2)
smape_value_T1_2<-smape(exchangeRate_test_T1$rate,exchangeRate_pred_T1_2)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T1_2))
print(paste("MAE: ", mae_value_T1_2))
print(paste("MAPE: ", mape_value_T1_2))
print(paste("SMAPE: ", smape_value_T1_2))
```

```
# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T1$rate)
plot(x,exchangeRate_test_T1$rate, col = "blue", type = "l",main = "Model 11 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T1_2, col = "red",lwd=2)

legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T1_2<-cbind(exchangeRate_test_T1$rate,exchangeRate_pred_T1_2)
head(cleanoutput_T1_2)

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
# Train the neural network model 12 using 1 variables with one hidden layer
#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++#
exchange_model_T1_3<-neuralnet(formula_T1 ,hidden = 5,
                               data=exchangeRate_train_T1,linear.output=FALSE)
plot(exchange_model_T1_3)
model_results_T1_3<-predict(exchange_model_T1_3,exchangeRate_test_T1)

# Unnormalize the predicted values
exchangeRate_pred_T1_3 <- unnormalize(model_results_T1_3, exchangeRate_min_T1,
exchangeRate_max_T1)
head(exchangeRate_pred_T1_3)

#Calculate performance metrics
rmse_value_T1_3 <- rmse(exchangeRate_test_T1$rate, exchangeRate_pred_T1_3)
mae_value_T1_3 <- mae(exchangeRate_test_T1$rate, exchangeRate_pred_T1_3)
mape_value_T1_3 <- mape(exchangeRate_test_T1$rate, exchangeRate_pred_T1_3)
smape_value_T1_3<-smape(exchangeRate_test_T1$rate,exchangeRate_pred_T1_3)

# Print performance metrics
print(paste("RMSE: ", rmse_value_T1_3))
print(paste("MAE: ", mae_value_T1_3))
print(paste("MAPE: ", mape_value_T1_3))
```

```
print(paste("SMAPE: ", smape_value_T1_3))

# Plotting actual vs predicted exchange rates
x=1:length(exchangeRate_test_T1$rate)
plot(x,exchangeRate_test_T1$rate, col = "blue", type = "l",main = "Model 12 (exchange rate
prediction)",xlab = "Days", ylab = "Exchange Rate")
lines(x,exchangeRate_pred_T1_3, col = "red",lwd=2)
legend("topright", legend = c("Actual", "Predicted"), fill = c("blue", "red"), col=2:3,adj = c(0,0.6))
grid()

cleanoutput_T1_3<-cbind(exchangeRate_test_T1$rate,exchangeRate_pred_T1_3)
head(cleanoutput_T1_3)

rmse_all_model<-c(rmse_value_T4_1,rmse_value_T4_2,rmse_value_T4_3,
                  rmse_value_T3_1,rmse_value_T3_2,rmse_value_T3_3,
                  rmse_value_T2_1,rmse_value_T2_2,rmse_value_T2_3,
                  rmse_value_T1_1,rmse_value_T1_2,rmse_value_T1_3)

mae_all_model<-c(mae_value_T4_1, mae_value_T4_2,mae_value_T4_3,
                 mae_value_T3_1,mae_value_T3_2,mae_value_T3_3,
                 mae_value_T2_1,mae_value_T2_2,mae_value_T2_3,
                 mae_value_T1_1,mae_value_T1_2,mae_value_T1_3)

mape_all_model<-c(mape_value_T4_1,mape_value_T4_2,mape_value_T4_3,
                  mape_value_T3_1,mape_value_T3_2,mape_value_T3_3,
                  mape_value_T2_1,mape_value_T2_2,mape_value_T2_3,
                  mape_value_T1_1,mape_value_T1_2,mape_value_T1_3)

smape_all_model<-c(smape_value_T4_1,smape_value_T4_2,smape_value_T4_3,
                   smape_value_T3_1,smape_value_T3_2,smape_value_T3_3,
                   smape_value_T2_1,smape_value_T2_2,smape_value_T2_3,
                   smape_value_T1_1,smape_value_T1_2,smape_value_T1_3)

model_comparison =
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_model,
           MAE= mae_all_model,
```

```
                    MAPE=mape_all_model,
                    SMAAPE=smape_all_model)
model_comparison
```