5DATA002W.2 Machine Learning and Data Mining Referal Coursework

Andrew Hanna
W1816963

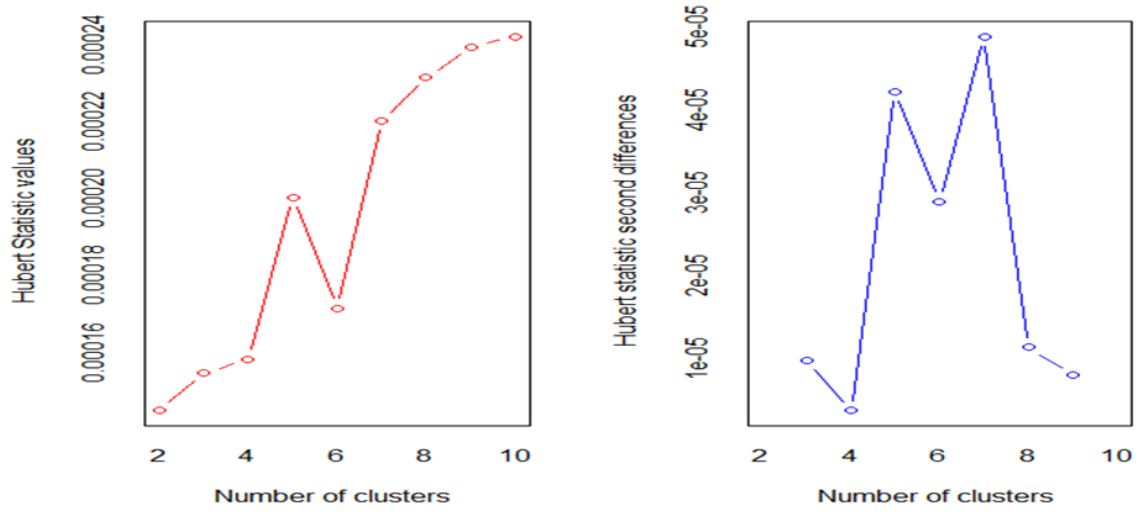
Partitioning Clusering Part

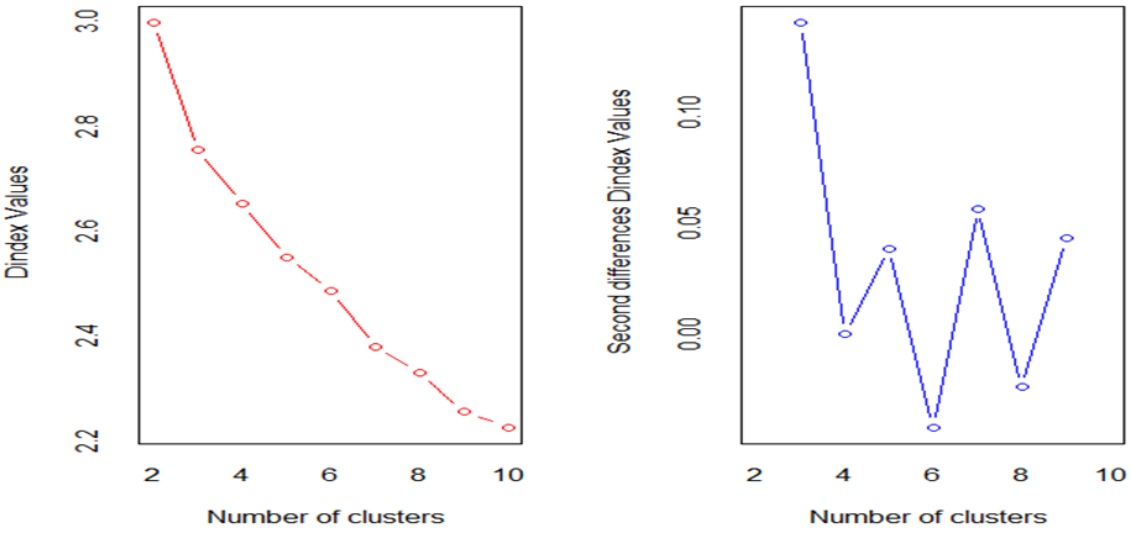
A)

Scaling refers to the process of transforming the variables in your dataset to a specific range.
Removing outliers is the process of removing outliers which are data points that deviate significantly from the majority of the data. Now the importance of the order.
Scaling before removing outliers: Outliers can have an impact on the scaling process if you scale the variables first and then get rid of them. By definition, outliers have extreme values that can make the scale of the variables stretch. Outliers can have a greater impact on the scaling process if they are removed before the variables are scaled. A biased scaling transformation that may not accurately represent the majority of the data can result from this.
Prior to scaling, remove outliers: However, if outliers are removed prior to scaling, extreme values that could disproportionately affect scaling can be eliminated. This can ensure that the scaling transformation is based on the majority of the data by first removing outliers, which makes it possible to scale the variables in a more representative manner.

B)

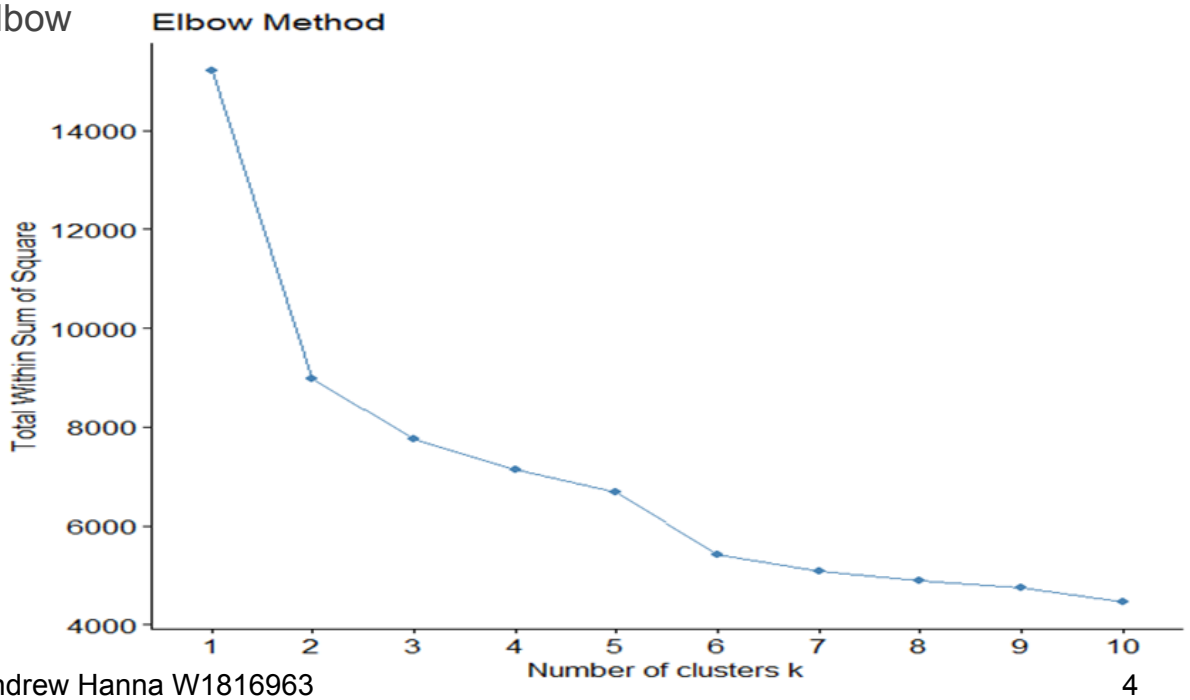
NBClust



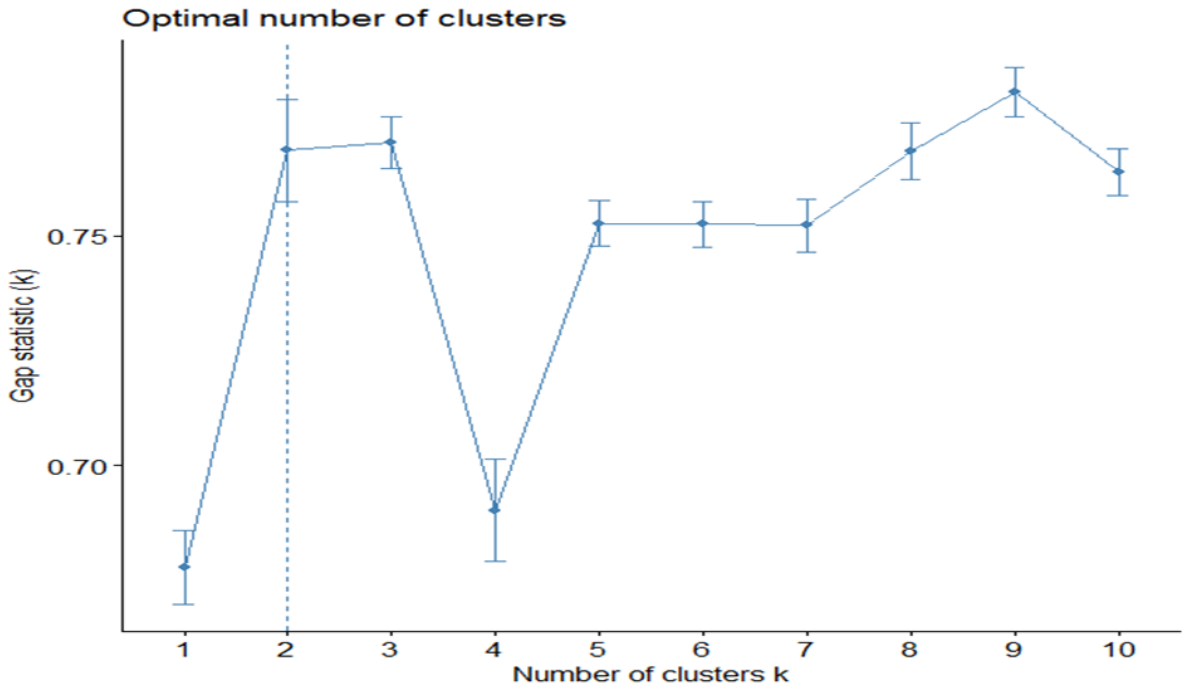



```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

*******************************************************************
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 2 proposed 3 as the best number of clusters
* 4 proposed 5 as the best number of clusters
* 2 proposed 6 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 2 proposed 9 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                  ***** Conclusion *****

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

Elbow

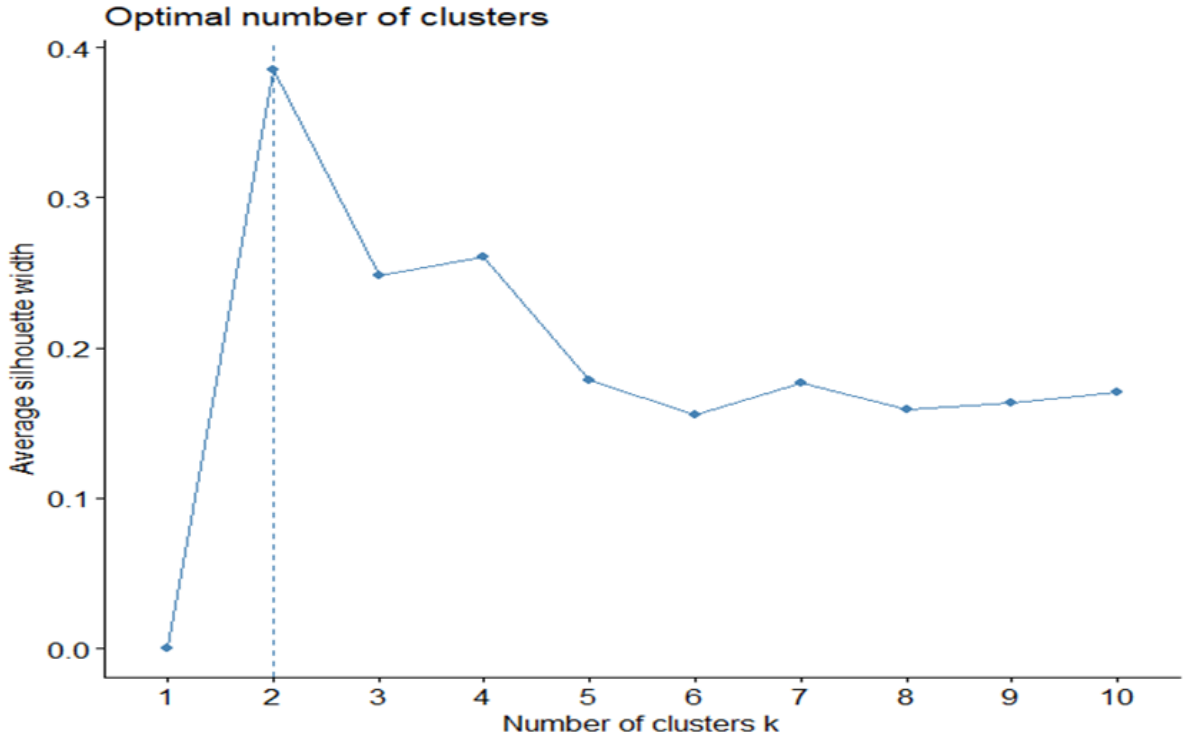


Gap Statistics



	logW	E.logW	gap	SE.sim
[1,]	7.062222	7.739346	0.6771248	0.007268825
[2,]	6.800486	7.569439	0.7689534	0.009405802
[3,]	6.717749	7.488783	0.7710337	0.005341158
[4,]	6.674103	7.423463	0.7493603	0.011843621
[5,]	6.636768	7.387118	0.7503494	0.005315809
[6,]	6.586387	7.357218	0.7708314	0.006015138
[7,]	6.554103	7.330152	0.7760492	0.006356966
[8,]	6.532066	7.305366	0.7732999	0.005587188
[9,]	6.508378	7.283298	0.7749201	0.006735150
[10,]	6.494766	7.265434	0.7706683	0.005132846

Silhouette



	NumClusters	SilhouetteScore
1	2	1.128290
2	3	1.293951
3	4	1.642597
4	5	1.754669
5	6	2.454098
6	7	2.880193
7	8	3.062219
8	9	3.601947
9	10	3.719492

Result Analysis

NBClust there are different opinions between the indices, however the majority rule suggests that the best number of clusters for this data is 2.

The Elbow method shows a graph where we can see that as the number of clusters increases the sum of squares deccreases. However the biggest drop is from k=2 which suggests that the best number of clusters is 2.

The Gap Statistics method calcuclates the gap values for clustering k=1 to k=10. The table indicates that k=2 might be a good number of clusters and the graph also suggests this.

The Silhoutte method shows that the best value of k is 2 as it shows the highest average silhouette width as .38.

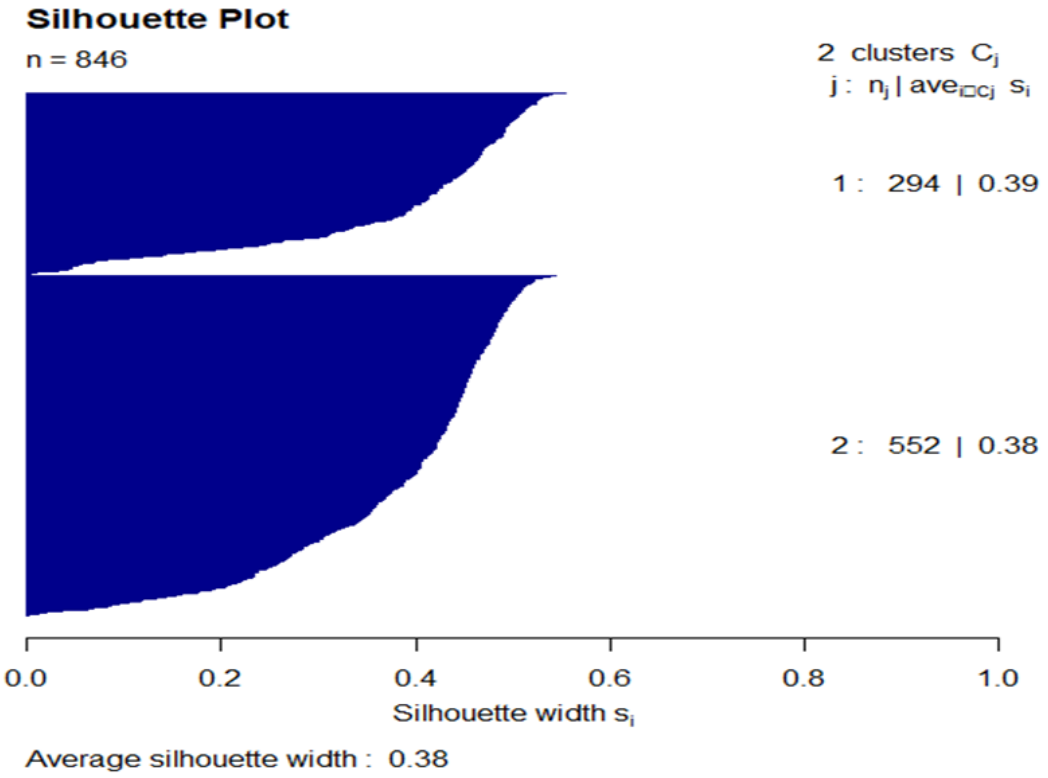
Based on all of these results from the four automated tools there is a common correlation between all of the method that 2 is the ideal number of clusters, so i ccan choose k=2 as the ideal number of clusters for this data.

C)

```
[1] "The value of BSS is: 6232.1719"
> print(paste("The value of WSS is:", round(wss, 4)))
[1] "The value of WSS is: 8977.8281"
>
> bss_ratio <- bss / tss
> cat("BSS/TSS Ratio:", bss_ratio, "\n")
BSS/TSS Ratio: 0.4097417
```

BSS value is 5232.17
WSS value is 8977.83
BSS/TSS value is .4097

D)



The average silhouette width is a measure of the quality or coherence of the obtained clusters. It quantifies how well each data point in a cluster is separated from data points in other clusters.

A value close to 1 indicates that the data points are well-clustered and properly separated from other clusters, indicating high quality clusters. Conversely, a value close to -1 suggests that data points might have been assigned to the wrong clusters, indicating poor quality clusters.

The overall average silhouette width for both clusters is calculated to be 0.3849. Based on these results, the clusters seem to have moderate coherence, with average silhouette widths around 0.38-0.39. It implies that there is room for improvement in the clustering results to achieve higher-quality clusters with better separation and cohesion.

E)

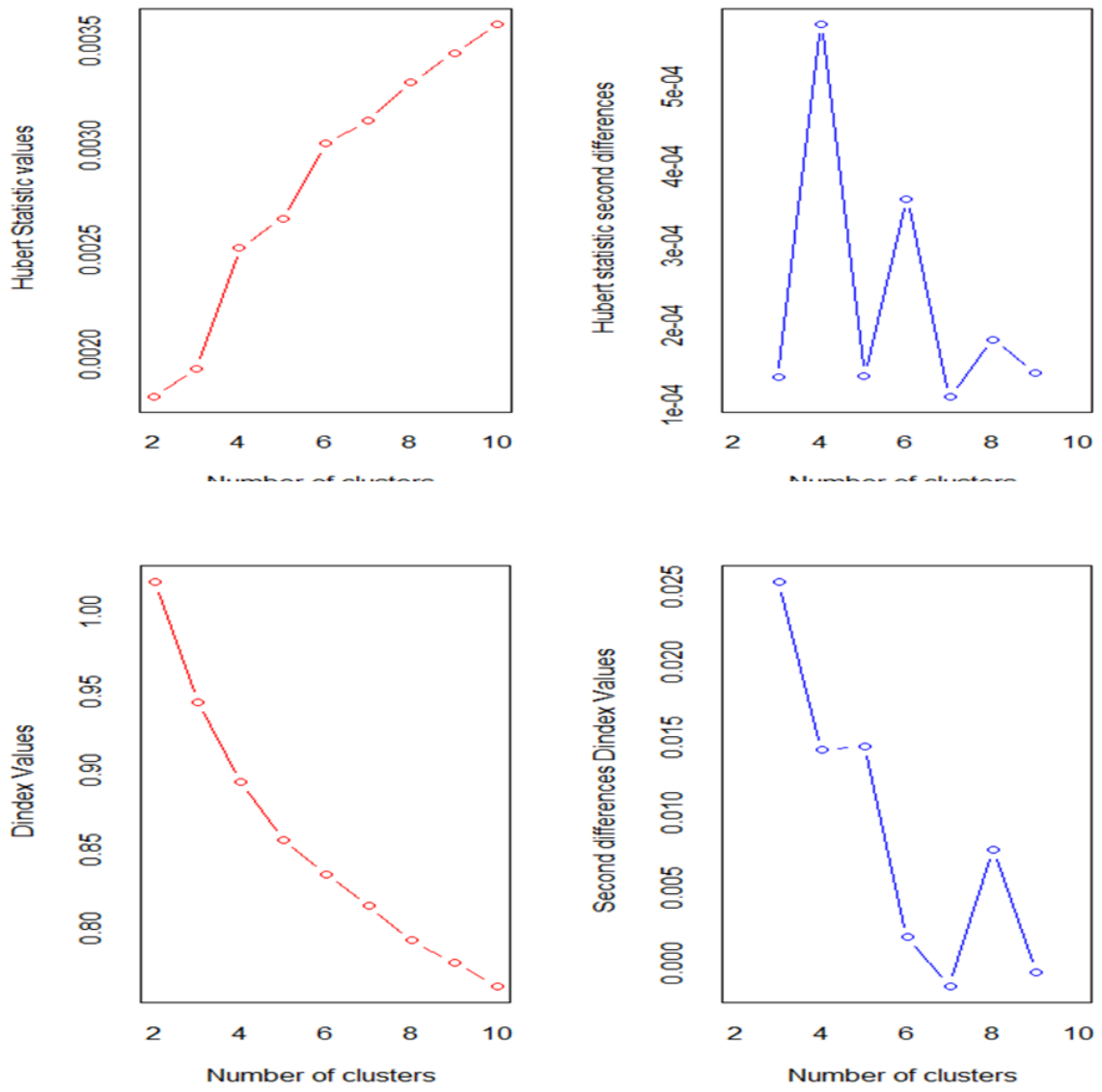
```
> cumulative_score
 [1]  52.14427  64.26836  74.79476  81.33854  86.82416  91.87100  94.57887  96.55359  97.77836  98.62364
[11]  99.12797  99.47491  99.66945  99.80124  99.89128  99.96354  99.99792 100.00000
> selected_pcs <- which(cumulative_score > 92)
```

The cumulative score represents the amount of variance explained by each PC and provides insights into how much information is retained as we include more PCs. By setting a cumulative score threshold of 92%, we aim to select a sufficient number of PCs that capture a significant portion of the dataset's overall variability.

Choosing a cumulative score threshold of 92% ensures that we retain a large proportion of the original dataset's information while reducing its dimensionality. This reduction in dimensionality helps in simplifying the dataset and potentially removing noise or irrelevant features.

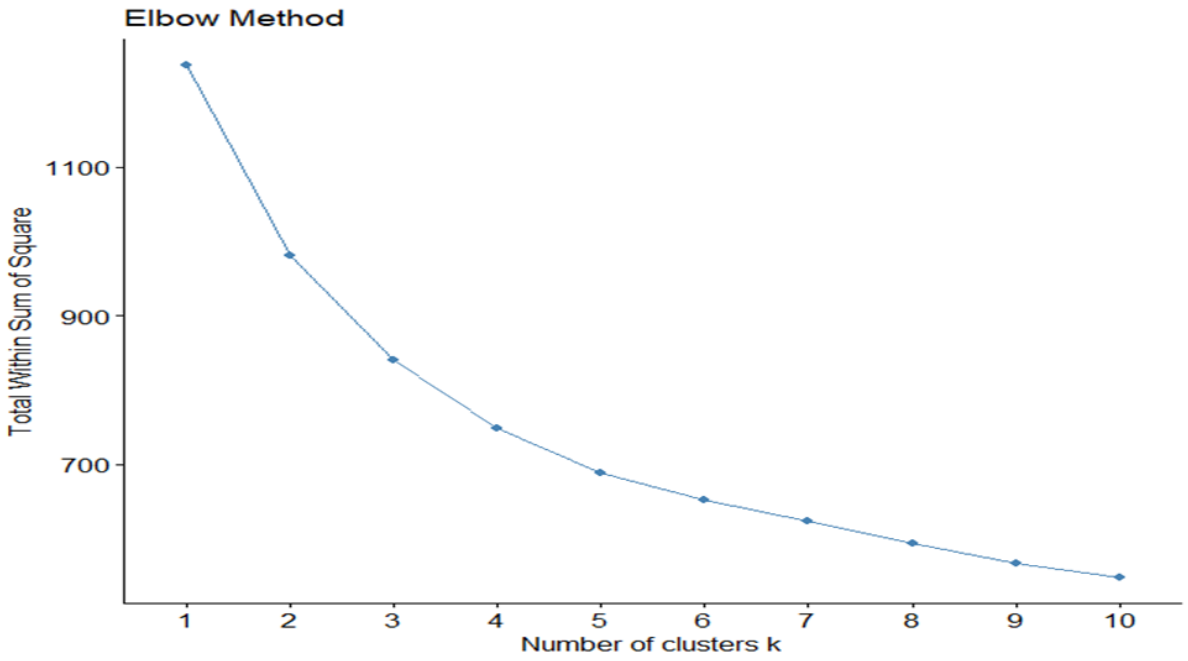
F)

NBClust

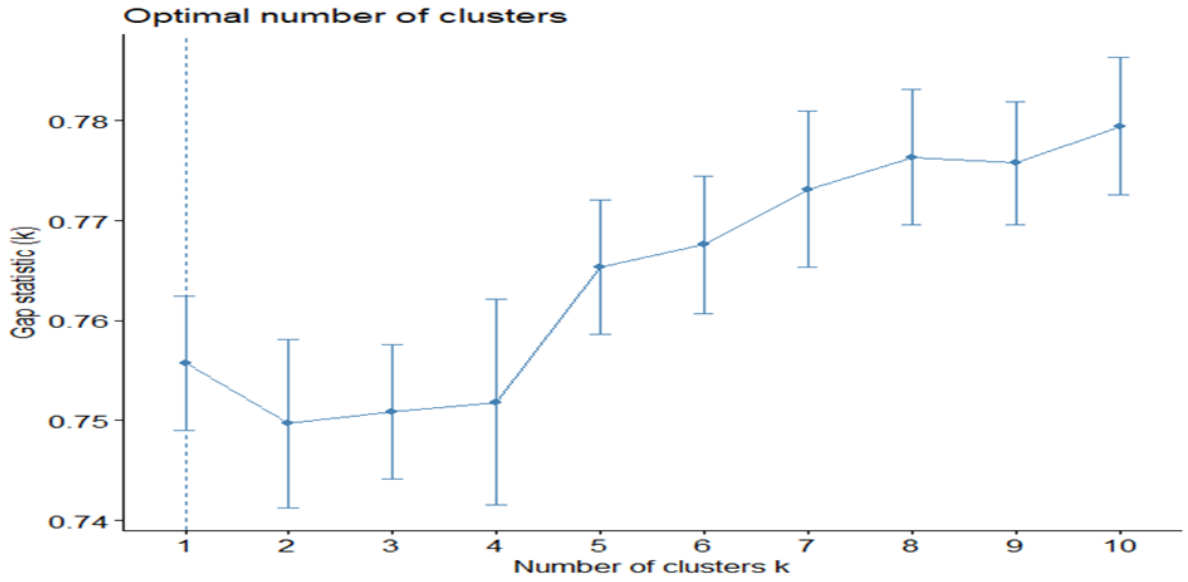


```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

*******************************************************************
* Among all indices:
* 7 proposed 2 as the best number of clusters
* 6 proposed 3 as the best number of clusters
* 3 proposed 4 as the best number of clusters
* 2 proposed 5 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 2 proposed 8 as the best number of clusters
* 2 proposed 10 as the best number of clusters

                    ***** Conclusion *****

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


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

Elbow

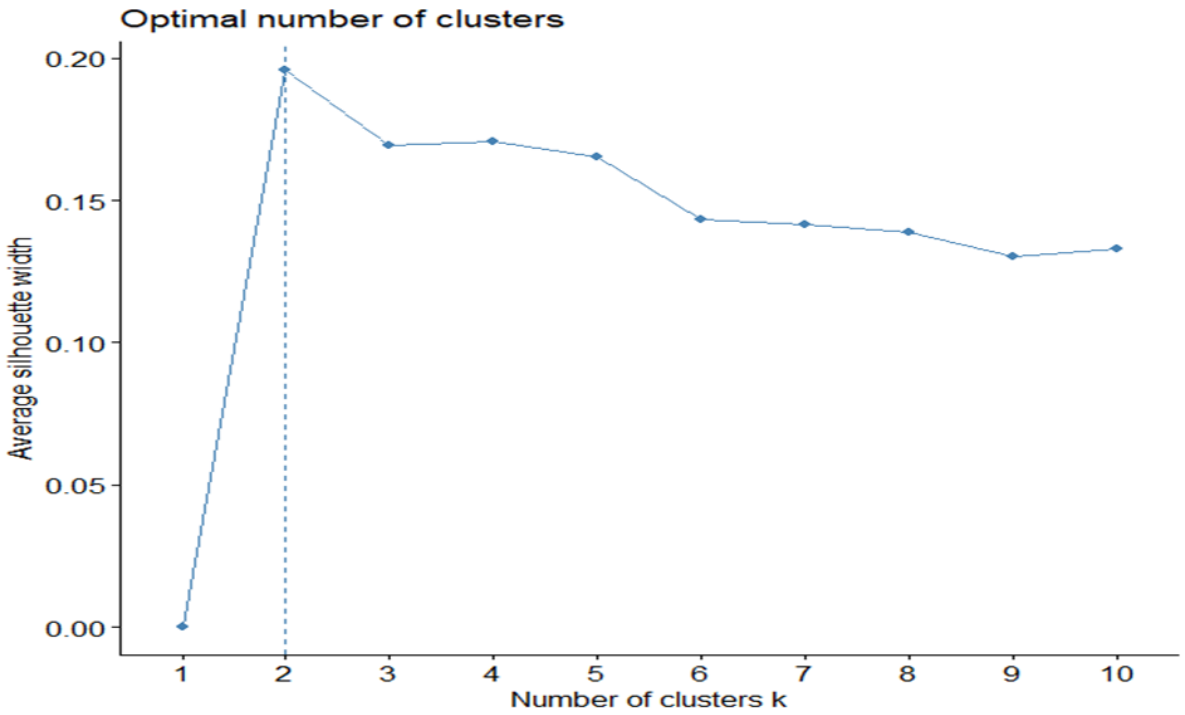


GAP Statistics



	logW	E.logW	gap	SE.sim
[1,]	5.832453	6.588213	0.7557595	0.006732861
[2,]	5.718447	6.468137	0.7496900	0.008444667
[3,]	5.640619	6.391538	0.7509185	0.006728172
[4,]	5.584842	6.336679	0.7518370	0.010288343
[5,]	5.542882	6.308218	0.7653358	0.006722407
[6,]	5.515349	6.282940	0.7675903	0.006882867
[7,]	5.487647	6.260788	0.7731418	0.007793553
[8,]	5.463220	6.239570	0.7763500	0.006782638
[9,]	5.444589	6.220360	0.7757709	0.006168309
[10,]	5.424212	6.203662	0.7794500	0.006863052

Silhouette



Analysis of Results

In NBClust there are different opinions amongst the indices but the majority rule suggests the best number of clusters is 2.

Elbow method graph shows that as the number of clusters increases the within sum of squares decreases, however we can see that k=4 is most suitable. This suggests that the best number of clusters is 4

GAP Statistics calculates the gap values for clustering k=1 to k=10. Based on the table it looks like k=1 would be the best option.

Silhouette method shows that the best value for k is 2.

Based on the results from the different automated methods the best number of clusters would either be 2 or 4, however the majority is 2 so the best k would be 2.

G)

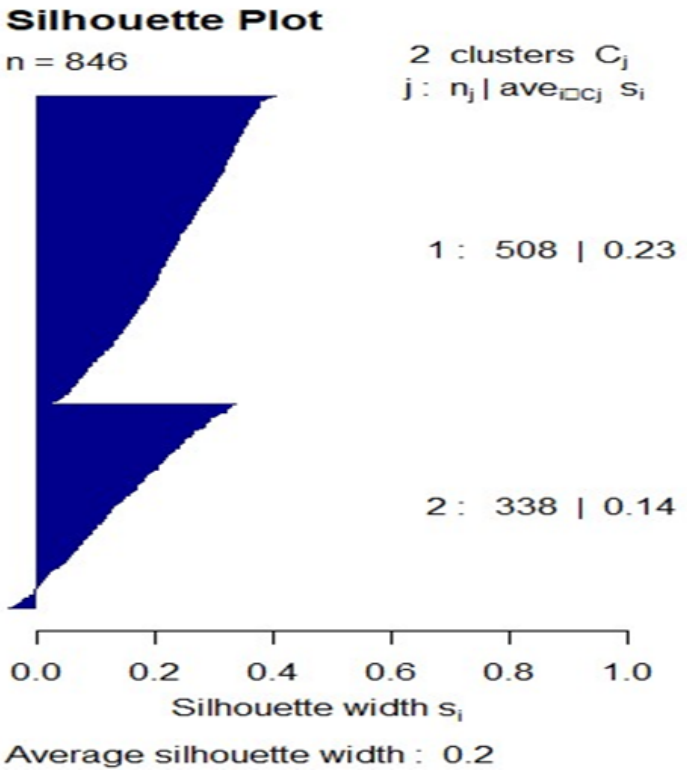
```
> print(paste("The value of BSS is:", round(bss, 4)))
[1] "The value of BSS is: 255.7847"
> print(paste("The value of WSS is:", round(wss, 4)))
[1] "The value of WSS is: 980.6366"
>
> bss_ratio <- bss / tss
> cat("BSS/TSS Ratio:", bss_ratio, "\n")
BSS/TSS Ratio: 0.206875
```

The BSS/TSS ratio represents the proportion of the total sum of squares (TSS) that is explained by the between-cluster sum of squares (BSS). In this case, the BSS/TSS ratio is approximately 0.2069, indicating that around 20.69% of the total variance in the dataset is accounted for by the clustering. A higher BSS/TSS ratio suggests better separation and distinctiveness among the clusters.

The BSS (255.7847) represents the sum of the squared distances between the cluster centers and the overall mean of the transformed dataset. It quantifies the separation between the clusters.

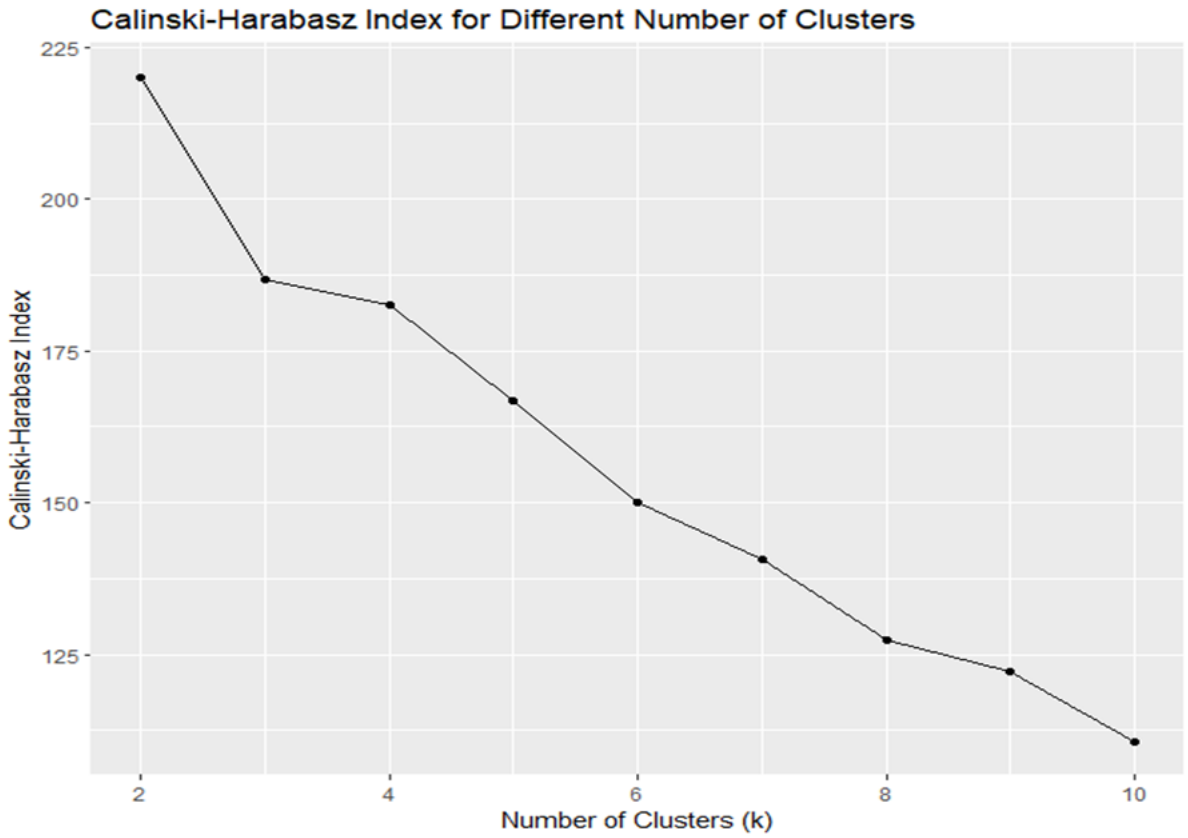
The WSS (980.6366) represents the sum of the squared distances between the data points and their respective cluster centers. It measures the compactness or cohesion within each cluster.

H)



Based on the average silhouette width of 0.1957, it indicates a moderate level of cluster quality. The average silhouette width ranges from -1 to 1, with values closer to 1 indicating well-separated clusters. A value of 0.1957 suggests that, on average, the data points are relatively close to the neighboring clusters and not strongly associated with their own cluster.

l)



From the plot, we can observe that as the number of clusters (k) increases, the Calinski-Harabasz Index generally decreases. This suggests that a larger number of clusters leads to a higher within-cluster dispersion relative to between-cluster dispersion. The highest Calinski-Harabasz Index value is observed at k = 2, around 223. This indicates that dividing the data into two clusters provides a better separation and compactness compared to other values of k. Additionally, the plot reveals that the lowest Calinski-Harabasz Index value of approximately 112 is observed when k=10. This suggests that dividing the data into 10 clusters results in less distinct and more scattered clusters, leading to a poorer clustering solution. Therefore as confirmed by the plot, it was reasonable to choose k=2 as the optimal number of clusters, as it corresponds to the highest Calinski-Harabasz Index value.

Financial Forecasting Part

A)

The Autoregressive (AR) Method:

The exchange rate's time-delayed values serve as input variables for the AR approach. This indicates that the exchange rate's previous values at various time lags make up the input vector. A one-step-ahead prediction, for instance, would have the exchange rate values at t-1, t-2, t-3, and so on in the input vector.

Moving Normal (Ma) Approach:

The moving averages of the exchange rate over a predetermined number of time periods are included in the input vector of the MA method. Short-term trends can be better captured and data noise can be reduced thanks to this. The moving average of the exchange rate over the previous five days, ten days, or any other window size could be included in the input vector, for example.

Technical Indicators:

The use of a variety of technical indicators as input variables is one more strategy. Mathematical calculations based on historical price and volume data are known as technical indicators. Models incorporate moving midpoints, relative strength record (RSI), stochastic oscillator, and Bollinger Groups. Market trends, volatility, and conditions of overbought or oversold can all benefit from additional information from these indicators.

External Components:

Relevant external factors that could have an impact on exchange rates can also be included in the input vector. Macroeconomic indicators like interest rates, inflation rates, GDP growth, or geopolitical events could be these factors. The neural network is capable of capturing the influence of fundamental factors on exchange rate movements by including such variables.

Sentiment Analysis:

Market or news sentiment can be used as input variables with the help of techniques for sentiment analysis. This involves looking at data from social media, news articles, or other sources to figure out how people feel about the currency pair you're considering (positive, negative, or neutral). The dynamics of the market can be better understood with the help of this information.

References:

Cheung, Y. W., & Chinn, M. D. (1998). Integration, cointegration, and the forecast consistency of structural exchange rate models. Journal of International Money and Finance, 17(5).

Exchange Rate Forecasting with An Artificial Neural Network Model: Can We Beat a Random Walk Model? By Y. SUN

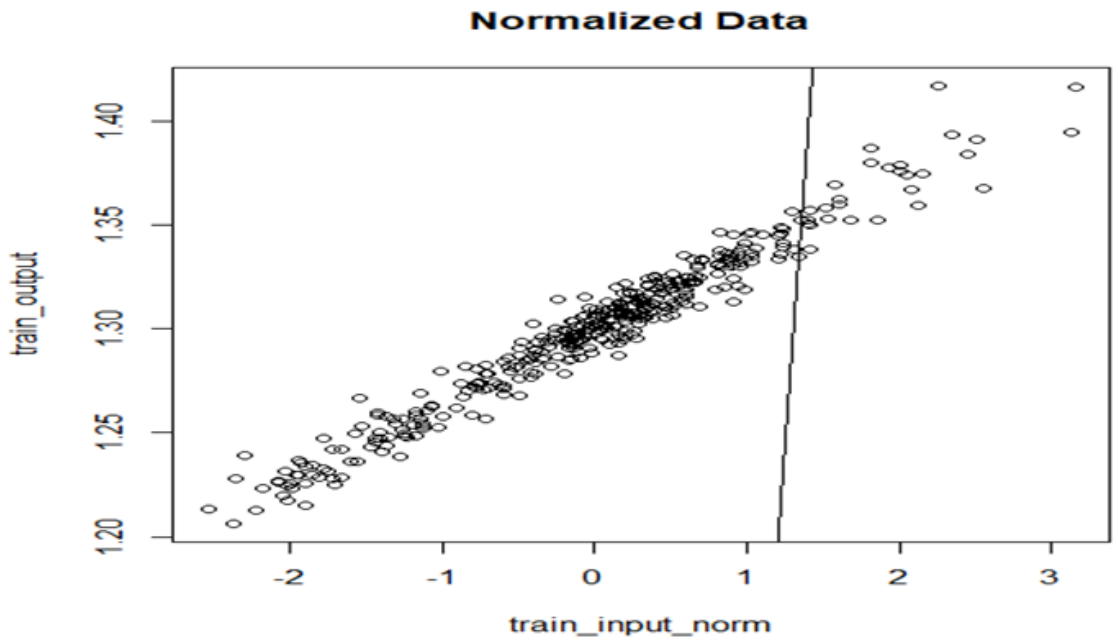
B)

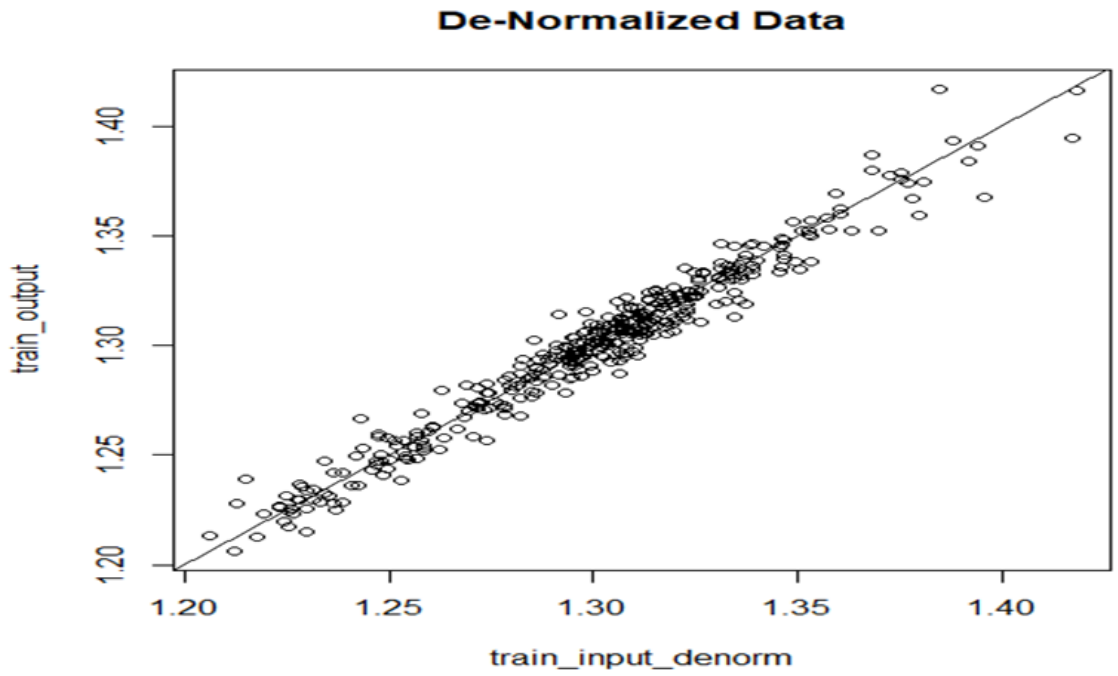
```
Training input matrix dimensions: 400 4
> cat("Training output vector length:", length(train_output), "\n")
Training output vector length: 400
> cat("Testing input matrix dimensions:", dim(test_input), "\n")
Testing input matrix dimensions: 96 4
> cat("Testing output vector length:", length(test_output), "\n")
Testing output vector length: 96
```

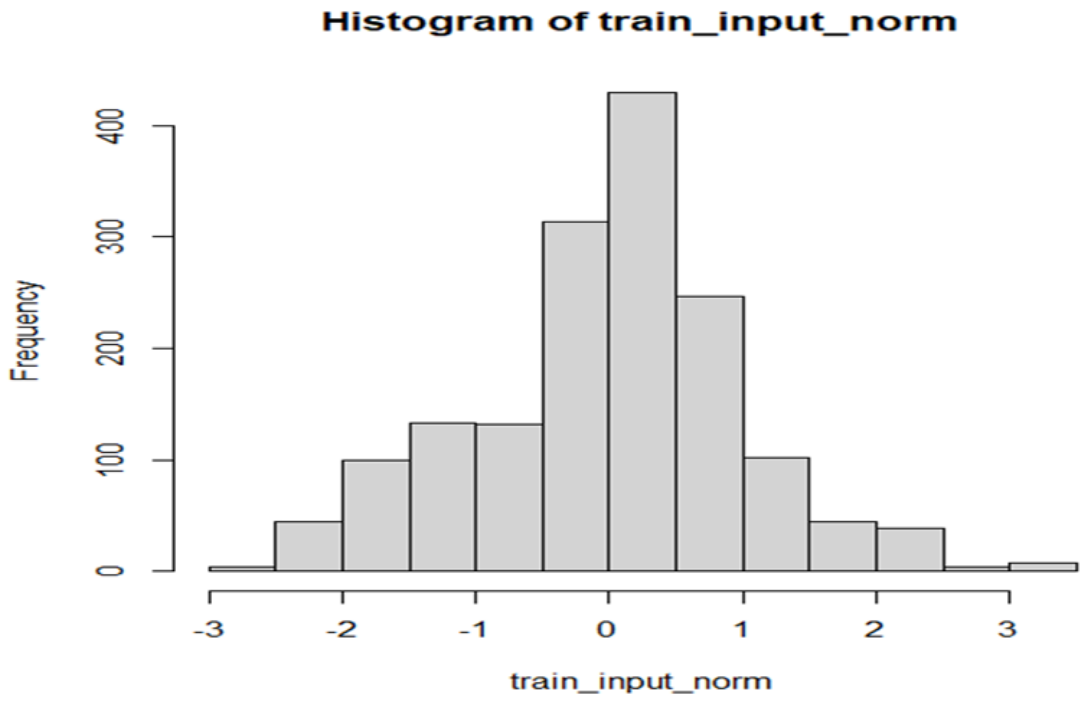
These dimensions indicate that the training input matrix has 400 samples, each with 4 time-delayed input variables. The training output vector has 400 corresponding output values.

Similarly, the testing input matrix has 96 samples, each with 4 time-delayed input variables. The testing output vector has 96 corresponding output values.

C)







Normalizing the data before using it in an MLP structure is a standard procedure, and it serves several important purposes:

Scale Independence: The scale of the input variables affects MLPs. The neural network might give more weight to variables with higher values when the scales of the input variables are very different. By normalizing the data, we ensure that each input variable contributes equally to learning by bringing them all to the same scale. This helps prevent bias in favor of some variables solely because of how big they are.

More rapid convergence: Normalizing the information can prompt quicker assembly during the preparation interaction. The MLP's weights and biases can adjust more effectively when the scales of the input variables are the same. Learning and convergence toward an ideal solution may speed up as a result.
Numerical Stability Avoidance: Numerical instability in the MLP can result from unnormalized data with extensive value ranges. The activation functions or gradient calculations may reach saturation with large values, or they may produce large gradients, which may impede the learning process. By normalizing the data, numerical instability issues like these are less likely to occur.

Generalization: The MLP's ability to generalize can benefit from normalization. The MLP is able to generalize unseen data more effectively and becomes less sensitive to specific values when the data are normalized to a common range, such as between 0 and 1. This improves the model's ability to accurately predict new, untested samples and helps to prevent overfitting.

D)

```
library(caret)
library(neuralnet)

# To calculate sMAPE (symmetric MAPE)

smape <- function(actual, predicted) {

200 * mean(abs(actual - predicted) / (abs(actual) + abs(predicted)))

}



# Define a list of  input vectors (time delays)

input_vectors <- list(1:1, 1:2, 1:3, 1:4)



# Define a list of  MLP models

mlp_models <- list(

  mlp1 = list(hidden = c(5), linear.output = TRUE, activation = "logistic", learningrate = 0.01),

  mlp2 = list(hidden = c(10, 5), linear.output = TRUE, activation = "tanh", learningrate =
0.001),

  mlp3 = list(hidden = c(15, 10, 5), linear.output = TRUE, activation = "relu", learningrate =
0.0001),
```

```
  mlp4 = list(hidden = c(5), linear.output = FALSE, activation = "logistic", learningrate = 0.01),

  mlp5 = list(hidden = c(10, 5), linear.output = FALSE, activation = "tanh", learningrate =
0.001),

  mlp6 = list(hidden = c(15, 10, 5), linear.output = FALSE, activation = "relu", learningrate =
0.0001)

)


results <- list()


for (input_vector in input_vectors) {

  for (i in 1:length(mlp_models)) {

          model <- neuralnet(

          formula = train_output ~ .,

          data = cbind(train_input[, input_vector], train_output),

          hidden = mlp_models[[i]]$hidden,

          linear.output = mlp_models[[i]]$linear.output,

          learningrate = mlp_models[[i]]$learningrate


          )



          predictions <- predict(model, cbind(test_input[, input_vector]))


          # Calculate evaluation metrics

          rmse <- RMSE(predictions, test_output)

          mae <- MAE(predictions, test_output)

          mape <- mean(abs(predictions - test_output) / test_output) * 100
```

```
            smape_val <- smape(predictions, test_output)

            # Store the results
            result <- list(
            input_vector = input_vector,
            model_name = paste0("mlp", i),
            RMSE = rmse,
            MAE = mae,
            MAPE = mape,
            sMAPE = smape_val
            )
            results[[length(results) + 1]] <- result
  }
}
# The results
for (result in results) {
  cat("Input Vector:", paste(result$input_vector, collapse = "-"), "\n")
  cat("Model:", result$model_name, "\n")
  cat("RMSE:", result$RMSE, "\n")
  cat("MAE:", result$MAE, "\n")
  cat("MAPE:", result$MAPE, "\n")
  cat("sMAPE:", result$sMAPE, "\n")
  cat("\n")
}
```

E)

These are the 4 stat indices:

Root Mean Square Error (RMSE):

RMSE measures the average magnitude of the differences between predicted and actual values, taking into account both the direction and the magnitude of the errors. It is calculated by taking the square root of the average of the squared differences between the predicted and actual values. RMSE provides a measure of the overall model performance, where lower values indicate better accuracy. However, RMSE is sensitive to outliers and larger errors can disproportionately affect its value.

Mean Absolute Error (MAE):

MAE represents the average absolute difference between predicted and actual values. It measures the average magnitude of the errors without considering their direction. MAE is calculated by taking the average of the absolute differences between the predicted and actual values. Similar to RMSE, lower MAE values indicate better accuracy. MAE is less sensitive to outliers compared to RMSE.

Mean Absolute Percentage Error (MAPE):

MAPE is a relative measure of the forecast accuracy, expressed as a percentage. It calculates the average percentage difference between the predicted and actual values. MAPE is calculated by taking the average of the absolute percentage differences between the predicted and actual values. MAPE provides insights into the relative magnitude of errors, making it useful for comparing forecasting performance across different datasets. However, MAPE has some limitations, particularly when actual values are close to zero or when zero values are present.

Symmetric Mean Absolute Percentage Error (sMAPE):

sMAPE is another relative measure of forecast accuracy, which addresses some of the limitations of MAPE. It calculates the average percentage difference between the predicted and actual values, but unlike MAPE, it takes into account the average of the predicted and actual values. sMAPE is calculated by taking the average of the absolute percentage differences between the predicted and actual values, normalized by the average of the predicted and actual values. sMAPE provides a balanced measure of accuracy and is suitable for datasets with varying magnitudes. It is also symmetric, meaning it treats overestimations and underestimations equally.

F)

Model	Description	RMSE	MAE	MAPE	SMAPE
MLP1	1 input vector, 1 hidden layers, 5 nodes, linear output, learningrate = 0.01 and logistic activation.	0.00639	0.00495	0.3742	0.3748
MLP2	1 input vector, 2 hidden layers, 10, 5 nodes, linear output learningrate = 0.001, and tanh activation.	0.00623	0.00478	0.3612	0.3617
MLP3	1 input vector, 3 hidden layers, 15,10,5 nodes, linear output learningrate = 0.0001, and relu activation.	0.00628	0.00484	0.3655	0.3659
MLP4	1 input vector, 1 hidden layers, 5 nodes, non-linear output, learningrate = 0.01 and logistic activation.	0.3249	0.3243	24.475	27.897
MLP5	1 input vector, 2 hidden layers, 10, 5 nodes, non-linear output learningrate = 0.001, and tanh activation.	0.3249	0.3243	24.4754	27.8975
MLP6	1 input vector, 3 hidden layers, 15, 10, 5 nodes, non-linear output, learningrate =	0.32495	0.3244	24.48	27.9

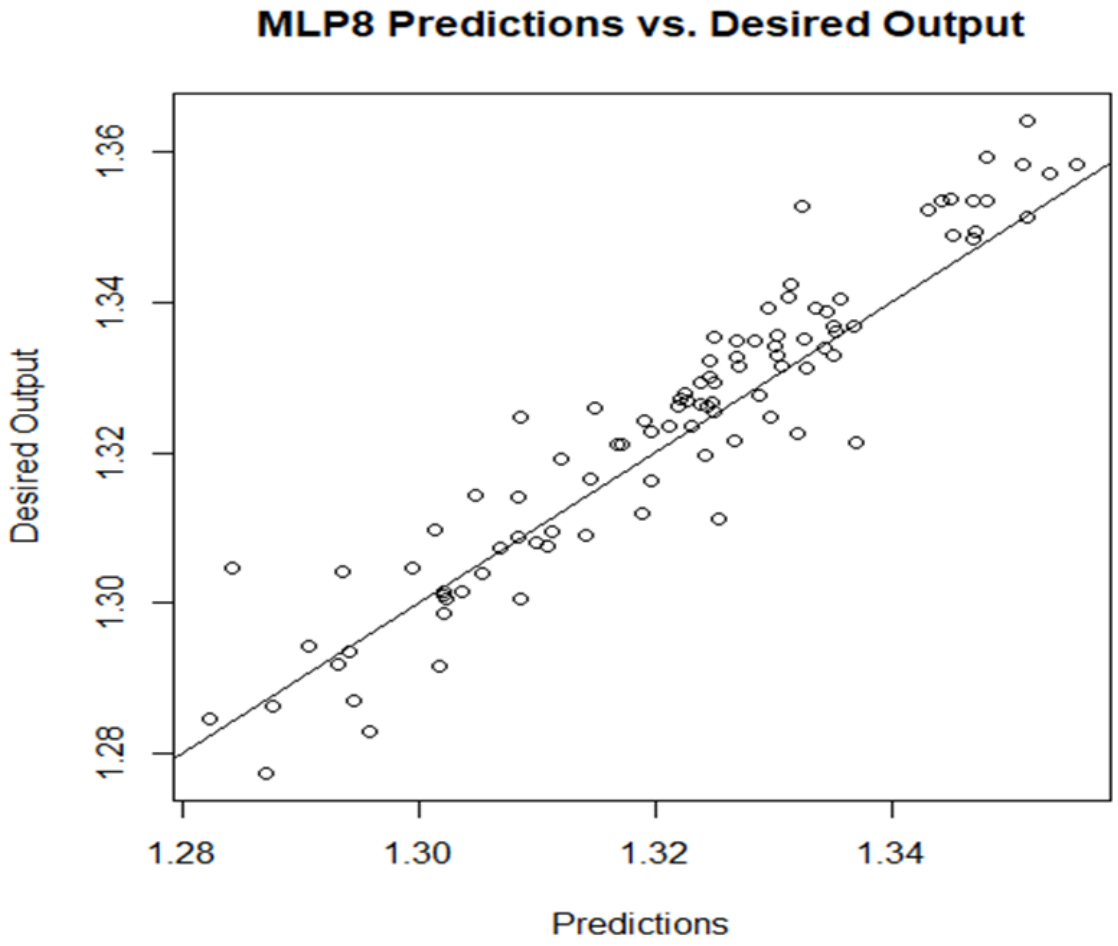
	0.0001 and relu activation.				
MLP7	1-2 input vector, 1 hidden layers, 5 nodes, linear output, learningrate = 0.01 and logistic activation.	0.00651	0.0050	0.3777	0.3783
MLP8	1-2 input vector, 2 hidden layers, 10, 5 nodes, linear output learningrate = 0.001, and tanh activation.	0.00649	0.00494	0.3735	0.3739
MLP9	1-2 input vector, 3 hidden layers, 15, 10, 5 nodes, linear output learningrate = 0.0001, and relu activation.	0.00765	0.00583	0.4405	0.4413
MLP10	1-2 input vector, 1 hidden layers, 5 nodes, non-linear output, learningrate = 0.01 and logistic activation.	0.3249	0.3243	24.475	27.897
MLP11	1-2 input vector, 2 hidden layers, 10, 5 nodes, non-linear output, learningrate = 0.001 and tanh activation.	0.3249	0.3243	24.4751	27.8971
MLP12	1-2 input vector, 3 hidden layers, 15, 10, 5 nodes, non-linear output, learningrate = 0.0001 and relu activation.	0.3250	0.3244	24.477	27.899

MLP13	1-2-3 input vector, 1 hidden layers, 5 nodes, linear output, and logistic activation.	0.00705	0.0057	0.4267	0.4271

G)

To determine the efficiency of the best one-hidden layer and two-hidden layer networks, we can consider the total number of weight parameters per network. Generally, a more efficient network would have fewer parameters while maintaining good performance. Let's analyze the table to make a comparison:
Best one-hidden layer network: MLP1
Number of hidden layers: 1
Number of nodes in the hidden layer: 5
Total number of weight parameters for MLP1: (5*1)+(1*5)= 10
Best two-hidden layer network: MLP8
Number of hidden layers: 2
Number of nodes in the first hidden layer: 10
Number of nodes in the second hidden layer: 5
Total number of weight parameters for MLP8 = (2 * 10) + (10 * 5) + (5 * 1)
= 20 + 50 + 5
= 75
Comparing the two networks, we can observe that MLP8 with two-hidden layers has more weight parameters compared to MLP1 with a single hidden layer. This is because MLP8 has additional connections between the first and second hidden layers, leading to a higher number of weights.
In terms of efficiency, then MLP1 is more efficient than MLP8 because it has a lower total number of weight parameters compared to MLP8.
However, I prefer the MLP8 approach because additional weights in MLP8 (two-hidden layer network) may provide more flexibility in capturing complex patterns in the data, potentially leading to improved forecasting accuracy. It also has a slightly lower MAE, MAPE and sMAPE than MLP1.

H)




```
> cat("RMSE:", rmse, "\n")
RMSE: 0.006869618
> cat("MAE:", mae, "\n")
MAE: 0.005354525
> cat("MAPE:", mape, "\n")
MAPE: 0.4039942
> cat("sMAPE:", smape_val, "\n")
sMAPE: 0.4045959
```

Full Code

Partitioning Clusering Part

#ANS part-a
library(readxl)
library(dplyr)
file_path <- "D:/New folder/vehicles.xlsx"

```
data<- read_excel(file_path)
# Extract the input variables
X <- data[, 1:18]

# Detect outliers using the Z-score method
outliers <- as.vector(which(apply(X, 2, function(x) any(abs(x) > 3))))

# Remove outliers
cleaned_X <- X[-outliers, ]

# Perform scaling using scale()
scaled_X <- scale(X)

#ANS part-b
data_subset<-scaled_X

library(NbClust)
library(cluster)
library(factoextra)
#THE FOUR AUTOMATED TOOLS
# 1. NBclust method
nb_results <- NbClust(data_subset, diss = NULL, distance = "euclidean", min.nc = 2, max.nc
= 10, method = "kmeans")
nb_optimal_clusters <- nb_results$Best.nc
cat("Number of clusters suggested by NBclust:", nb_optimal_clusters, "\n")

# 2. Elbow method
fviz_nbclust(data_subset, kmeans, method = "wss") + ggtitle("Elbow Method")


# 3. Gap statistics method
gap_stat <- clusGap(data_subset, FUNcluster = kmeans, K.max = 10, B = 50)
print(gap_stat$Tab)
fviz_gap_stat(gap_stat)

# 4. Silhouette method
fviz_nbclust(data_subset, kmeans, method = "silhouette")

#ANS part c
library(cluster)
k<-2
kmeans_model <- kmeans(data_subset, centers = k, nstart = 10)
cluster_centers <- kmeans_model$centers
cat("Cluster Centers:\n")
print(cluster_centers)
cluster_assignments <- kmeans_model$cluster
# Display the clustered results
```

```
cat("Cluster Assignments:\n")
print(cluster_assignments)

# Calculate the within-cluster sum of squares (WSS)
wss <- kmeans_model$tot.withinss

# Calculate the between-cluster sum of squares (BSS)
tss <- sum(kmeans_model$tot.withinss) + sum(kmeans_model$betweenss)
bss <- tss - wss

print(paste("The value of BSS is:", round(bss, 4)))
print(paste("The value of WSS is:", round(wss, 4)))

bss_ratio <- bss / tss
cat("BSS/TSS Ratio:", bss_ratio, "\n")

#ANS part d
silhouette_scores <- silhouette(kmeans_model$cluster, dist(data_subset))
plot(silhouette_scores, main = "Silhouette Plot",border=NA,col="darkblue")

avg_sil_width <- mean(silhouette_scores[, "sil_width"])
print(paste("The average silhouette width is:", round(avg_sil_width, 4)))



# ANS 2 part e

pca <- prcomp(data_subset, scale = TRUE)
eigenvalues <- pca$sdev^2
eigenvectors <- pca$rotation

print(paste0(" These are the eigenvalues: ", eigenvalues))
print(paste0(" These are the eigenvectors: ", eigenvectors))

cumulative_score <- cumsum(eigenvalues) / sum(eigenvalues) * 100
cumulative_score
selected_pcs <- which(cumulative_score > 92)
transformed_data <- data_subset[, -19] %*% eigenvectors[, selected_pcs]

# ANS 2 part f
library(NbClust)
library(cluster)
#NBclust
nb_results <- NbClust(transformed_data, diss = NULL, distance = "euclidean", min.nc = 2,
max.nc = 10, method = "kmeans")

#ELBOW
fviz_nbclust(transformed_data, kmeans, method = "wss") + ggtitle("Elbow Method")
```

```
#Gap Statistics
gap_stat <- clusGap(transformed_data, FUNcluster = kmeans, K.max = 10, B = 50)
print(gap_stat$Tab)

fviz_gap_stat(gap_stat)


#Silhouette
# 4. Silhouette method
fviz_nbclust(transformed_data, kmeans, method = "silhouette")


# ANS 2 part g
k <- 2
kmeans_model <- kmeans(transformed_data,centers = k, nstart = 10)
cluster_centers <- kmeans_model$centers
cat("Cluster Centers:\n")
print(cluster_centers)
cluster_assignments <- kmeans_model$cluster
# Display the clustered results
cat("Cluster Assignments:\n")
print(cluster_assignments)

# Calculate the within-cluster sum of squares (WSS)
wss <- kmeans_model$tot.withinss

# Calculate the between-cluster sum of squares (BSS)
tss <- sum(kmeans_model$tot.withinss) + sum(kmeans_model$betweenss)
bss <- tss - wss

print(paste("The value of BSS is:", round(bss, 4)))
print(paste("The value of WSS is:", round(wss, 4)))

bss_ratio <- bss / tss
cat("BSS/TSS Ratio:", bss_ratio, "\n")

# ANS 2 part h
silhouette_scores <- silhouette(kmeans_model$cluster, dist(transformed_data))
plot(silhouette_scores, main = "Silhouette Plot",border=NA,col="darkblue")

avg_sil_width <- mean(silhouette_scores[, "sil_width"])
print(paste("The average silhouette width is:", round(avg_sil_width, 4)))


# ANS 2 part i
library(fpc)
```

```
library(ggplot2)

# Initialize empty vectors to store the index values and k values
index_values <- c()
k_values <- c()

# Iterate over different values of k and calculate the Calinski-Harabasz Index
for (k in 2:10) {
  kmeans_result <- kmeans(transformed_data, centers = k)
  diss_matrix <- dist(transformed_data)
  ch_index <- cluster.stats(diss_matrix, kmeans_result$cluster)$ch
  index_values <- c(index_values, ch_index)
  k_values <- c(k_values, k)
}

# Create a data frame of k values and index values
index_df <- data.frame(k = k_values, index = index_values)

# Create a line plot to illustrate the Calinski-Harabasz Index
ggplot(index_df, aes(x = k, y = index)) +
  geom_line() +
  geom_point() +
  labs(x = "Number of Clusters (k)", y = "Calinski-Harabasz Index") +
  ggtitle("Calinski-Harabasz Index for Different Number of Clusters")
```

Financial Forecasting Part

```
# ans b
# Import the data
library(readxl)
library(caret)
# Load the dataset
dataset <- read_excel("D:/New folder/financial forecasting part.xlsx")
exchange_rates <- dataset$`USD/EUR`

# Define the maximum time delay for the input vector
max_delay <- 4

# Initialize the input/output matrices
input_matrix <- matrix(NA, nrow = length(exchange_rates) - max_delay, ncol = max_delay)
output_vector <- exchange_rates[(max_delay + 1):length(exchange_rates)]

# Construct the input/output matrices
for (i in 1:max_delay) {
  input_matrix[, i] <- exchange_rates[(max_delay - i + 1):(length(exchange_rates) - i)]
}

# Split the data into training and testing sets
```

```
train_samples <- 400
train_input <- input_matrix[1:train_samples, ]
train_output <- output_vector[1:train_samples]
test_input <- input_matrix[(train_samples + 1):nrow(input_matrix), ]

test_output <- output_vector[(train_samples + 1):length(output_vector)]

# Perform any necessary preprocessing or normalization of the input and output data

# Print the dimensions of the input/output matrices
cat("Training input matrix dimensions:", dim(train_input), "\n")
cat("Training output vector length:", length(train_output), "\n")
cat("Testing input matrix dimensions:", dim(test_input), "\n")
cat("Testing output vector length:", length(test_output), "\n")

#ans c
train_input_norm <- scale(train_input)
test_input_norm <- scale(test_input)

# Check the distribution of the normalized data
hist(train_input_norm)

# De-normalize the data
train_input_denorm <- train_input_norm * sd(train_input) + mean(train_input)
test_input_denorm <- test_input_norm * sd(test_input) + mean(test_input)

# Truncate the train_input_denorm vector to the same length as the train_output vector
train_input_denorm <- train_input_denorm[1:length(train_output)]

# Plot the de-normalized data
plot(train_input_denorm, train_output, main = "De-Normalized Data")
abline(0, 1)

train_input_norm <- train_input_norm[1:length(train_output)]

plot(train_input_norm, train_output, main = "Normalized Data")
abline(0, 1)

#ANS d


library(caret)
library(neuralnet)

# To calculate sMAPE (symmetric MAPE)
smape <- function(actual, predicted) {
  200 * mean(abs(actual - predicted) / (abs(actual) + abs(predicted)))
}
```

```
# Define a list of  input vectors (time delays)
input_vectors <- list(1:1, 1:2, 1:3, 1:4)

# Define a list of  MLP models
mlp_models <- list(
  mlp1 = list(hidden = c(5), linear.output = TRUE, activation = "logistic", learningrate = 0.01),
  mlp2 = list(hidden = c(10, 5), linear.output = TRUE, activation = "tanh", learningrate =
0.001),
  mlp3 = list(hidden = c(15, 10, 5), linear.output = TRUE, activation = "relu", learningrate =
0.0001),
  mlp4 = list(hidden = c(5), linear.output = FALSE, activation = "logistic", learningrate = 0.01),
  mlp5 = list(hidden = c(10, 5), linear.output = FALSE, activation = "tanh", learningrate =
0.001),
  mlp6 = list(hidden = c(15, 10, 5), linear.output = FALSE, activation = "relu", learningrate =
0.0001)
)

results <- list()

for (input_vector in input_vectors) {
  for (i in 1:length(mlp_models)) {
    model <- neuralnet(
      formula = train_output ~ .,
      data = cbind(train_input[, input_vector], train_output),
      hidden = mlp_models[[i]]$hidden,
      linear.output = mlp_models[[i]]$linear.output,
      learningrate = mlp_models[[i]]$learningrate

    )


    predictions <- predict(model, cbind(test_input[, input_vector]))

    # Calculate evaluation metrics
    rmse <- RMSE(predictions, test_output)
    mae <- MAE(predictions, test_output)
    mape <- mean(abs(predictions - test_output) / test_output) * 100
    smape_val <- smape(predictions, test_output)

    # Store the results
    result <- list(
      input_vector = input_vector,
      model_name = paste0("mlp", i),
      RMSE = rmse,
      MAE = mae,
      MAPE = mape,
      sMAPE = smape_val
```

```
    )
    results[[length(results) + 1]] <- result
  }
}
# The results
for (result in results) {
  cat("Input Vector:", paste(result$input_vector, collapse = "-"), "\n")
  cat("Model:", result$model_name, "\n")
  cat("RMSE:", result$RMSE, "\n")
  cat("MAE:", result$MAE, "\n")
  cat("MAPE:", result$MAPE, "\n")
  cat("sMAPE:", result$sMAPE, "\n")
  cat("\n")
}

# ans h

library(neuralnet)
library(caret)


smape <- function(actual, predicted) {
  200 * mean(abs(actual - predicted) / (abs(actual) + abs(predicted)))
}

input_vector <- 1:2
model <- neuralnet(
  formula = train_output ~ .,
  data = cbind(train_input[, input_vector], train_output),
  hidden = c(10, 5),
  linear.output = TRUE,
  learningrate = 0.001
)


predictions <- predict(model, cbind(test_input[, input_vector]))

# Calculate stat. indices
rmse <- RMSE(predictions, test_output)
mae <- MAE(predictions, test_output)
mape <- mean(abs(predictions - test_output) / test_output) * 100
smape_val <- smape(predictions, test_output)

# Create a scatter plot
plot(predictions, test_output, main = "MLP8 Predictions vs. Desired Output", xlab =
"Predictions", ylab = "Desired Output")
abline(0, 1)
```

```
# Print the stat. indices
cat("RMSE:", rmse, "\n")
cat("MAE:", mae, "\n")
cat("MAPE:", mape, "\n")
cat("sMAPE:", smape_val, "\n")
```