UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

University of Westminster & Informatics Institute of Technology

IIT School of Computing,
Degree: BEng(Hons) in Software Engineering

Unit Code and Description:
(2022) DATA001C.2 Machine Learning and Data Mining

Module Leader: Mr.Nipuna Senanayake.

Assignment: Course Work

Assignment Type: Individual Course Work
Issue Date: 2nd February 2023.
Deadline: 04th May 2023 at 01.00 pm.

Tutorial Group : J

Student Name: S.Mohanaranjan.
Student ID
IIT : 20200607
UOW : 18705841 / W1870584

Table of Contents

Table of Figures

Partitioning Clustering Part

In this assignment, we consider a set of observations on a number of silhouettes related to different type of vehicles, using a set of features extracted from the silhouette. Each vehicle may be viewed from one of many different angles. The features were extracted from the silhouettes by the HIPS (Hierarchical Image Processing System) extension BINATTS, which extracts a combination of scale independent features utilising both classical moments based measures such as scaled variance, skewness and kurtosis about the major/minor axes and heuristic measures such as hollows, circularity, rectangularity and compactness. Four model vehicles were used for the experiment: a double decker bus, Chevrolet van, Saab and an Opel Manta. This particular combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be readily distinguishable, but it would be more difficult to distinguish between the cars.

Objectives/Deliverables (partitioning clustering)

One dataset (vehicles.xls) is available and has 846 observations/vehicle samples. This dataset is defined by 18 attributes (i.e. input variables) and one output (i.e. class). There are 4 classes. This is a classic multi-dimensional, in terms of features, problem. For this clustering part, you need to use only the first 18 attributes to your calculations. Clustering is an unsupervised scheme, thus, the information included in the "class" attribute can't be used.

Description of attributes:

1. Comp: Compactness
2. Circ: Circularity
3. D.Circ: Distance Circularity
4. Rad.Ra: Radius ratio
5. Pr.Axis.Ra: pr.axis aspect ratio
6. Max.L.Ra: max.length aspect ratio
7. Scat.Ra: scatter ratio
8. Elong: elongatedness
9. Pr.Axis.Rect: pr.axis rectangularity
10. Max.L.Rect: max.length rectangularity
11. Sc.Var.Maxis: scaled variance along major axis
12. Sc.Var.maxis: scaled variance along minor axis
13. Ra.Gyr: scaled radius of gyration
14. Skew.Maxis: skewness about major axis
15. Skew.maxis: skewness about minor axis
16. Kurt.maxis: kurtosis about minor axis
17. Kurt.Maxis: kurtosis about major axis
18. Holl.Ra: hollows ratio
19. Class: type of cars (desired output)

The work in this part is divided into two subtasks:

- **In the 1st subtask, the analysis will be performed with all initial attributes, as the aim is to assess clustering results using all input variables.**
- **In the 2nd subtask, however, principal component analysis (PCA) will be applied to reduce the input dimensionality and the newly produced dataset will be again clustered. The aim in this 2nd subtask is to help students understand the principles and effects of reducing dimensionality in multi-dimensional problems.**

1st Subtask Objectives:

1. **Before conducting the k-means, perform the following pre-processing tasks: scaling and outliers detection/removal and briefly justify your answer. (Suggestion: the order of scaling and outliers removal is important. The outlier removal topic is not covered in tutorials, so you need to explore it yourself). Obviously, you can implement this clustering task without exploring this "outlier" component, however, you will not be awarded the allocated marks for this component!**

   To find k-means, detect the outliers and remove it from the data. For that we can use
   1. Min-max Normalization
   2. Z-Score Method

   Min-max normalization is a scaling technique used to transform the values of a variable to a range between 0 and 1. While this method can help to reduce the impact of outliers to some extent, it is not a suitable method for detecting or removing outliers from the data due to scale of wide range in the vehicles data set.

   Z-Score Method

   $Z = (X – \mu) / \boldsymbol{\sigma}$

   Z : z-score
   X : Score
   μ : Mean
   $\boldsymbol{\sigma}$: sum of deviation

   Code

```
#Load Library to reading excel file
library(readxl)

#Input the data set
df<-
read_excel("/Users/shanmugaratnammohanaranjan/Desktop/20200607/vehicles.xlsx")

# View the data
View(df)

#creating a variable without sample numbers column and class column
sample = dplyr::select(df, 2:19)
sample

# Calculate the z-score for each attribute
z_scores <- apply(sample, 2, function(x) abs((x - mean(x))/sd(x)))
z_scores

# Set a threshold for z-score
```

```
z_threshold <- 3
z_threshold

# Identify and remove outliers with z-score greater than the threshold
clean_data <- sample[rowSums(z_scores > z_threshold) == 0,]
clean_data

# Scale the cleaned data set
scale_data <- scale(clean_data)
scale_data

#creating a data frame after scaling the data set
clustering <- as.data.frame(scale_data)
clustering
```

Here I am used the z-score method to find and remove outliers for that creating the "sample" data variable to scale the column 2:19 and select the integer data inputs without the string data which contains samples number column and class column. Then use z-score method.
Delete / Remove outliers.
Generally, a norm of 2 or 3 is used as the threshold value for deleting outliers. Because values with a z-score outside of this range are thought to be extremely high and unlikely to happen by accident. They are to be actual data mistakes, measurement errors, or other forms of bias or noisy data. So, taking and assume as 3 and find outliers and remove then assign to scale the data and make it for clustering part.

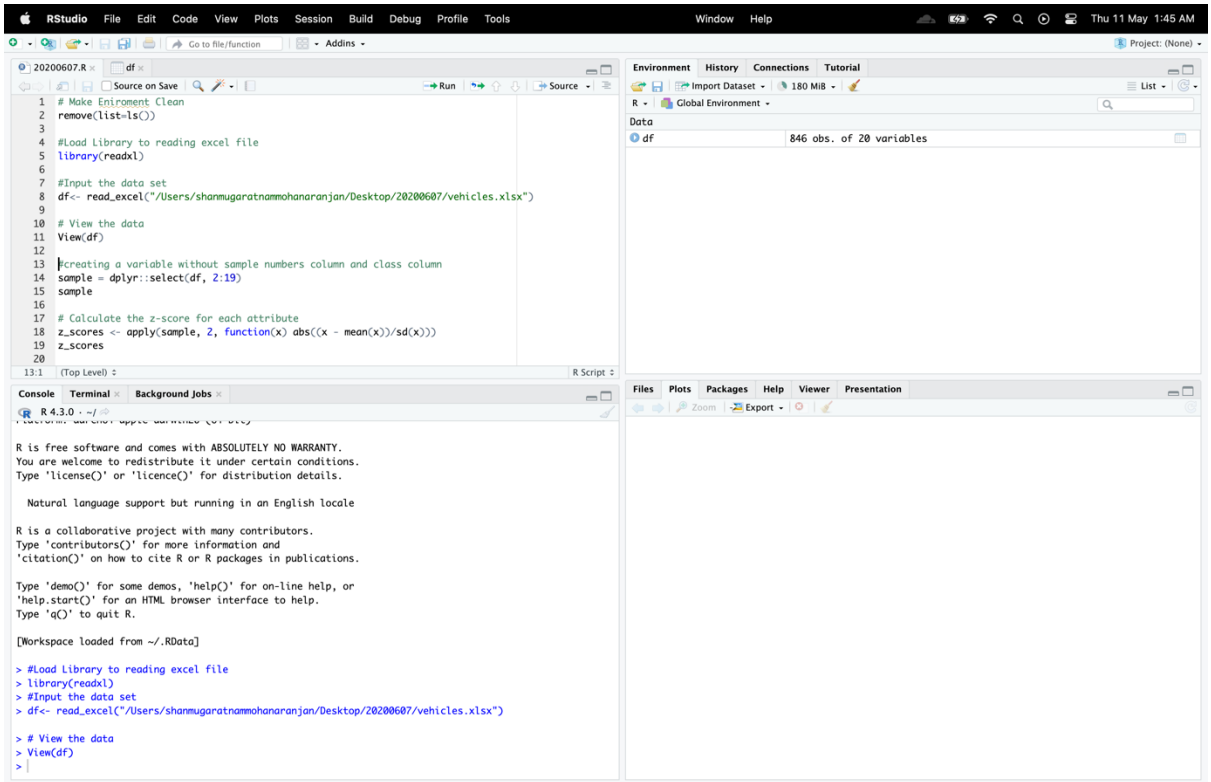

Figure 1: Input Data

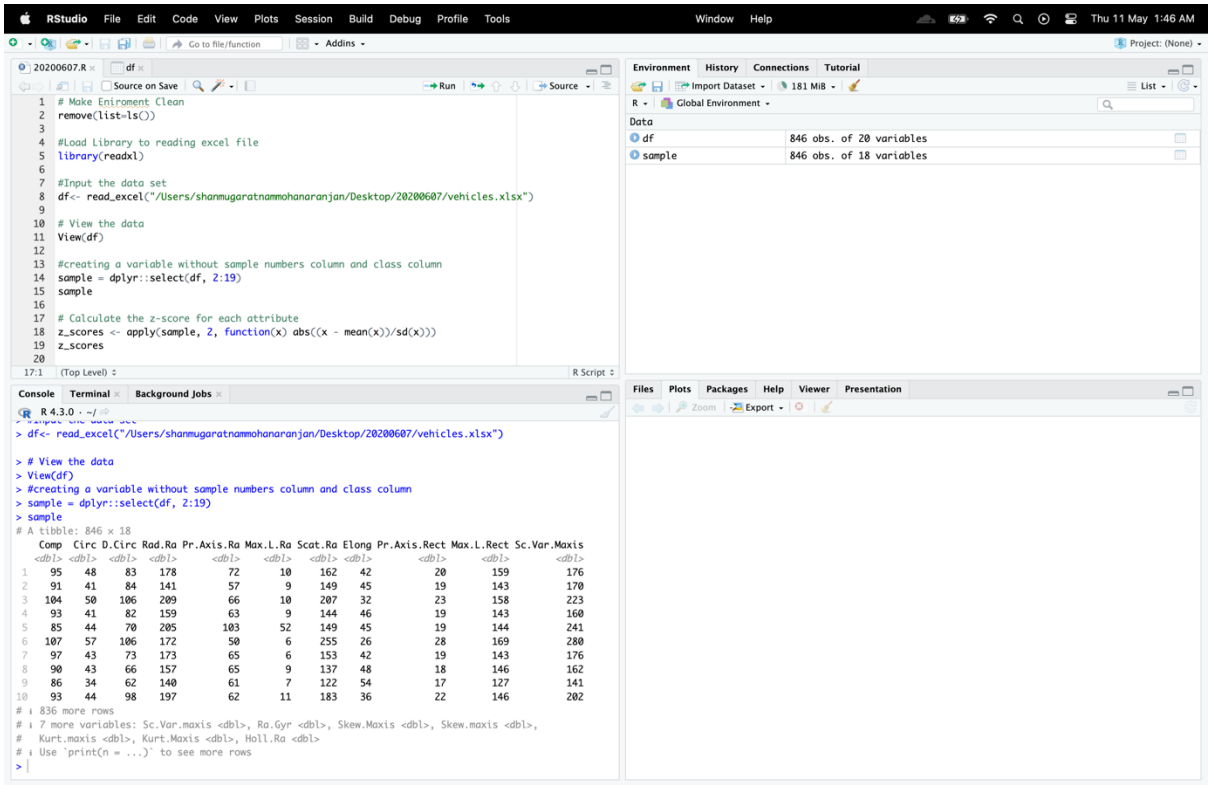

Figure 2 : View of Data

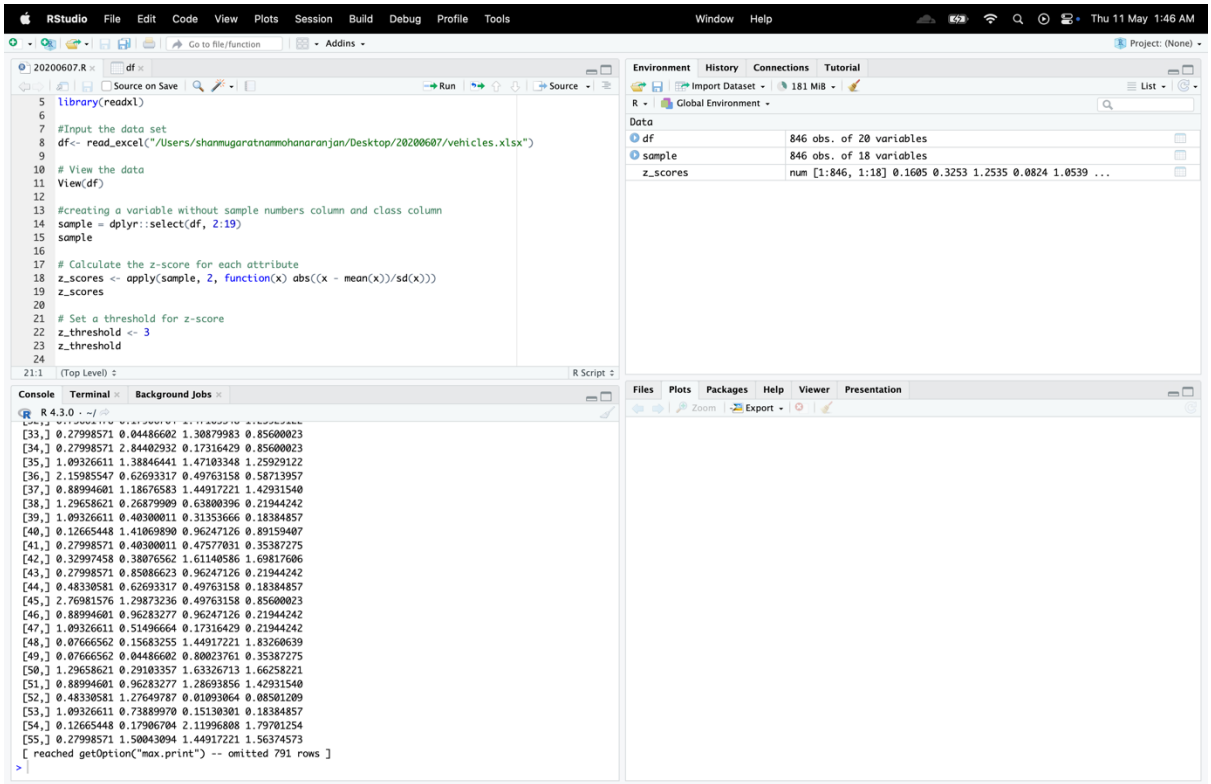

Figure 3: View of Console

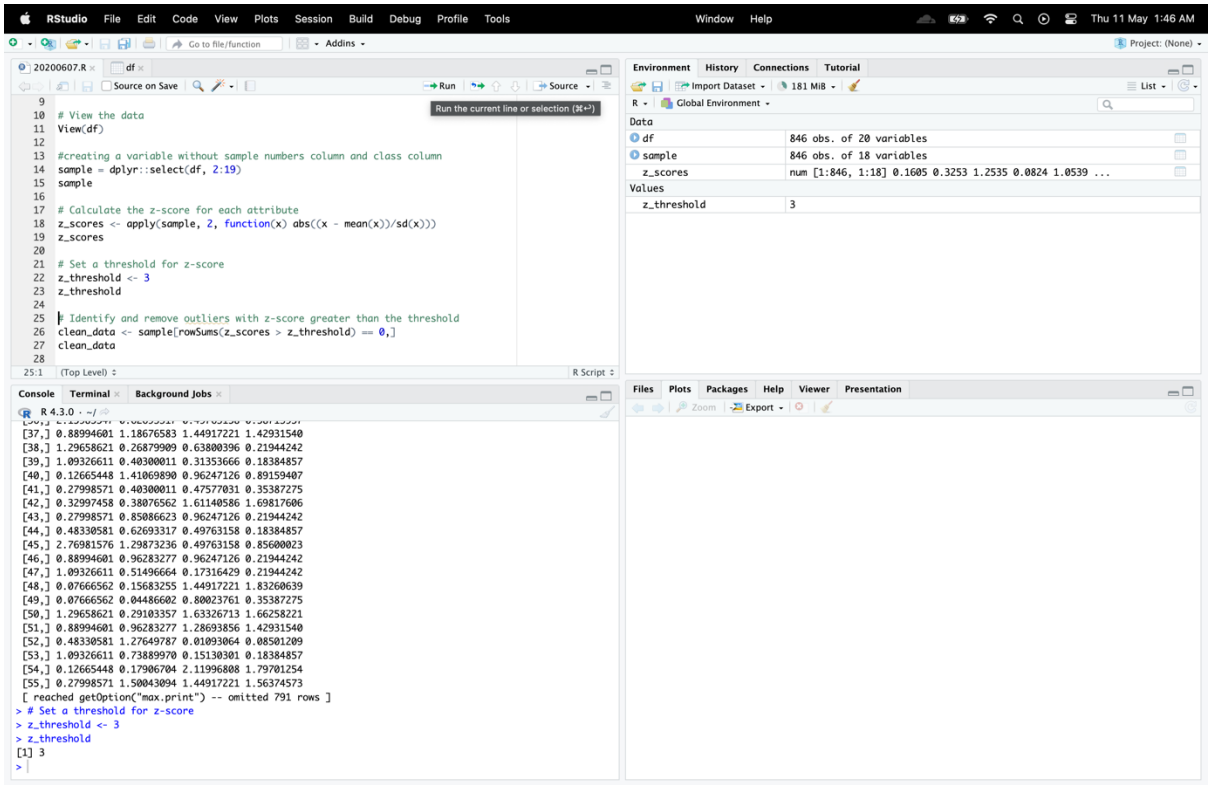

Figure 4: Thresh Hold

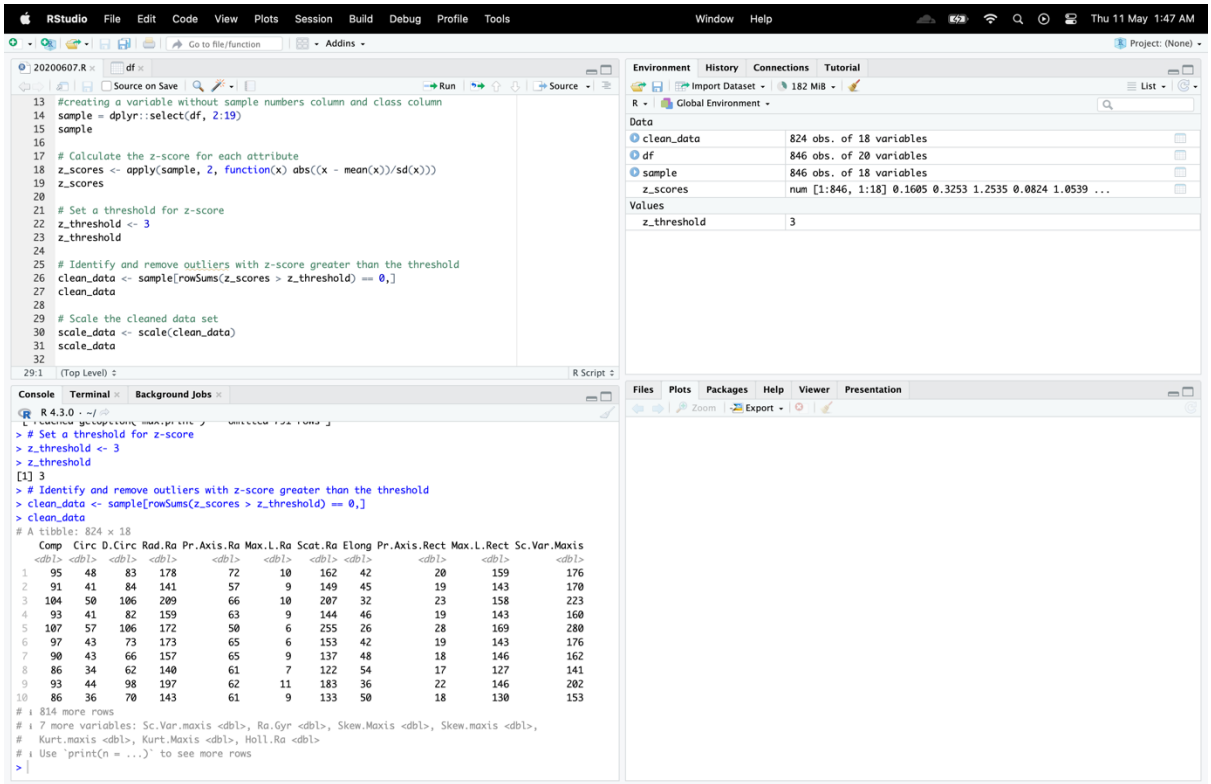

Figure 5: Clean Data

846 – 824 = 22

There are twenty-two outliers found and get removed.

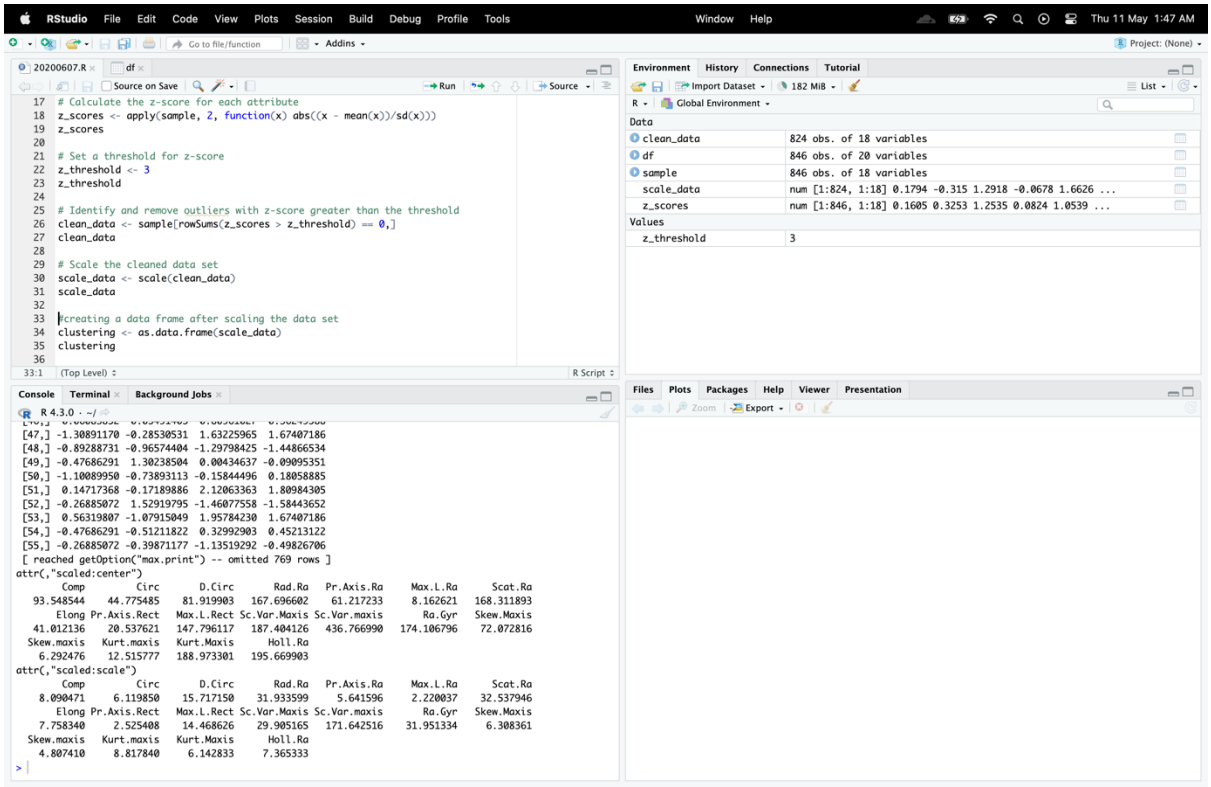

Figure 6: Scale Data

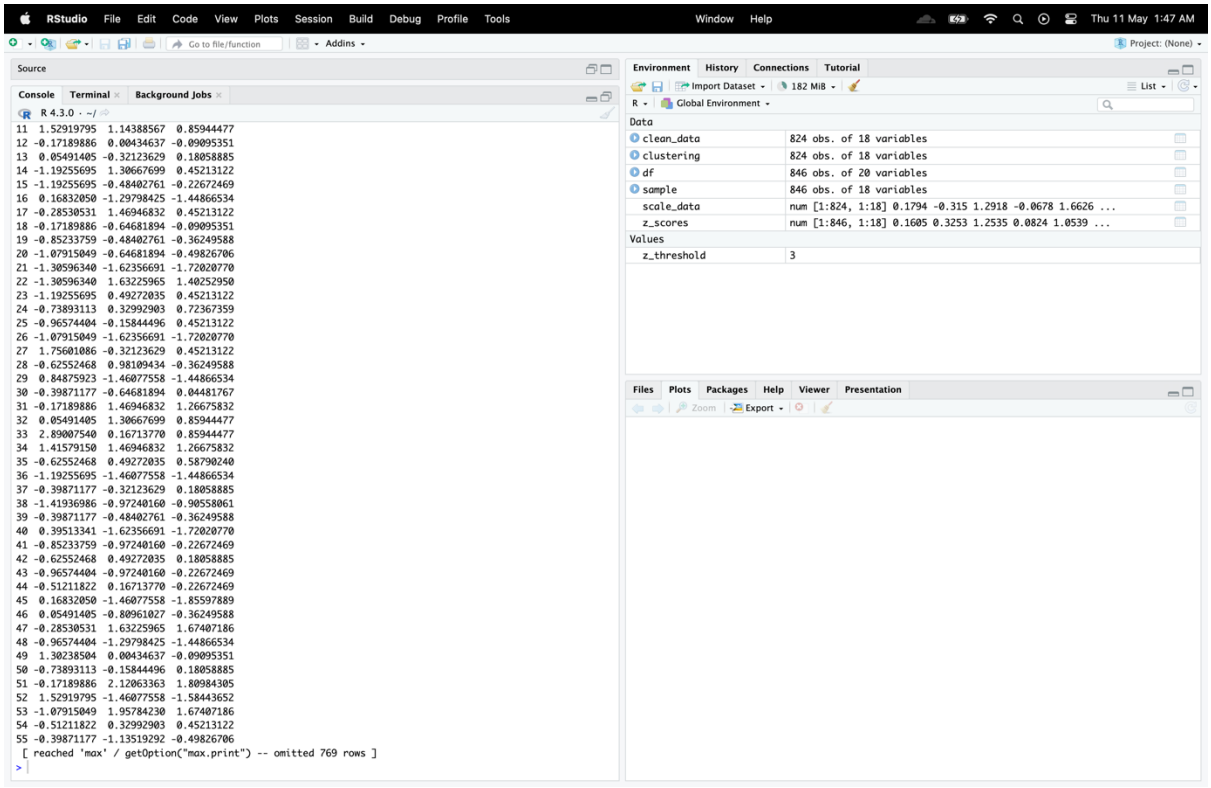

Figure 7: View of Console

2. **You need then to determine the number of cluster centres via four "automated tools". The "automated tools" should include NBclust, Elbow, Gap statistics and silhouette methods. You need to provide, in your report, the related R-outputs and your discussion on these outcomes.**

   Four automated tools to determine the number of cluster centers in R

   NBClust : Using multiple internal and external clustering validation indices, the NbClust R program offers a framework for identifying the ideal number of clusters in a dataset. In order to compare the findings across various values of the number of clusters (k), it computes 30 alternative clustering indices.

   Code

```
#NBClust

#installing and loading nbclust package
# install.packages("NbClust")

# Load NbClust
library(NbClust)

#Rundom number creator for working with random numbers
set.seed(1234)

#assign value to nb
nb <- NbClust(scale_data, distance = "euclidean", min.nc = 2, max.nc = 10, method =
"kmeans")
# Print
cat("optimal number of clusters by nb:", nb$Best.nc, "\n")
```

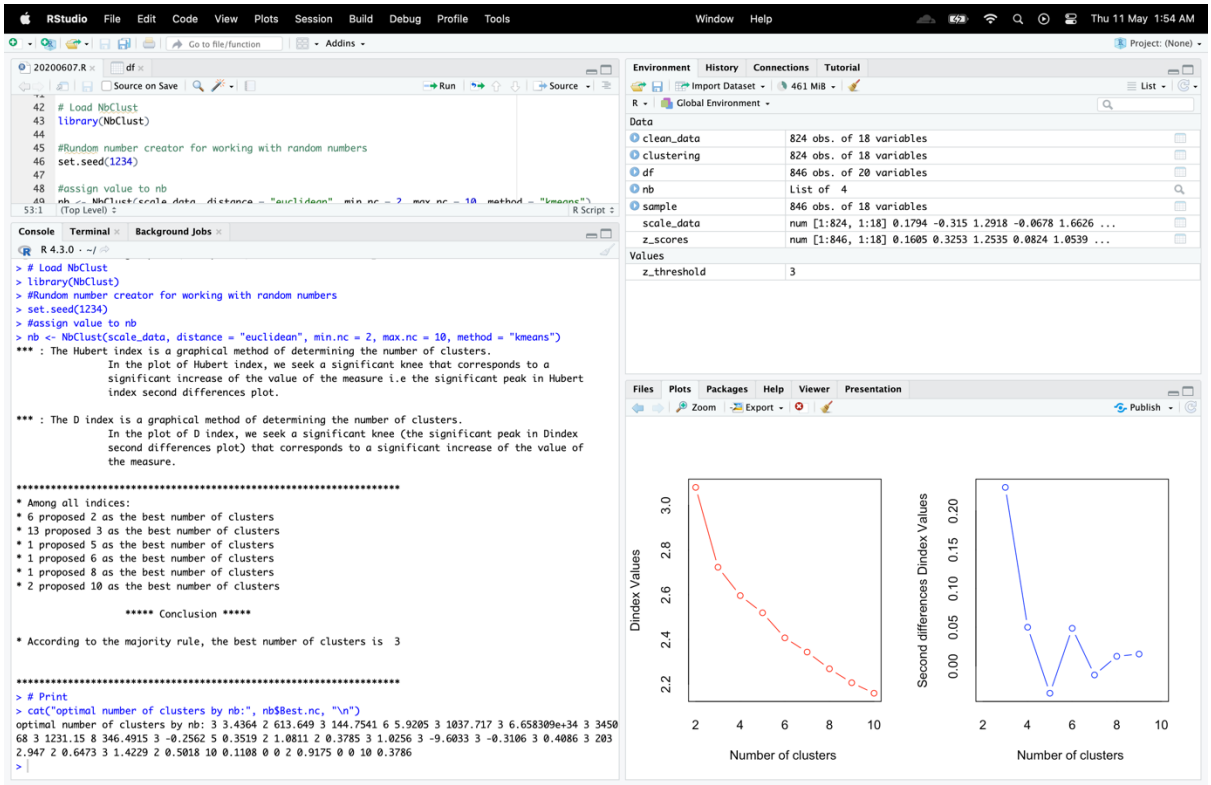

Figure 8: View of NBClust

To utilize the NbClust() function, you must first install and load the NbClust package. To guarantee that the results could be replicated, then i used set.seed(123) to set a random value. Next, determine the ideal number of clusters for the dataset using the NbClust() method. The minimum number and maximum number variable specify the lowest and maximum number of clusters to take into account, respectively, while the distance argument gives the distance metrics to utilize for clustering (in this case, Euclidean distance). The k-means technique is specified by the method argument.

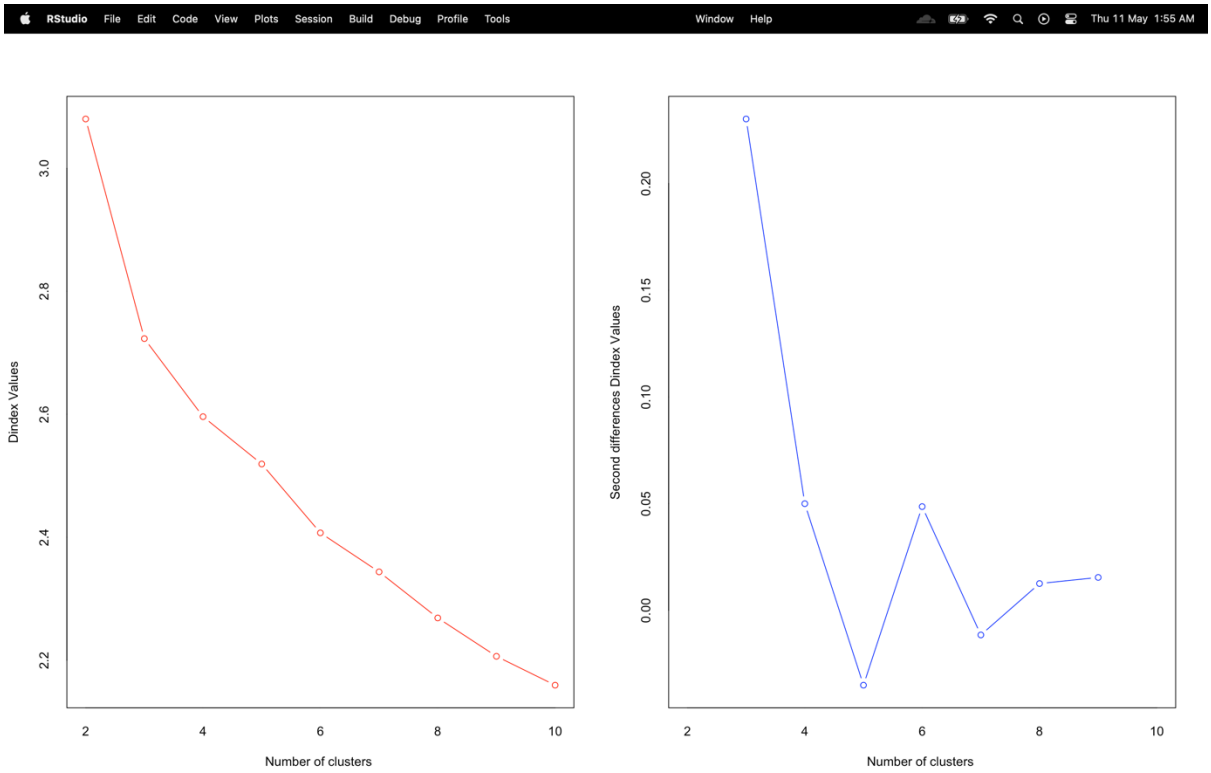

Figure 9: Nb Clust Graph

Elbow : The elbow method is a straightforward yet popular technique for choosing the ideal number of clusters in a dataset. To exhibit the findings on a line graph, the sum of squared distances between each point and its designated cluster centre must be calculated for various values of k. The ideal number of clusters is shown by the "elbow" point on the graph, which is when the slope of the line begins to level off.

Code

```
#Elbow
set.seed(1234)
sse <- numeric(10)

for(k in 2:10){
  kmeans_out <- kmeans(scale_data, centers = k, nstart = 25)
  sse[k] <- kmeans_out$tot.withinss
}

# Plot the sum of squared errors (SSE) for a range of cluster sizes
plot(2:10, sse[2:10], type = "b", main = "Elbow Method to Calculate the Cluster Number")
```

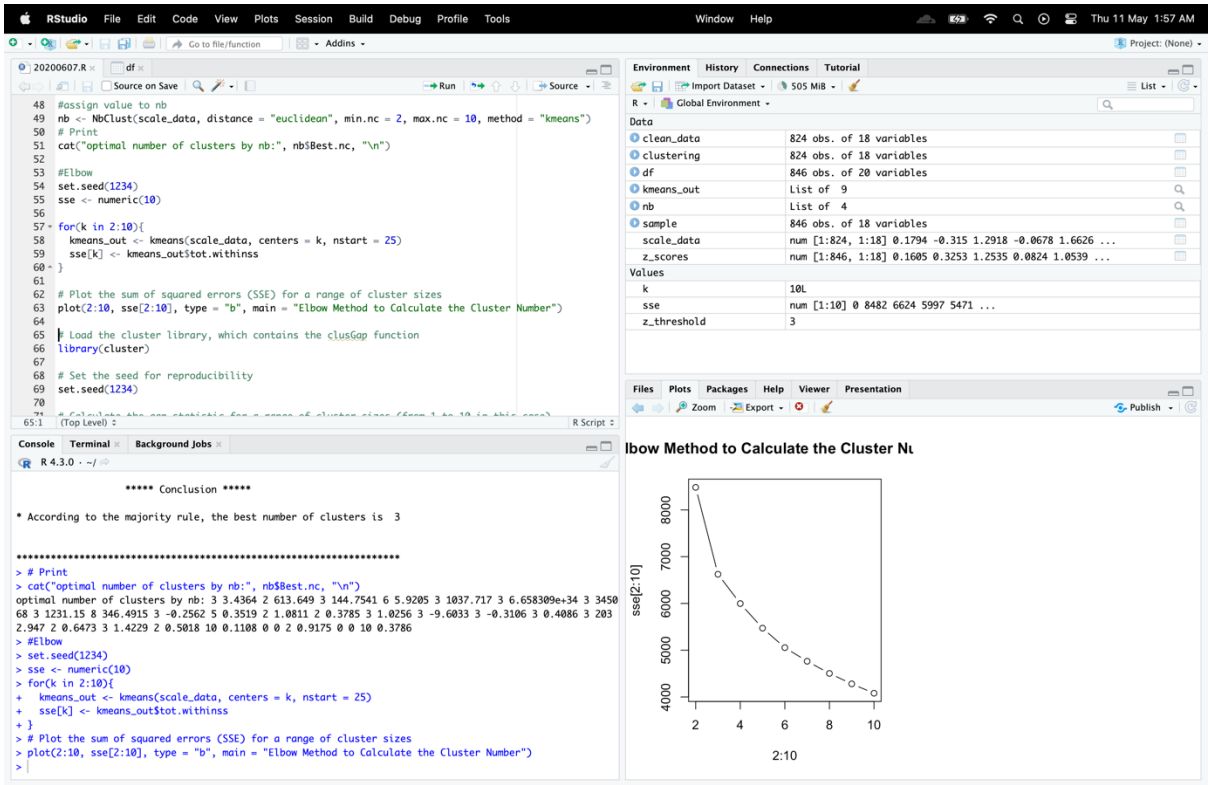

Figure 10: Elbow Method Graph

As with the NbClust approach, results can be replicated. The sum of squared errors (SSE) for each value of k should then be initialized in an empty 10-dimensional numeric vector called sse. Then I iterated through the values of k from 2 to 10 using a for loop. Utilizing the kmeans() function, cluster the dataset according to each value of k. Then set the nstart parameter to 25, which indicates the number of random initializations to utilize, and the centres of argument to k, which indicates the number of clusters to generate. In order to extract the SSE for the current value of k, which is kept in the kmeans_out object's tot.withinss component

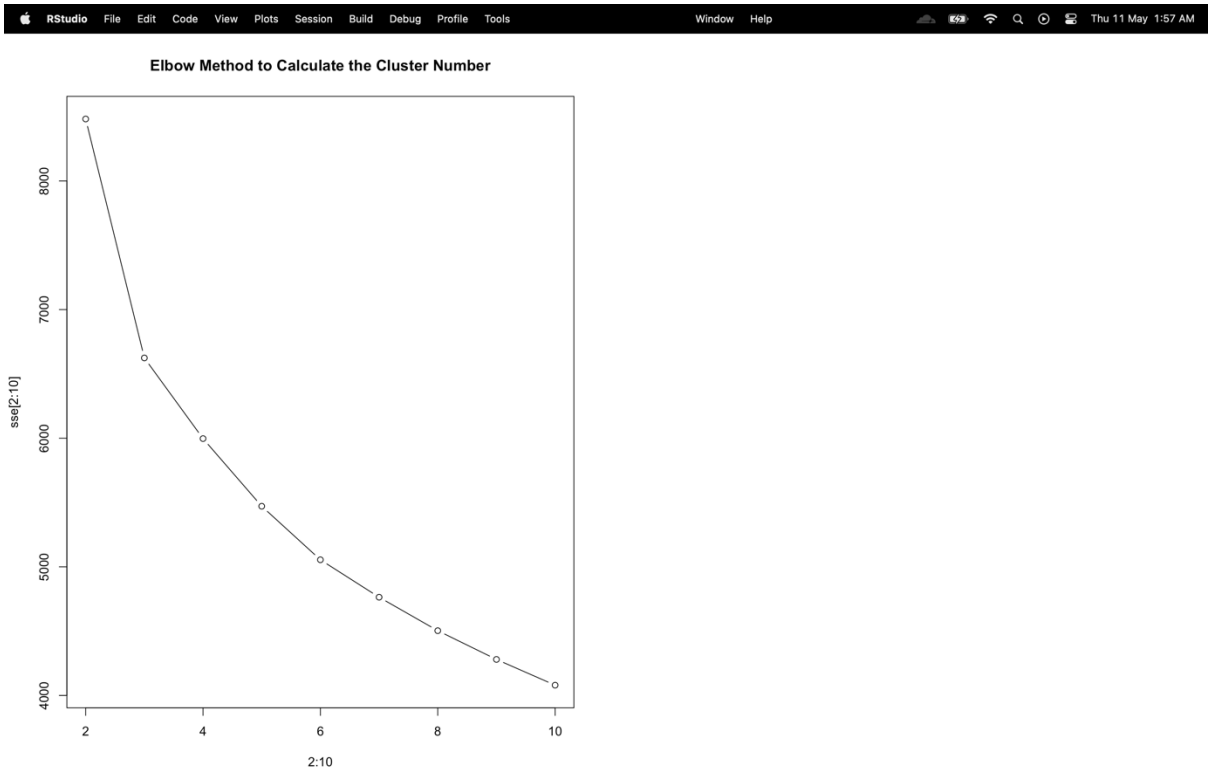

Figure 11: Elbow Method Graph

This technique involves calculating the mean distance and plotting it against the k-values. The objective is to locate the "elbow point," which is when the rate of decline in the mean distance abruptly changes. For the provided dataset, this number of clusters is regarded as the ideal number. 3 is the ideal number of clusters, just like the elbow method, according to the graphic I get from the code.

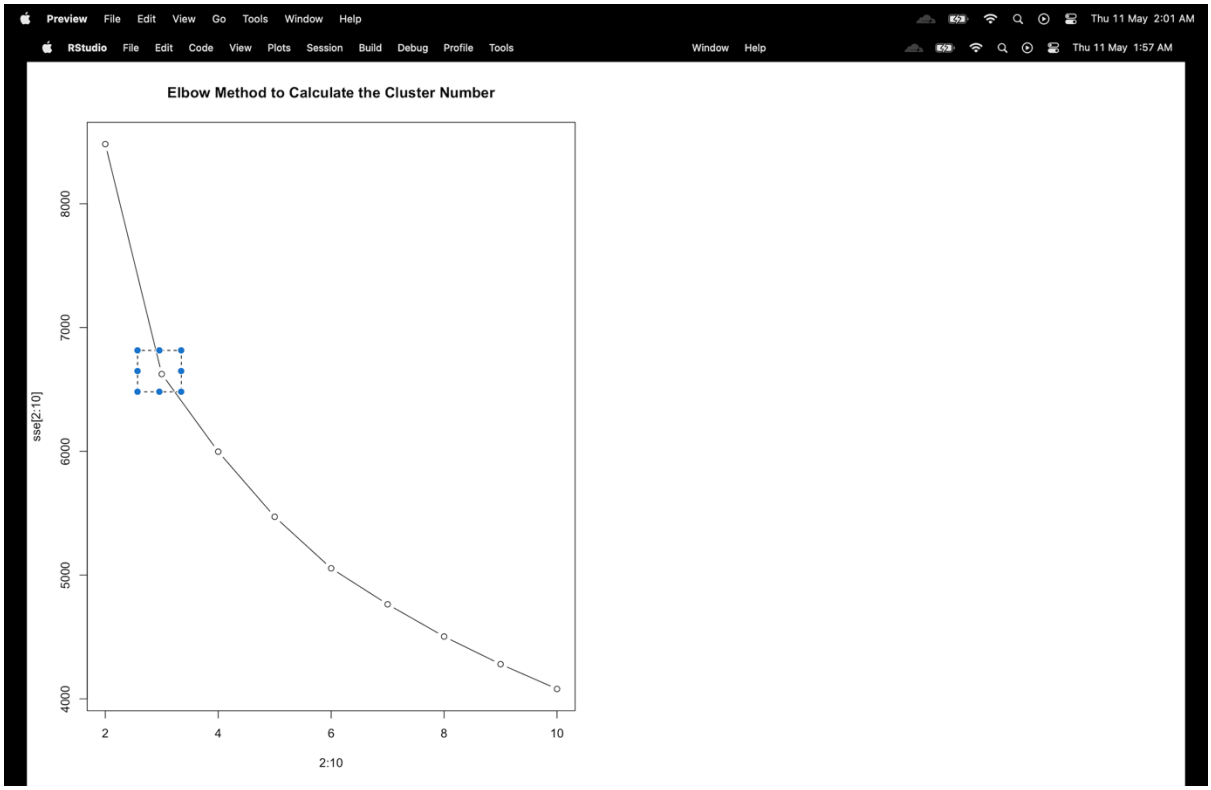

Figure 12: Elbow Point

Gap Statistics : Another well-liked technique for choosing the ideal number of clusters in a dataset is gap statistics. To do this, one must compare the within-cluster dispersion (WCD) for various k-values to the WCD that would be anticipated under a null reference distribution. The number of clusters that maximizes the gap statistic, which is the difference between the observed and expected WCD, is the ideal number of clusters.

Code

```
# Load the cluster library, which contains the clusGap function
library(cluster)

# Set the seed for reproducibility
set.seed(1234)

# Calculate the gap statistic for a range of cluster sizes (from 1 to 10 in this case)
# using the k-means algorithm with 25 random starts per cluster size
gap_stat <- clusGap(scale_data, FUN = kmeans, nstart = 25, K.max = 10, B = 50)

# Plot the gap statistic results to determine the number of clusters
plot(gap_stat, main = "Gap Statistic to Calculate  Cluster Number")
```

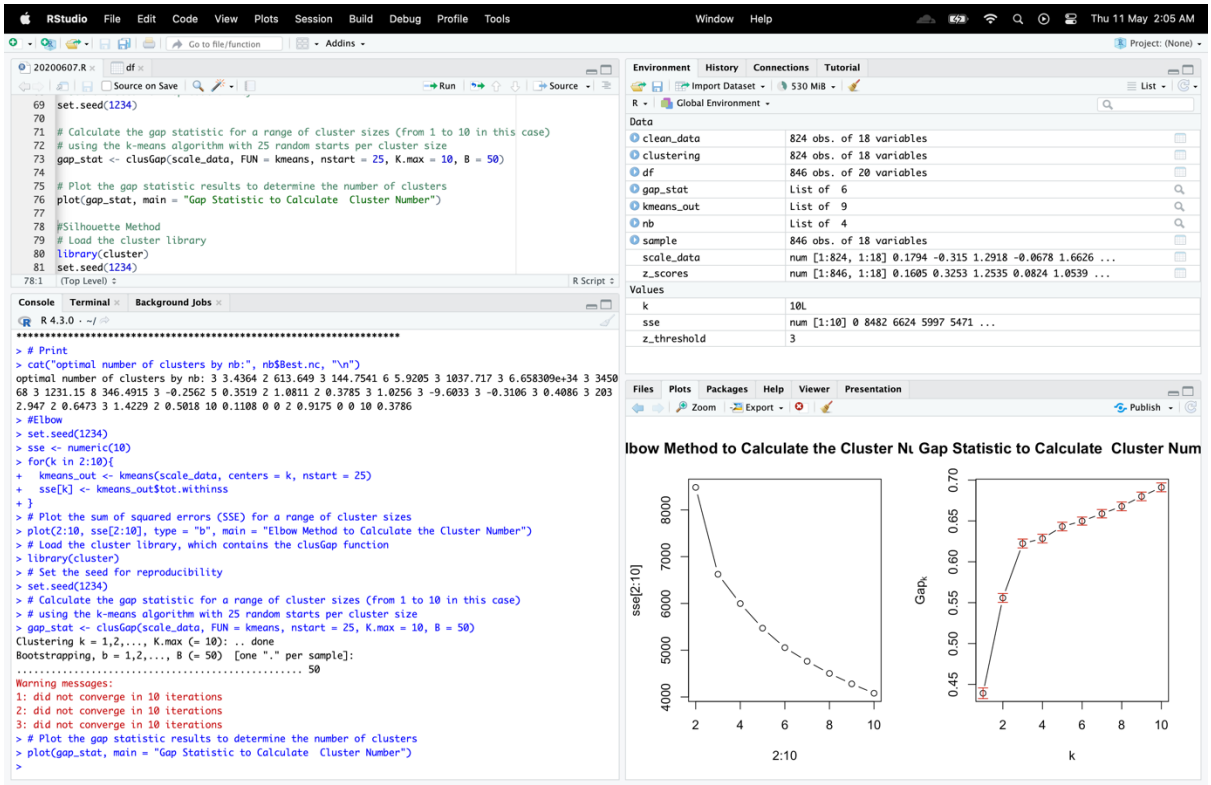

Figure 13: View Gap Statics

I used 50 bootstrap samples, 25 random starts for each k value, and a range of k values from 1 to 10 for this procedure.

When the gap statistic first reaches a maximum or seems to level off in gap static charts. For your data, this value of k is the ideal number of clusters. In this instance, three clusters are again the ideal number.

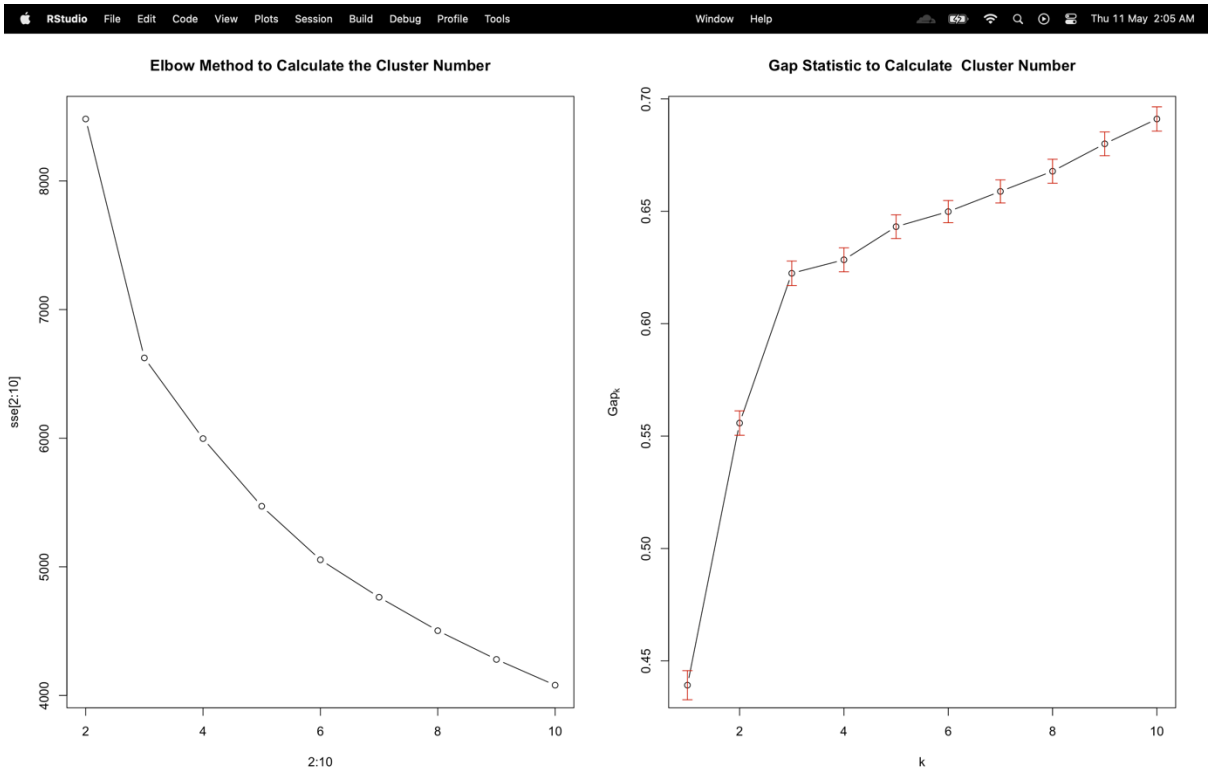

Figure 14: Gap Statics Graph

Silhoute Methods : A popular way for figuring out how many clusters should be included in a dataset is the silhouette method. Each data point's silhouette coefficient, which assesses its degree of similarity to the allocated cluster in relation to other clusters, must be calculated. The silhouette coefficient has a range of values between -1 and 1, with values near to 1 indicating a well-clustered data point, close to 0 indicating a data point on the edge of two clusters, and close to -1 indicating a misclassified data point.

Code

```
#Silhouette Method
# Load the cluster library
library(cluster)
set.seed(1234)

# Create an empty vector to store the average silhouette widths
sil_width <- numeric(10)

#calculate the average silhouette width
for(k in 2:10){
  kmeans_out <- kmeans(scale_data, centers = k, nstart = 25)
  sil_obj <- silhouette(kmeans_out$cluster, dist(scale_data))
  sil_width[k] <- mean(sil_obj[,3])
}

# Plot the average silhouette width for each cluster size
```

plot(2:10, sil_width[2:10], type = "b", main = "Silhouette Method to Determine Cluster Number")

According to this method, the average silhouette width for each value of k, which ranges from 2 to 10, is plotted after the silhouette widths for each cluster have been calculated. Inferring the ideal number of clusters based on the largest silhouette width is possible using the resulting plot.

The ideal k value should be found by searching for the k value with the largest average silhouette width. So, based on the results I received, two clusters will work better for this dataset when using the silhouette method.

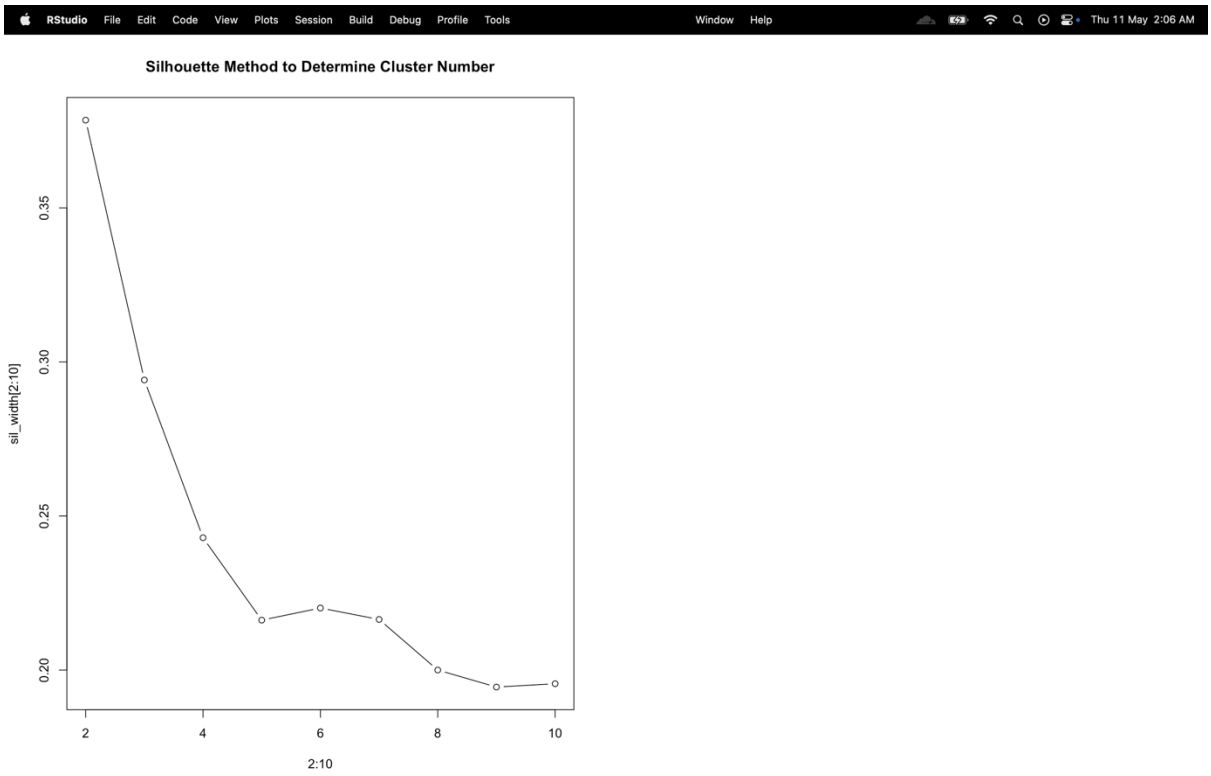

Figure 15: Silhoutte Method Graph

Finding the ideal number of clusters in a dataset is easy and obvious using the elbow approach. When plotting the within cluster sum of squares (WSS) versus cluster count,the "elbow" point which denotes the point at which larger clusters no longer significantly increase clustering quality—is sought after. The elbow approach frequently provides a fair estimate of the ideal number of clusters with little computational effort and straightforward interpretation, but other methods like nbclust, gap statistic, and silhouette analysis may provide more advanced assessments of cluster quality. To choose the precise number of clusters, it is still crucial to take into account the outcomes of other approaches and to rely on professional judgment. The gap static technique and nb clust method both indicated that three clusters are the ideal number, thus I choose to use that as the value of k.

3. **The next step is the kmeans clustering investigation. Using, again, all input variables, perform a kmeans analysis using the most favoured "k" from those "automated" methods. For this k-means attempt, show the related R-based kmeans output, including information for the centres, clustered results, as well as the ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_Squares (TSS). It is also important to calculate/illustrate the BSS and the within_cluster_sums_of_squares (WSS) indices (internal evaluation metrics).**

   Code

```
#Assuming k is 3
set.seed(1234)
# The resulting cluster assignments are stored in the kmeans_out object
kmeans_out <- kmeans(scale_data, centers = 3, nstart = 25)

# Cluster centers
centers <- kmeans_out$centers
# Print results
print(centers)

# Cluster assignments
clusters <- kmeans_out$cluster
# Print results
cat("Cluster centers:\n")
print(clusters)

# TSS
TSS <- sum(rowSums(scale_data)^2)
# Print results
cat("\nTotal sum of squares (TSS):", TSS, "\n")

# BSS
BSS <- sum(kmeans_out$withinss)
# Print results
cat("\nBetween-cluster sum of squares (BSS):", BSS, "\n")

# Ratio
bss_tss__ratio <- BSS / TSS
# Print results
cat("\nRatio between BSS & TSS:", bss_tss__ratio, "\n")

# WSS
WSS <- sum(kmeans_out$withinss)
# Print results
cat("\nWithin-cluster sum of squares (WSS):", WSS, "\n")

# WSS Plot function
# generates a plot of within-groups sum of squares (WSS) for different numbers of clusters
```

```
wss_plot <- function(scale_data, nc=15, seed=1234)
  {
   wss <- (nrow(scale_data)-1)*sum(apply(scale_data,2,var))
   for (i in 2:nc) {
    set.seed(seed)
    wss[i] <- sum(kmeans(scale_data, centers = i)$withinss)
   }
   # plot of the WSS for each number of clusters
   plot(1:nc, wss, type="b", xlab="Number of clusters",
   ylab="within groups sum of squares")
}

wss_plot(scale_data)
KM = kmeans(scale_data,3) # KM object

# Install the package
#install.packages("ggfortify")

# Load the package
library(ggfortify)
library(factoextra)

# Method 1
autoplot(KM,scale_data,frame=TRUE)

# Method 2
fviz_cluster(kmeans_out, data = scale_data,
          palette = c("#B4EEB4", "#0FF7F0", "#E34800", "#bB00e7"),
          geom = "point", ellipse.type = "convex", ggtheme = theme_bw() )
```

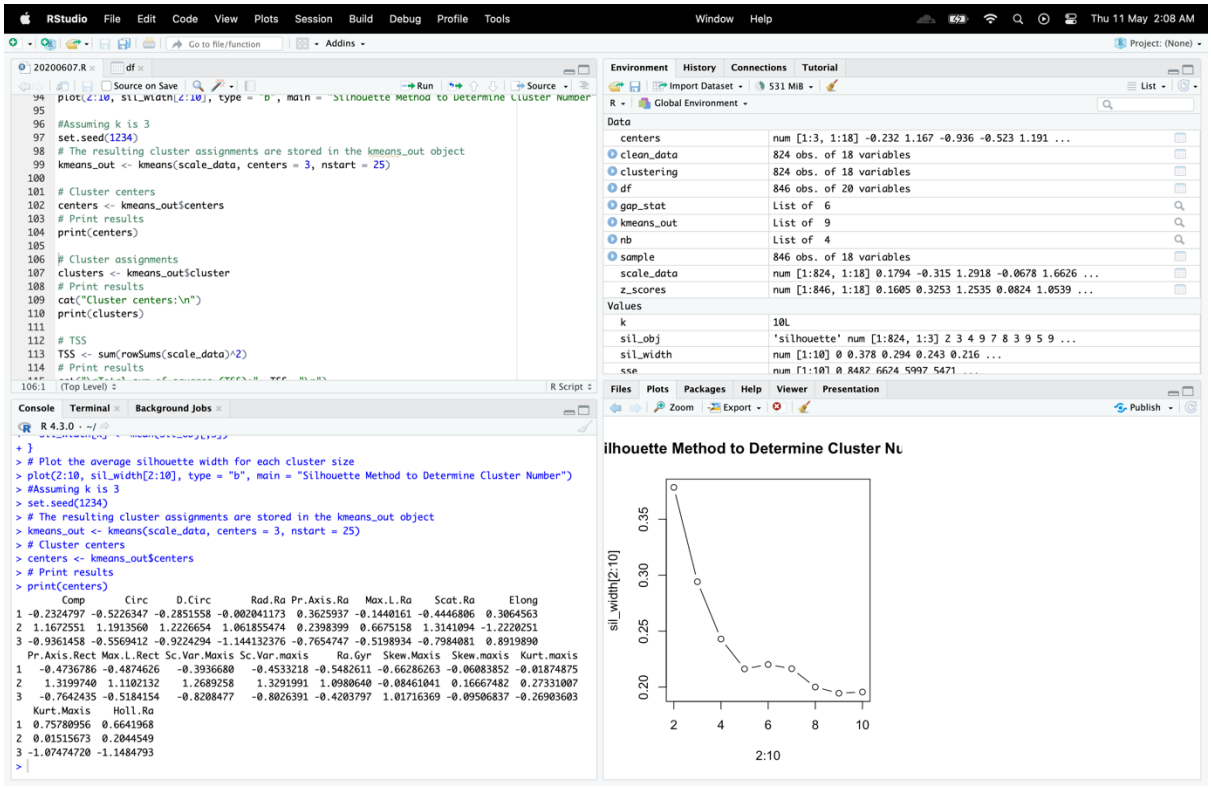

Figure 16: Center of Clusters

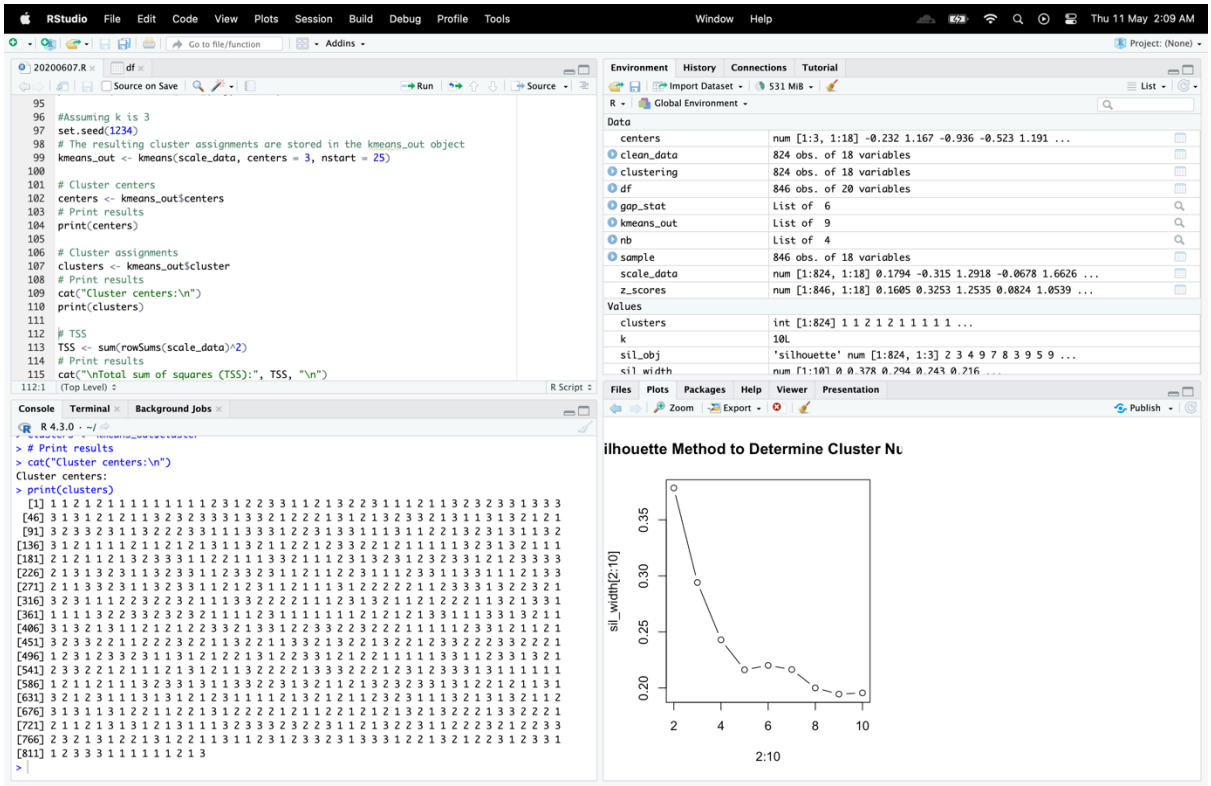

Figure 17: Print of Clusters

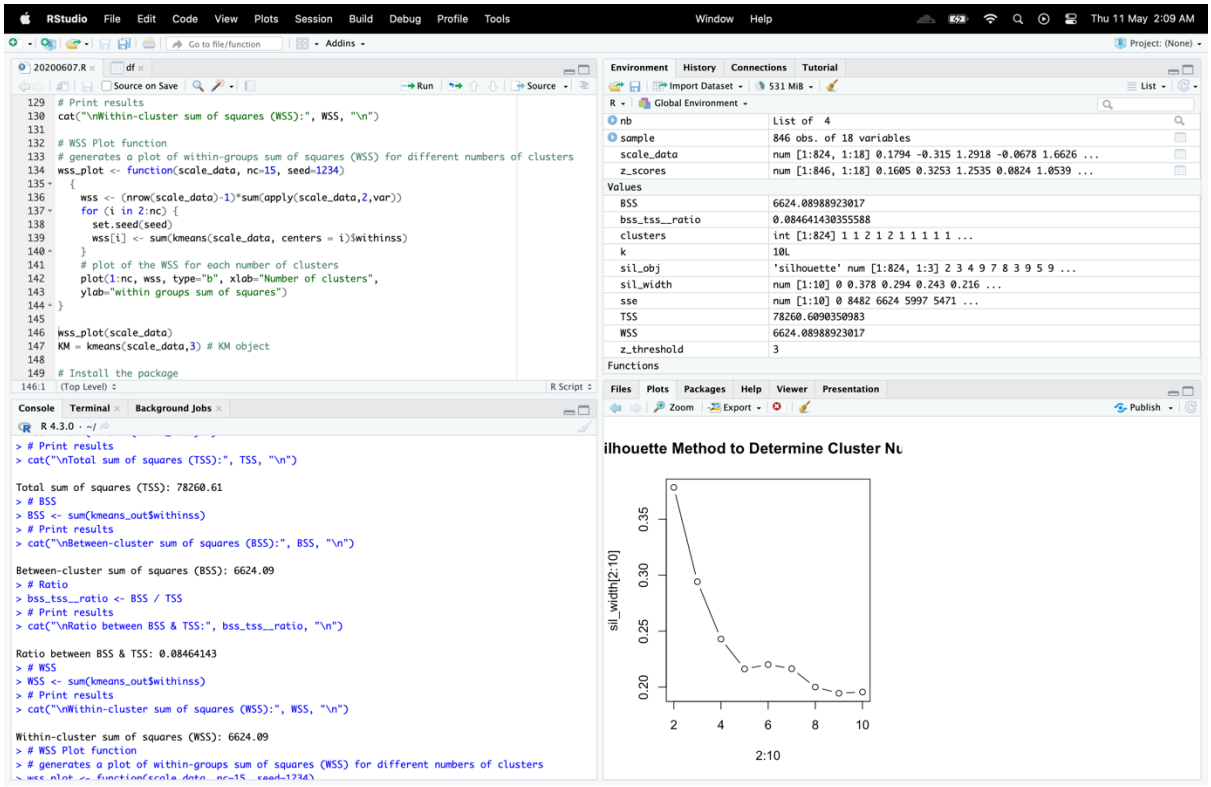

Figure 18: TSS, BSS and their Ratio

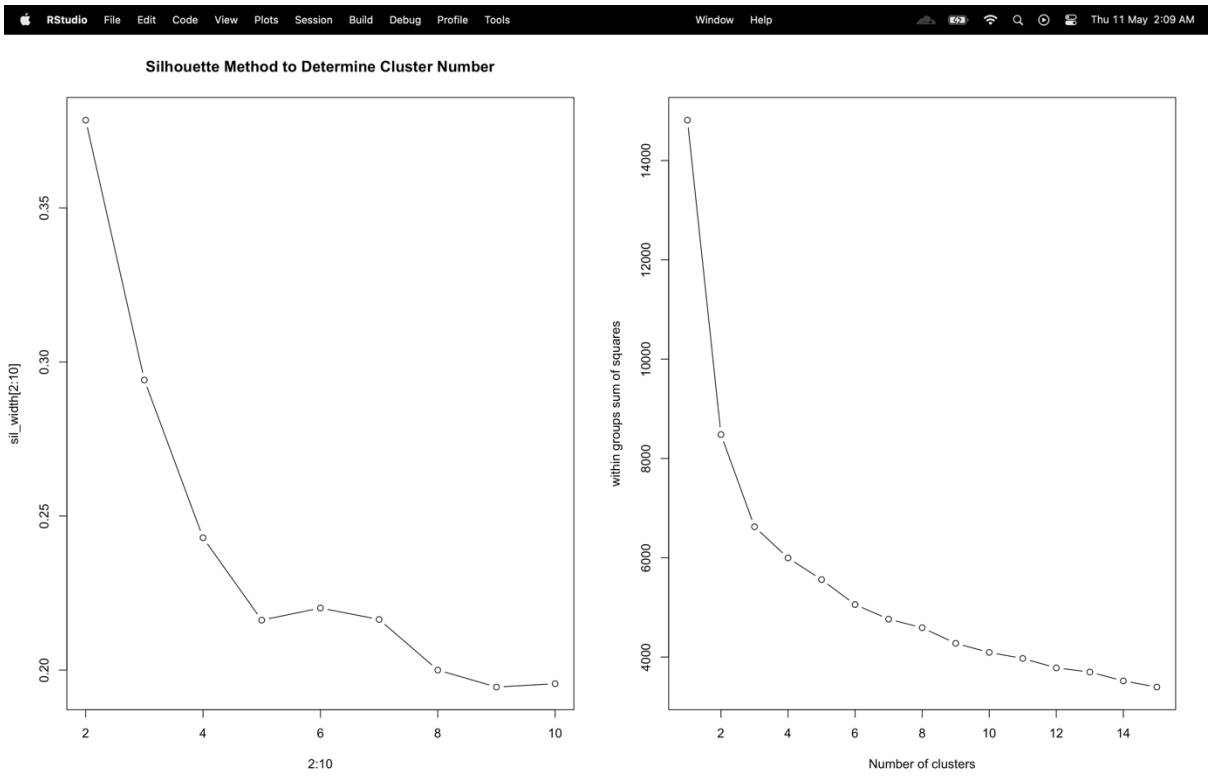

Figure 19: WSS Graph

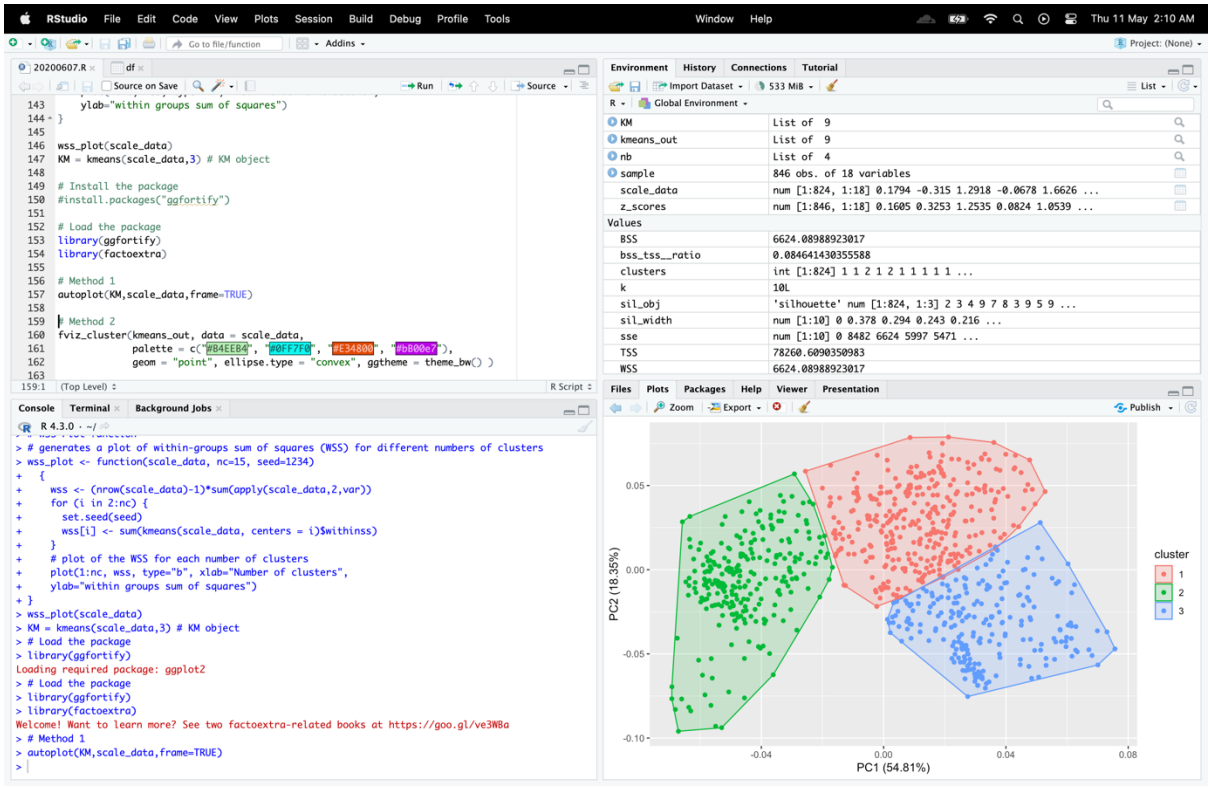

Figure 20: Kmeans cluster

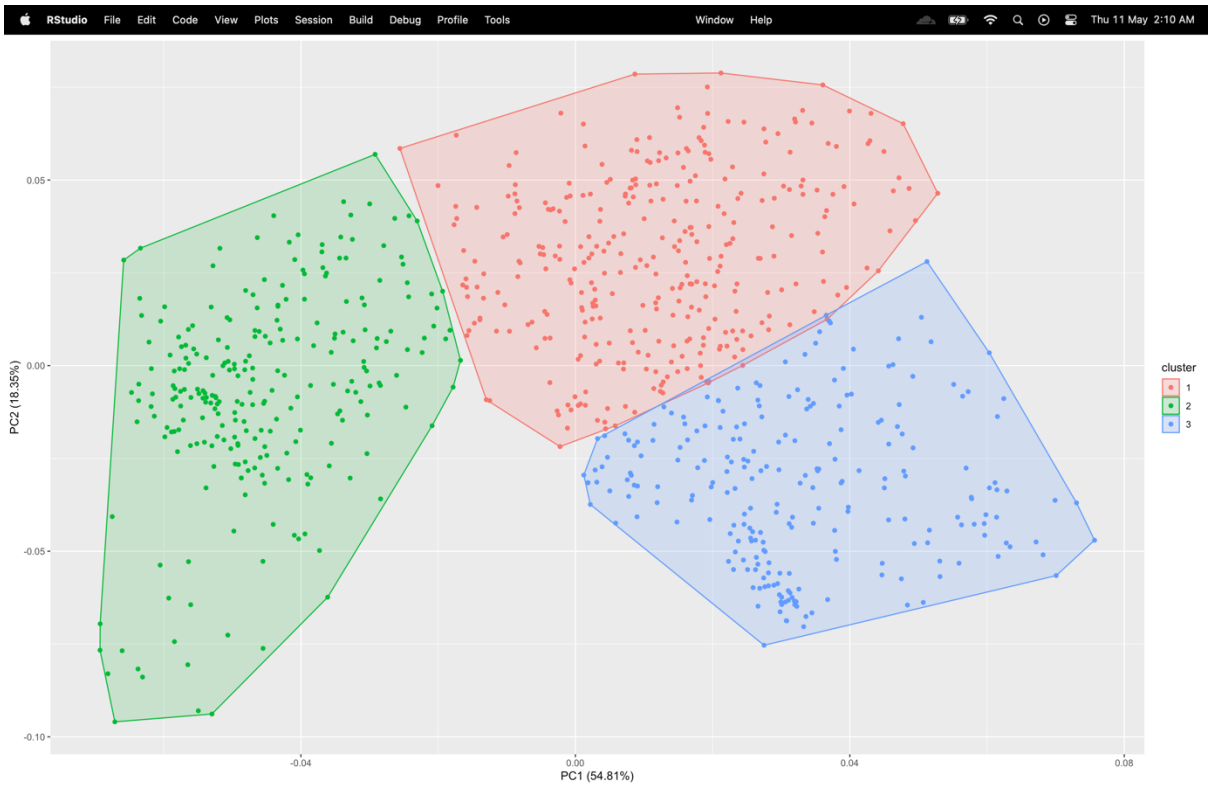

Figure 21: Kmeans graph

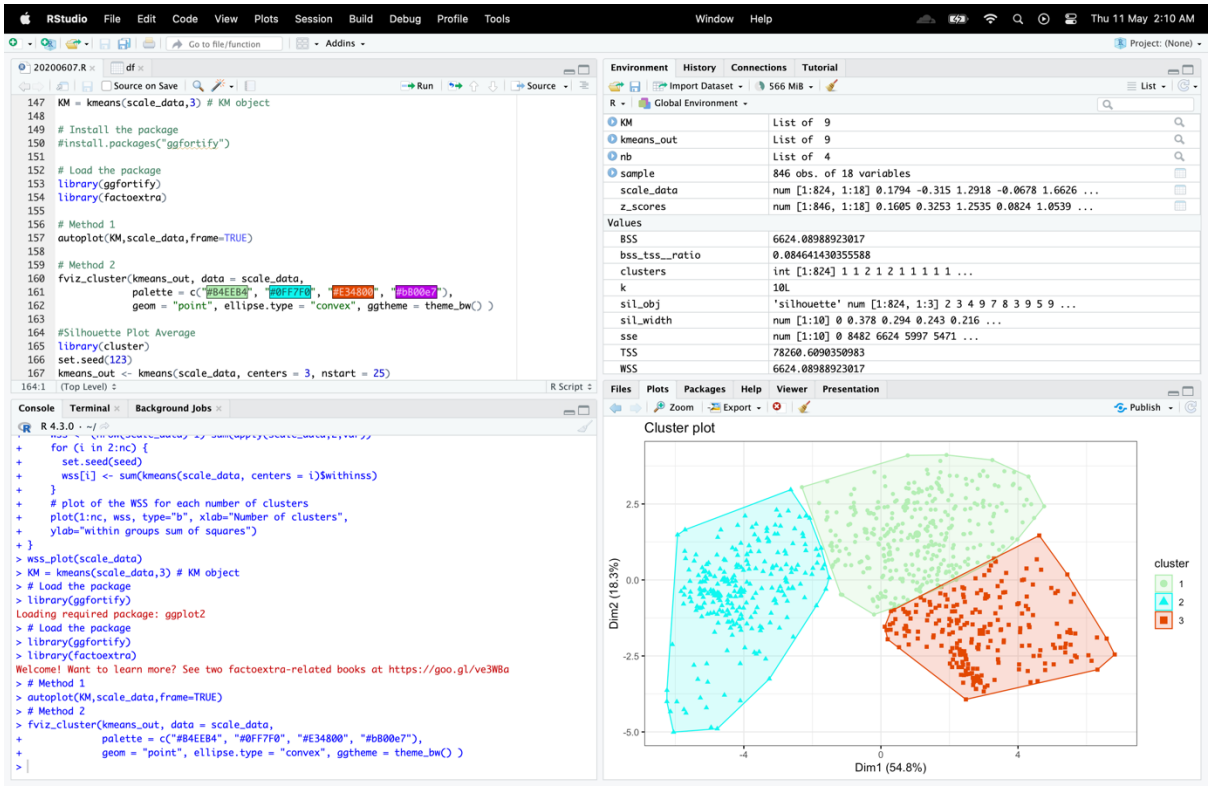

Figure 22: Kmeans M2

Figure 23 : K Means M2 Graph

4. **Following the kmeans attempt, provide the silhouette plot (another internal evaluation metric) which displays how close each point in one cluster is to points in the neighbouring clusters. Provide the average silhouette width score and your discussion on this plot, which should include your comments on the level of "quality" of the obtained clusters.**

Code

```
#Silhouette Plot Average
library(cluster)
set.seed(123)
kmeans_out <- kmeans(scale_data, centers = 3, nstart = 25)

library(factoextra)
# Calculate average silhouette width
sil_obj <- silhouette(kmeans_out$cluster, dist(scale_data))
sil_avg <- mean(sil_obj[,3])

# Plot silhouette plot with color palette
fviz_silhouette(sil_obj, palette = "Set2", main = paste0("Silhouette Plot (Avg. Silhouette Width: ", round(sil_avg, 2), ")"))
```

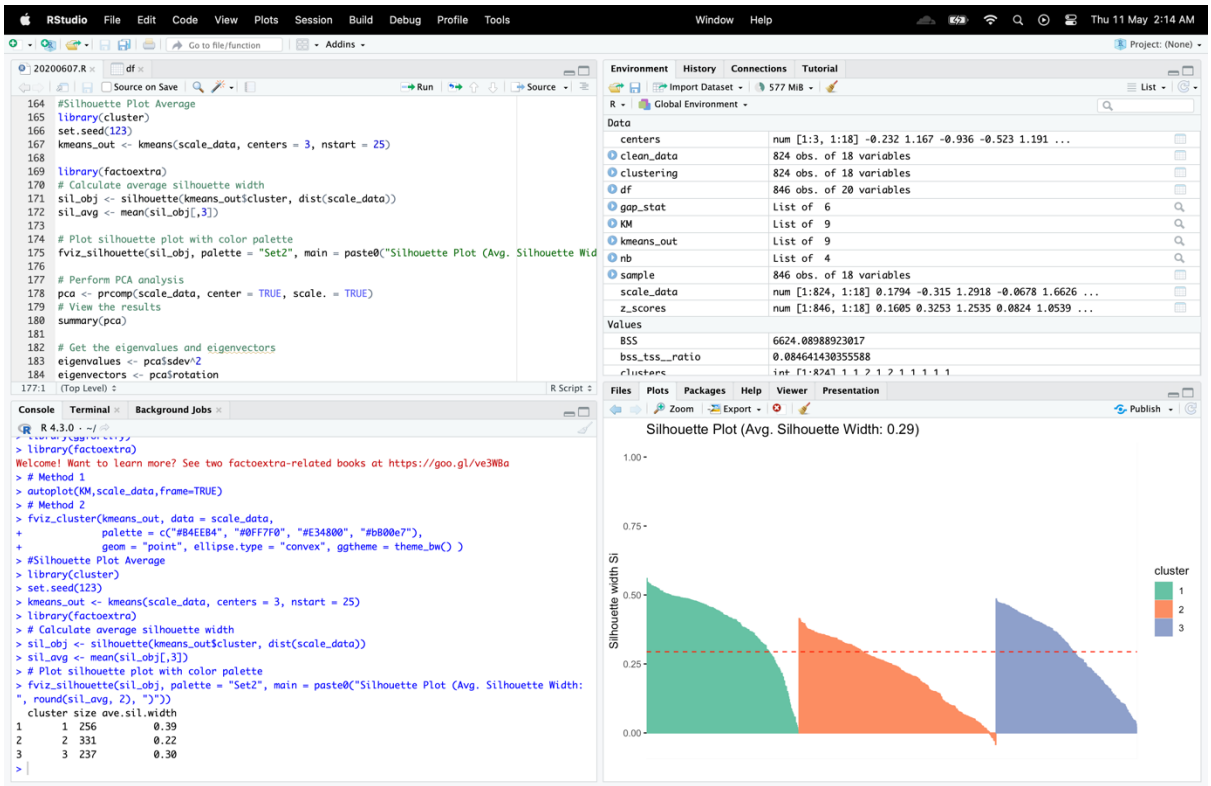

Figure 24: Silhoute Plot

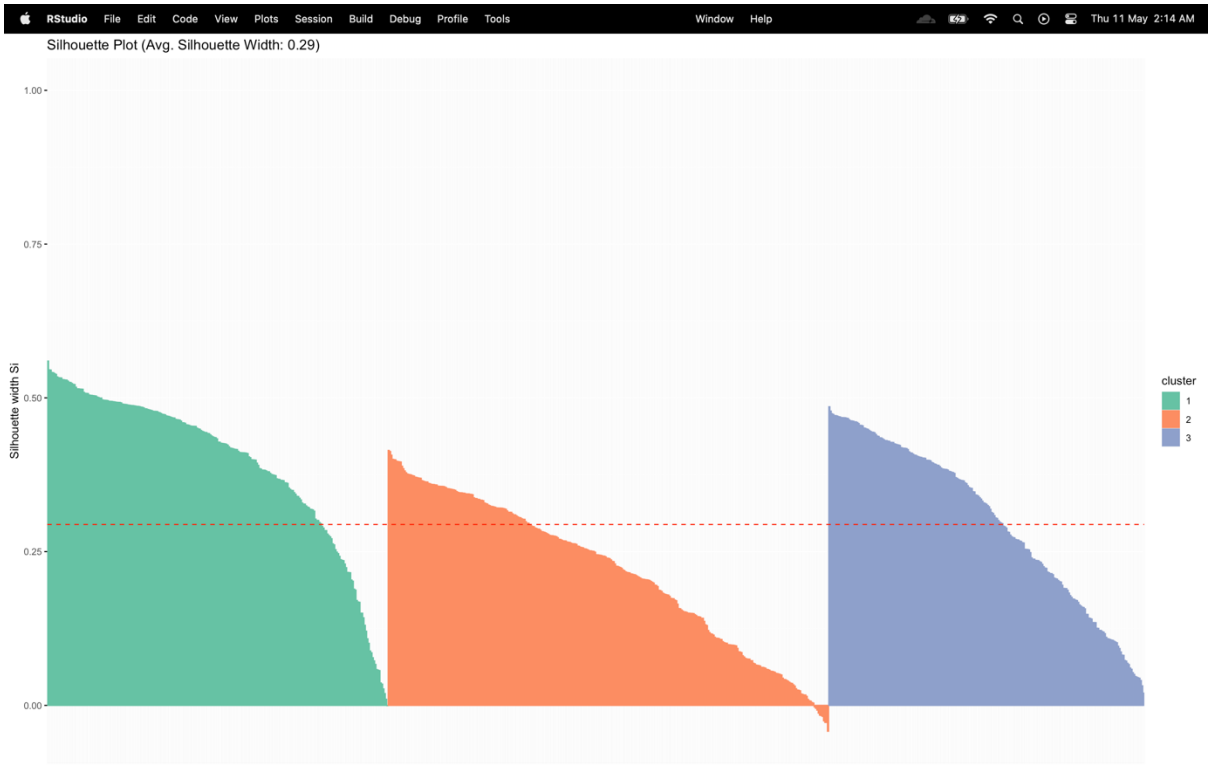

Figure 25: Silhoutte PPlot Graph

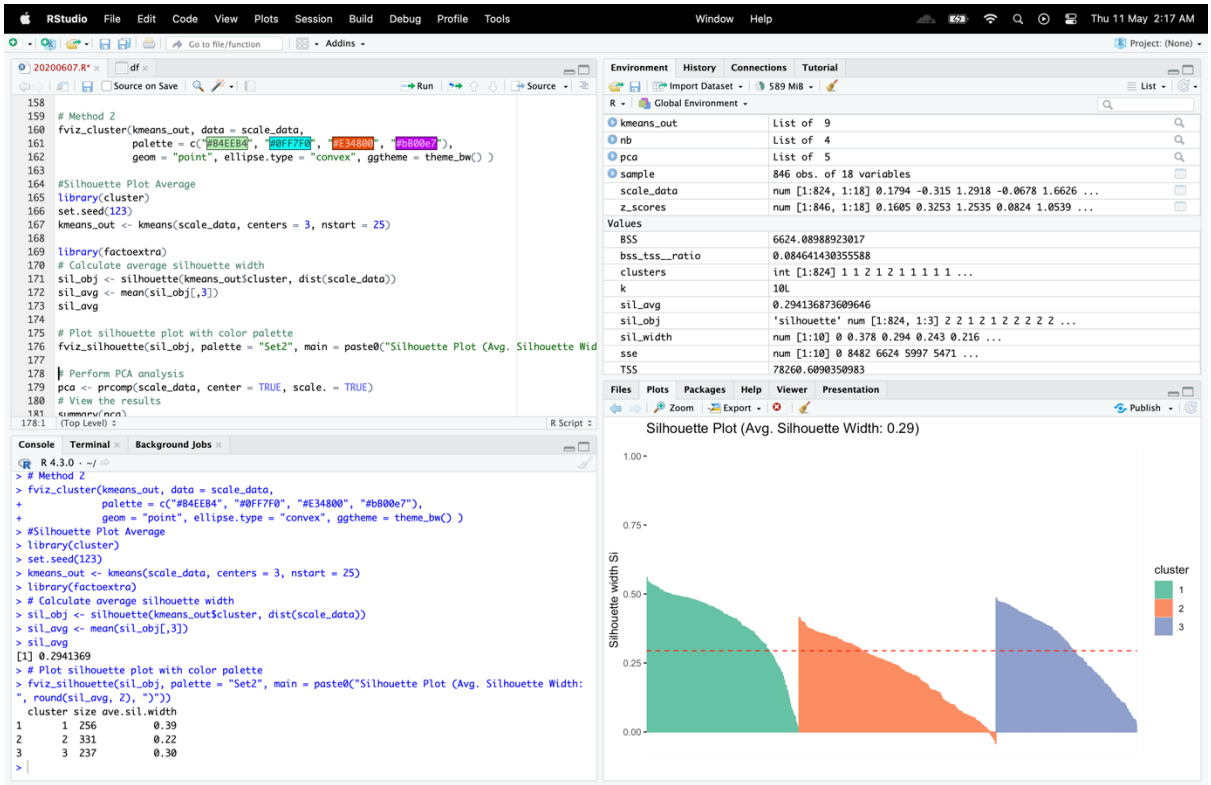

Figure 26: Sihoutte Average

A approach used to assess how effectively a clustering algorithm has done is called the Silhouette method. It measures the degree to which an observation is comparable to the observations in its own cluster as opposed to observations in other clusters. The level of cluster separation and compactness can be determined using this metric. The average silhouette width was determined in order to evaluate how well the clustering method handled the dataset. Based on the distance between each observation and the other observations in both its own cluster and other clusters, the silhouette width for each observation was computed. Next, the average silhouette width for all data was calculated, which gave a general indicator of cluster quality.

The resulting silhouette plot, where the height of each bar represents the silhouette width for that observation and the color denotes the appropriate cluster, shows the silhouette width for each observation, grouped by cluster. The average silhouette width is stated in the plot's major title. A higher average silhouette width indicates that the clustering method has created well-defined clusters with good separation and compactness. A smaller average silhouette width, on the other hand, suggests that the clustering algorithm has produced poorly separated clusters or clusters containing inappropriate observations.

In general, a reliable way for assessing the caliber of the clusters generated by the algorithm was offered by the Silhouette method in conjunction with K-means clustering. It was possible to identify any clusters that had 12 ill-suited observations by looking at the resulting Silhouette plot, which also allowed one to gauge the clusters' amount of distance and compactness. This data can be utilized to improve the clustering method or change the number of clusters used to more accurately represent the underlying structure of the data.

width of a typical silhouette: 0.294

2nd Subtask Objectives:

5. **As this is a typical multi-dimensional, in terms of features problem, you need also to apply the PCA method to this vehicle dataset. You need to show all R-outputs related to PCA analysis, including eigenvalues/eigenvectors, cumulative score per principal components (PC). Create a new "transformed" dataset with principal components as attributes. Choose those PCs that provide at least cumulative score > 92%. Provide a brief discussion for your choice to choose specific number of PCs.**

   PCA (Principal Component Analysis): A statistical method for dimensionality reduction of high-dimensional data is PCA (Principal Component Analysis). It functions by locating the underlying patterns in the data and changing the coordinate system so that the data points are represented by fewer variables known as main components.

   Reducing the number of dimensions in the data while retaining as much variability as possible is the aim of PCA. This is accomplished by identifying the primary components, often known as the directions in which the data vary most widely. The direction of the largest variability is represented by the first principal component, the direction of the second highest variability is represented by the second principal component, and so on.

   Machine learning, data mining, signal processing, and image analysis are just a few of the areas where PCA is extensively employed. It can be applied to noise reduction, data compression, feature extraction, and visualization. In addition, it is employed as a pre-processing stage before other statistical methods like clustering and classification.

   Code

```
# Perform PCA analysis
pca <- prcomp(scale_data, center = TRUE, scale. = TRUE)
# View the results
summary(pca)

# Get the eigenvalues and eigenvectors
eigenvalues <- pca$sdev^2
eigenvectors <- pca$rotation
# Calculate the cumulative score per principal component
per_var <- eigenvalues/sum(eigenvalues)
cumu_per_var <- cumsum(per_var)

# Print the cumulative proportion of variance explained by each principal component
cat("Cumulative proportion of variance explained by each PC:\n")
print(cumu_per_var)
# Create a new dataset with principal components as attributes
pca_data <- as.data.frame(pca$x)
# Choose PCs that provide at least cumulative score > 92%
chosed_pcs <- which(cumu_per_var > 0.92)
# Print the chosen PCs
cat("Chosen PCs:", chosed_pcs, "\n")
```

```
# Subset the transformed dataset to include only the chosen PCs
transform_data <- pca_data[, chosed_pcs]
```

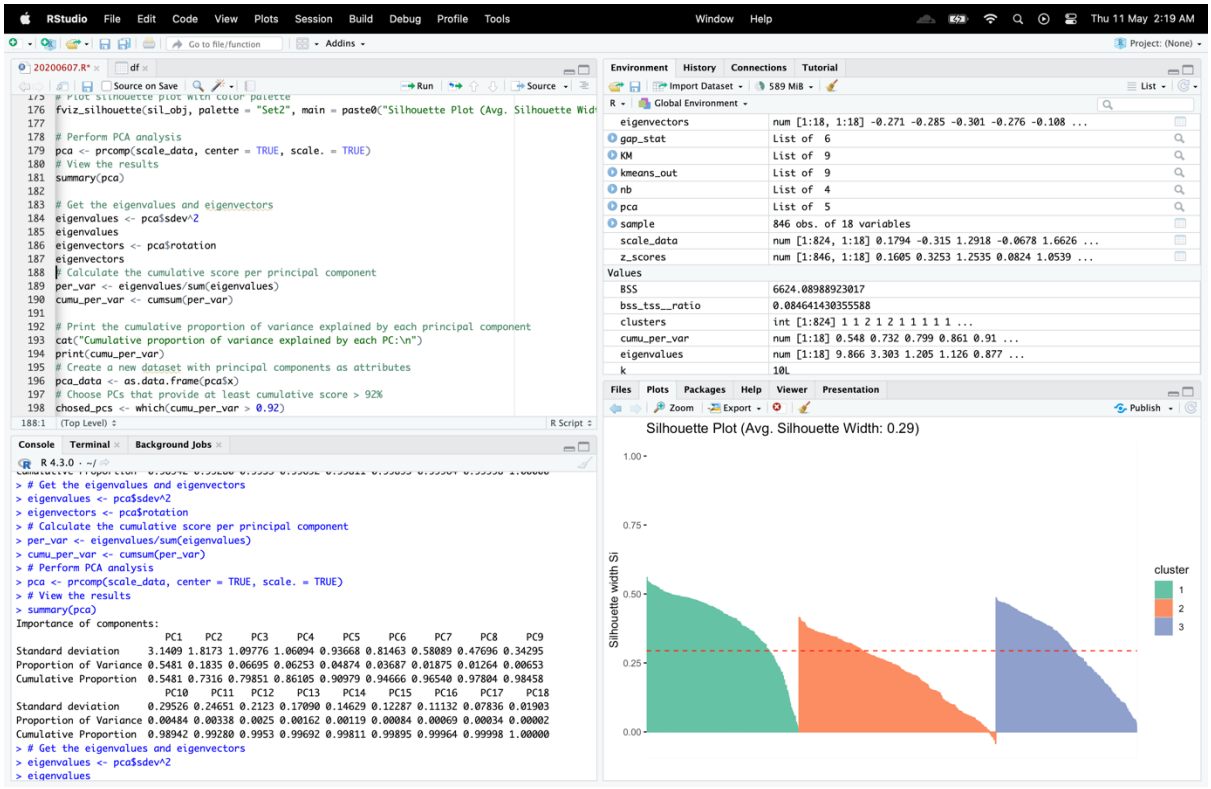

Figure 27: PCA ViewFigure 28: Eigenvalues

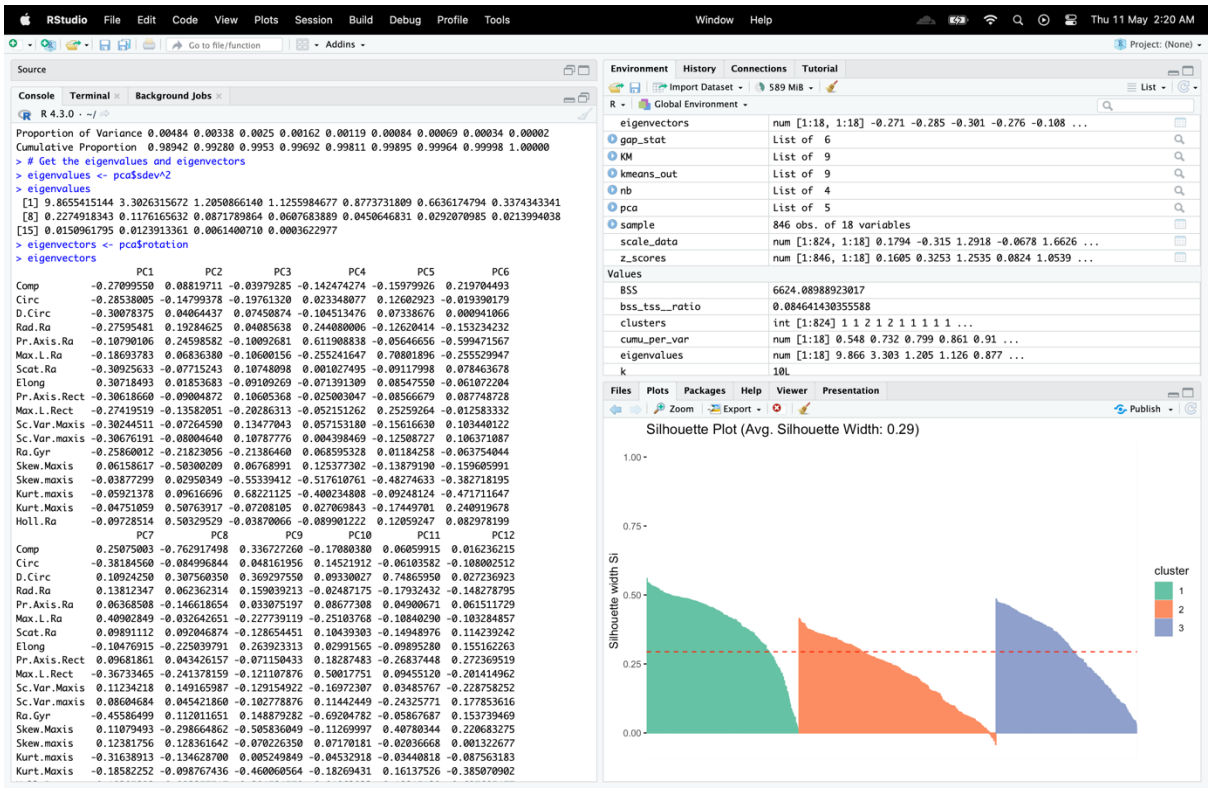

Figure 29: Eigen vectors

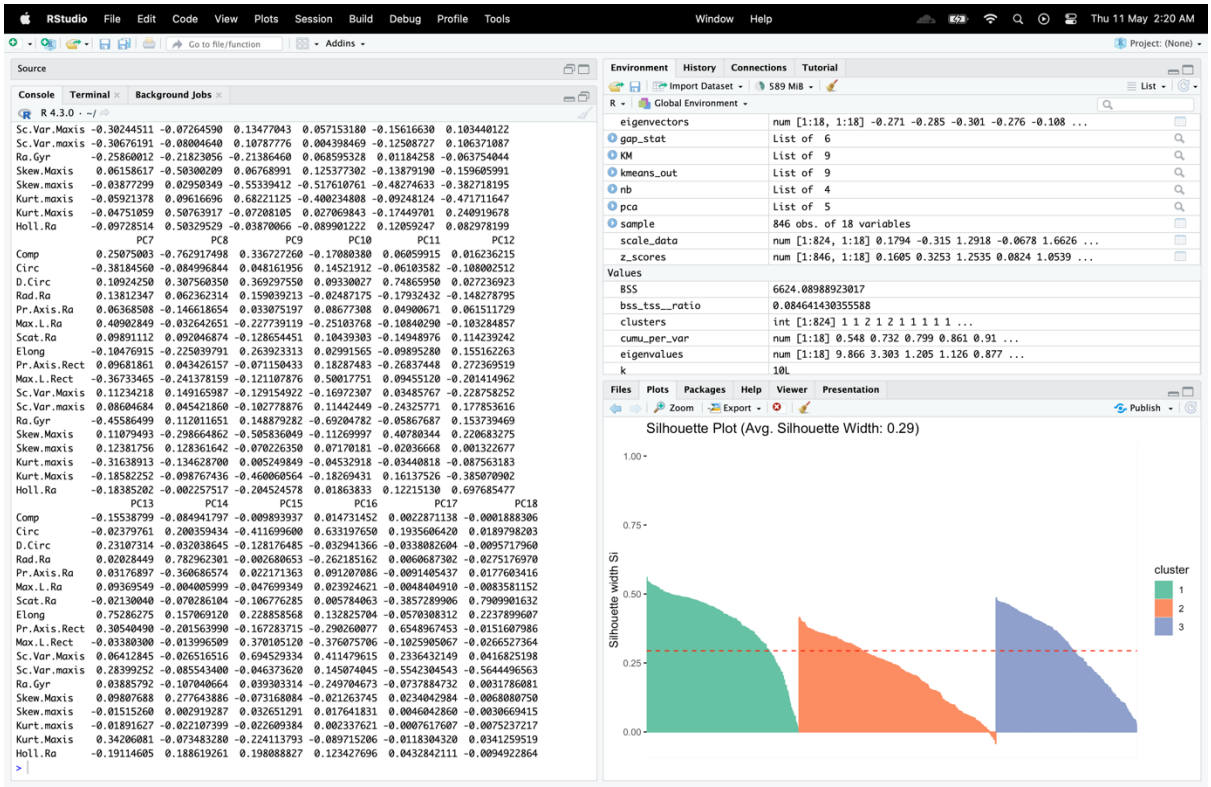

Figure 30; Eigen vectors

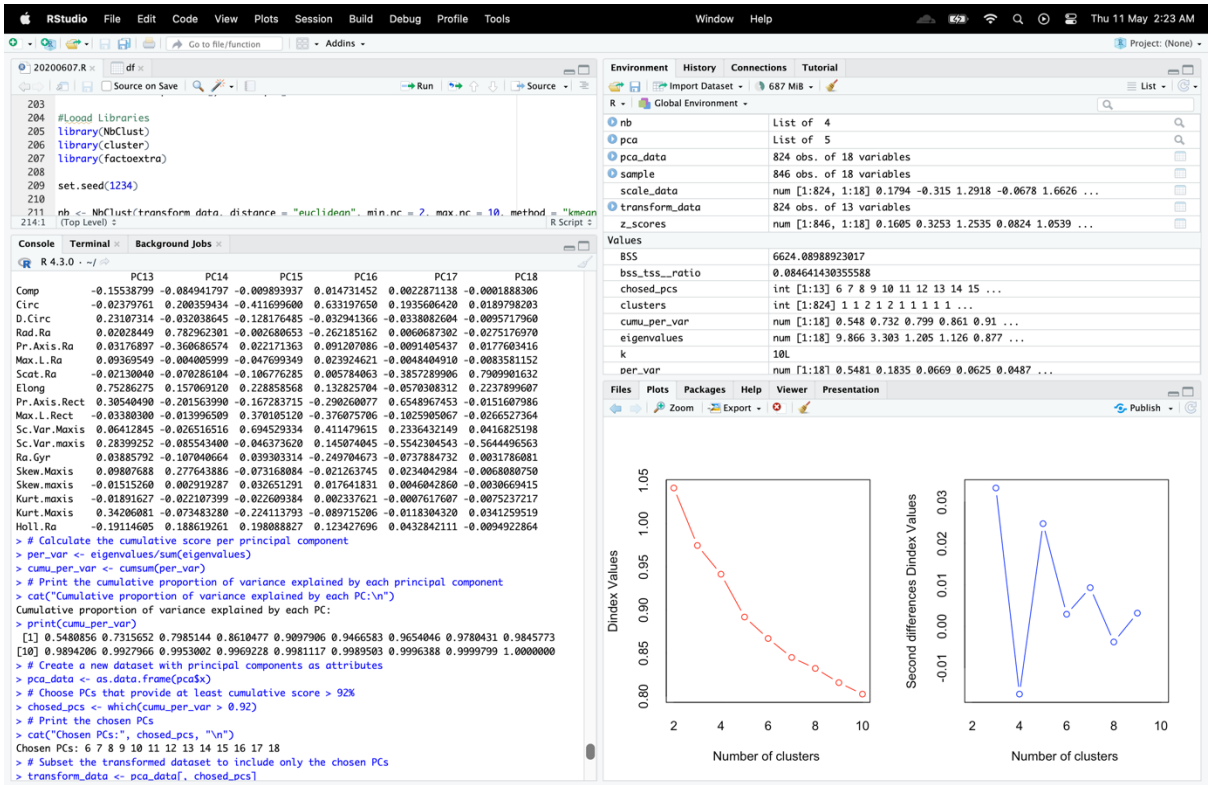

Figure 31: Cumulative and Chosen PCs

Using the PCA technique, many correlated variables is reduced to a smaller set of uncorrelated variables. It aids in finding patterns and connections between the variables in the dataset. The PCA analysis on the 'scale_data' dataset is carried out using the 'prcomp' function as mentioned previously. The 'pca' variable contains the results of the PCA analysis. The 'center = TRUE' and 'scale. = TRUE' parameters state that the data should be scaled and cantered before analysis. Then, some descriptive statistics for the PCA analysis are displayed using the'summary' function.

The 'eigenvalues' and 'eigenvectors' variables, respectively, contain the retrieved eigenvalues and eigenvectors from the PCA analysis. The coefficients that define how each variable contributes to the principal components are represented by the eigenvectors, whilst the eigenvalues represent the variance of each principal component. The “cumu_per_var” variable contains the cumulative scores for each principal component. These ratings show how much of the total variance is accounted for by each major component.
The primary components that account for at least 92% of the final score are then chosen by the code.

This is accomplished by locating the principal component 14 indexes and storing them in the 'chosed_pcs' variable if their cumulative scores are greater than 0.92.
The final step is to construct a transformed dataset by adding the selected principal components as attributes to a fresh data frame and saving it in the 'transform_data' variable. You can use this converted dataset for more analysis or visualization.

6. **In reality, as we have practically a new dataset, we need to find an appropriate k for our “new” kmeans clustering attempt. Like previously, apply the same four “automated” tools to this new pca-based dataset. You need to provide, in your report, the related R-outputs and your discussion on these “new” outcomes.**

Code

```
#Looad Libraries
library(NbClust)
library(cluster)
library(factoextra)

set.seed(1234)

nb <- NbClust(transform_data, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
cat("optimal number of clusters by nb:", nb$Best.nc, "\n")
```

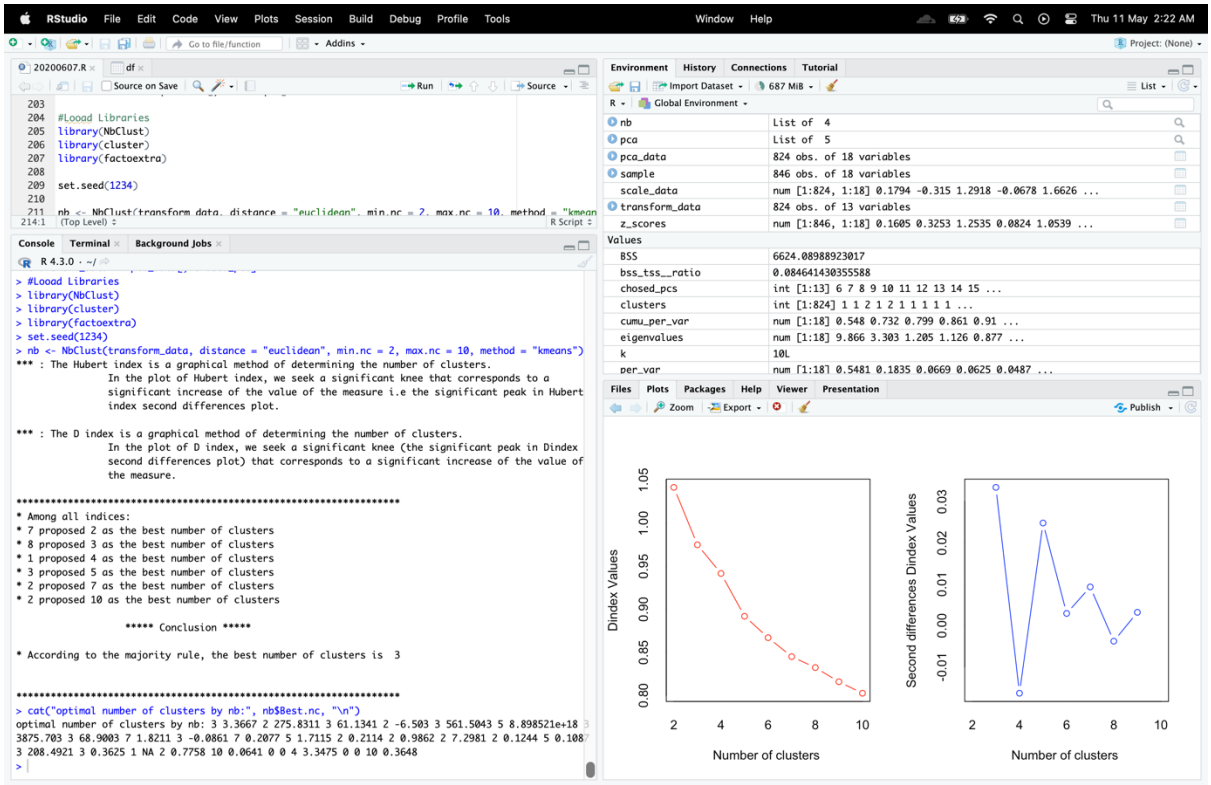

Figure 32: New Kmeans

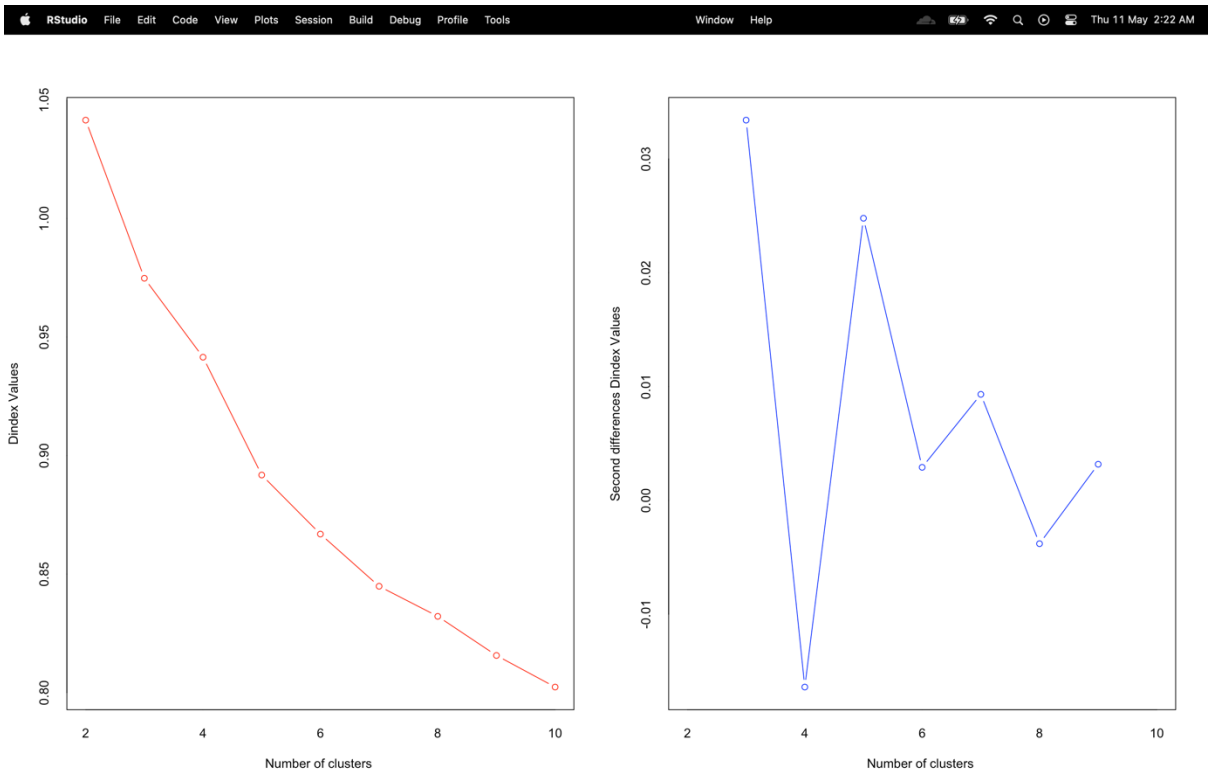

Figure 33: New Kmeans Graph

Code

```
# Elbow Method
fviz_nbclust(transform_data, kmeans, method = "wss")
```

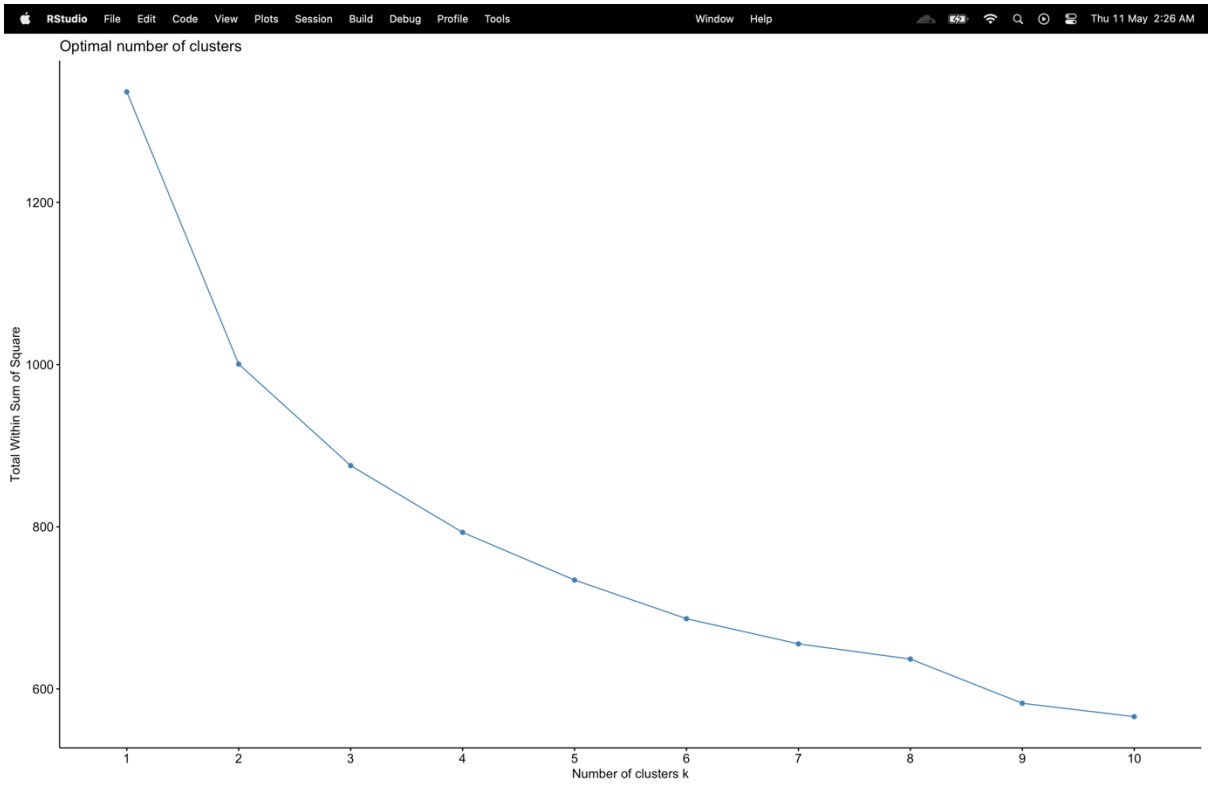

Figure 34: New Elbow

Code

```
# Run gap statistic method
# Method 1
fviz_nbclust(transform_data, kmeans, method = 'gap_stat')
```

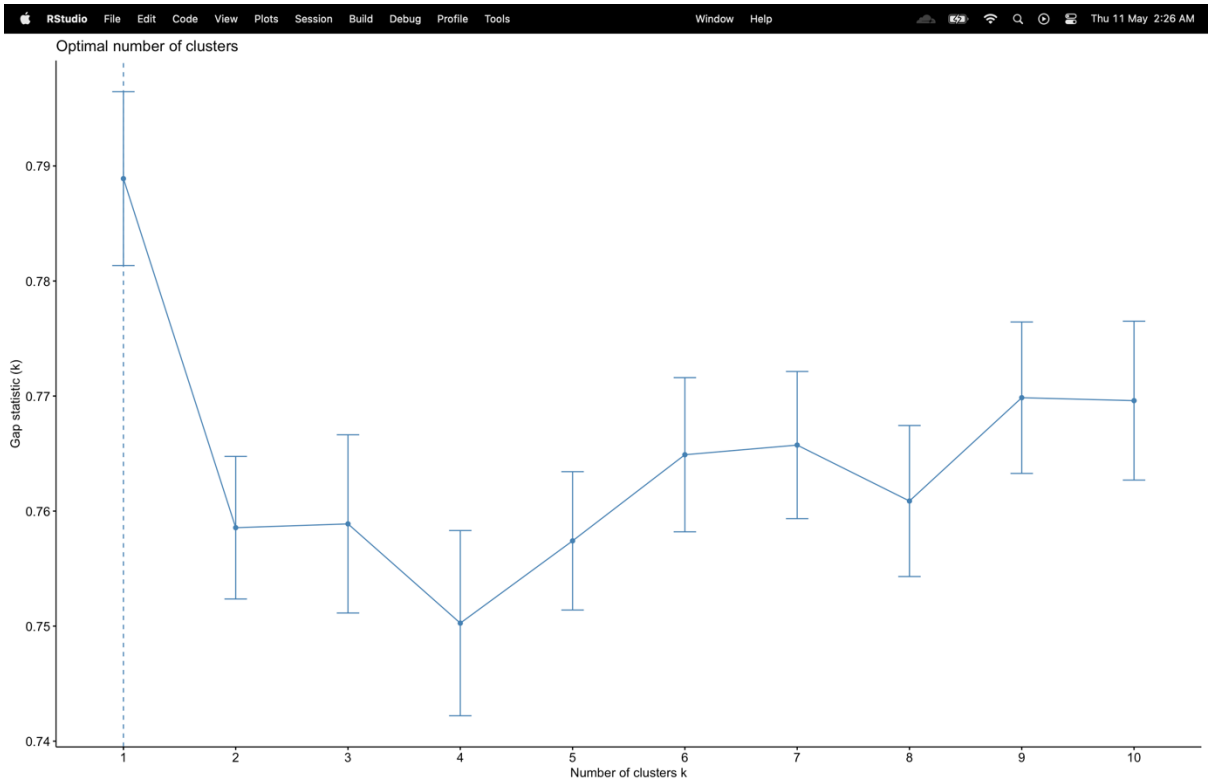

Figure 35New Gap Statics

determined that k value was equal to 2.

```
# Method 2
gap_stat <- clusGap(transform_data, FUN = kmeans, nstart = 25,
                    K.max = 10, B = 50)
fviz_gap_stat(gap_stat)
```

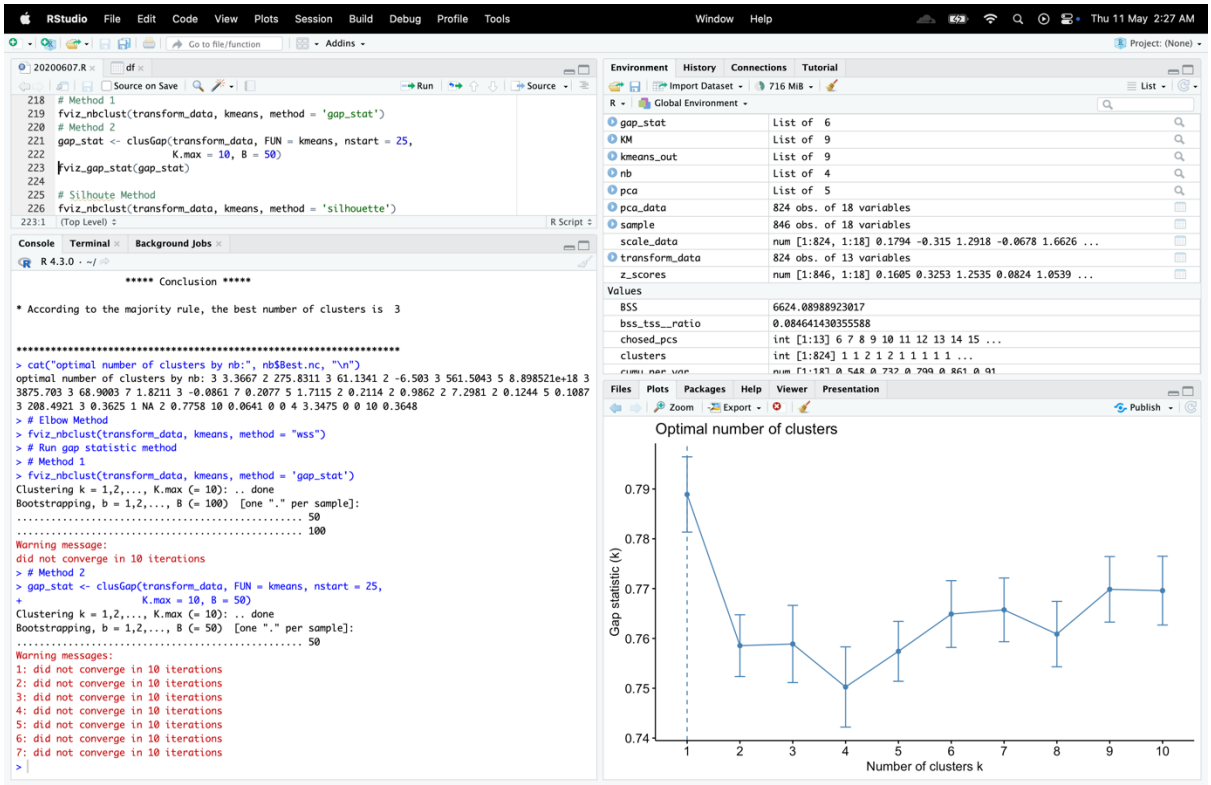

Figure 36: New Gap Statics M2

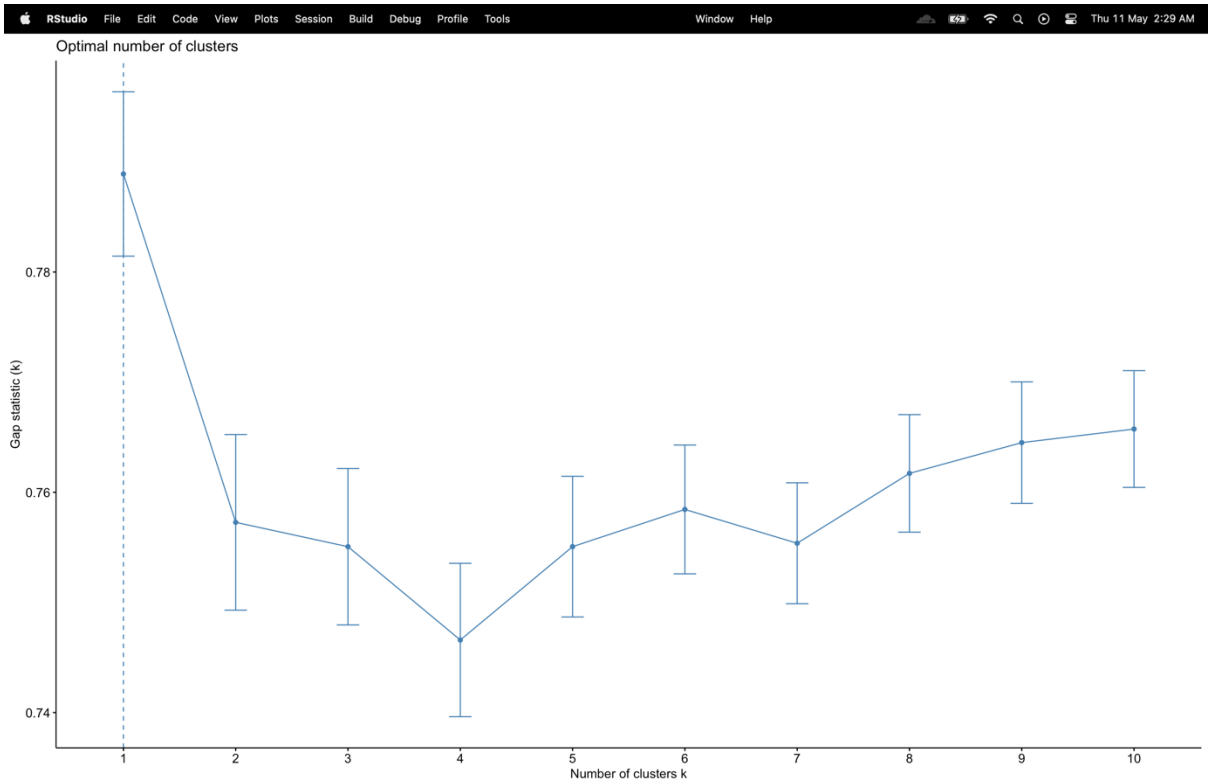

Figure 37: New Gap Statics M2 Graph

Code

```
# Silhoute Method
fviz_nbclust(transform_data, kmeans, method = 'silhouette')
```

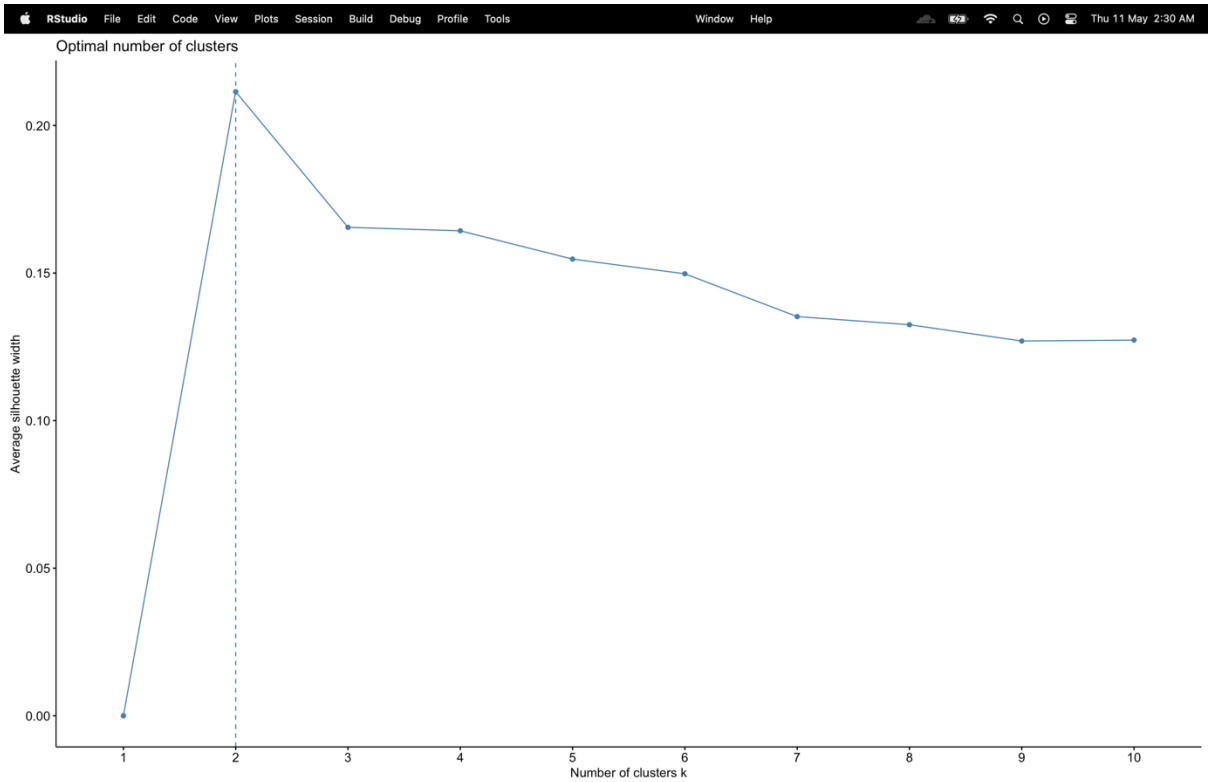

Figure 38: New k=2 Nb clust Silhoutte

determined that k value was equal to 2.

7. **Using this new pca-dataset, perform a kmeans analysis using the most favoured k from those "automated" methods. For this k-means attempt, show the related R-based kmeans output, including information for the centres, clustered results, as well as the ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_Squares (TSS). It is also important to calculate/illustrate the BSS and the within_cluster_sums_of_squares (WSS) indices (internal evaluation metrics).**

   New cluster number,
   After PCA, the elbow technique determined that k value was equal to 2. The factoextra and nbclust libraries' features made it simple to complete the aforementioned automated procedures.

   Code

```
# Perform k-means clustering on transformed data with k=2
kmeans_output <- kmeans(transform_data, 2)

# Print the k-means output
kmeans_output
# Print the centers of each cluster
kmeans_output$centers
# Print the cluster assignments for each observation
kmeans_output$cluster

#Between_cluster_sums_of_squares (BSS)
BSS <- sum(kmeans_output$size * dist(kmeans_output$centers)^2)
# (TSS) Total_sum_of_Squares
TSS <- sum(dist(transform_data)^2)
# Calculate the ratio of BSS and TSS
BSS_TSS_ratio <- BSS / TSS
cat("BSS/TSS ratio:", BSS_TSS_ratio, "\n")
#ploting new clustures
newKM = kmeans(transform_data,2)

#install.packages("ggfortify")
# Install the package

library(ggfortify) # Load the package
autoplot(newKM,transform_data,frame=TRUE)
```

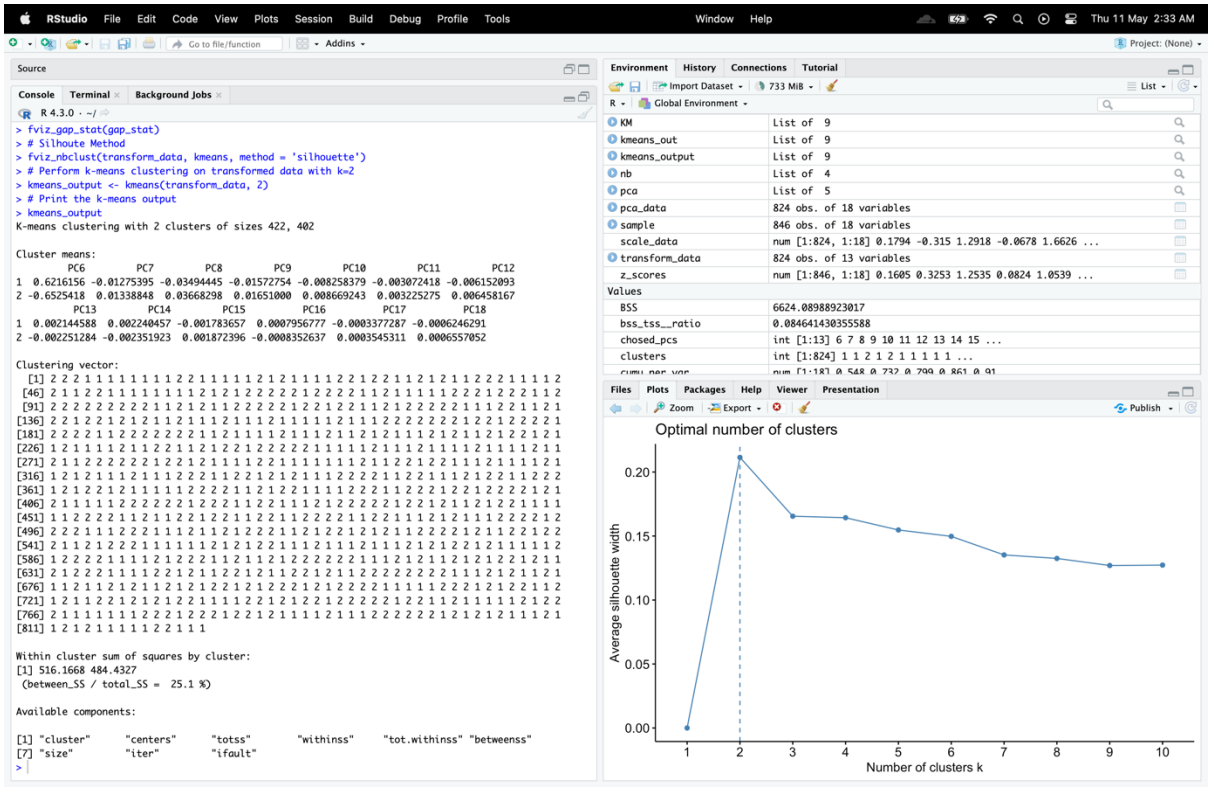

Figure 39 : k means output centres

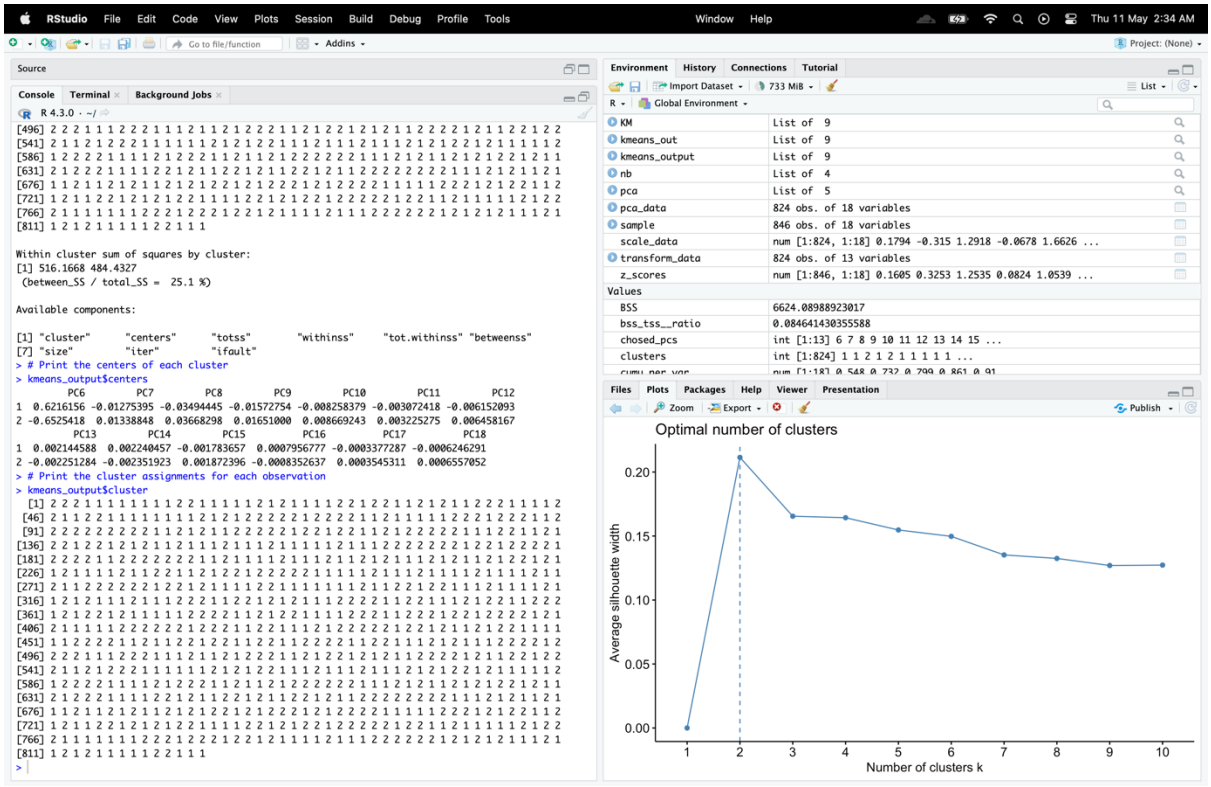

Figure 40: Cluster Assignment for each

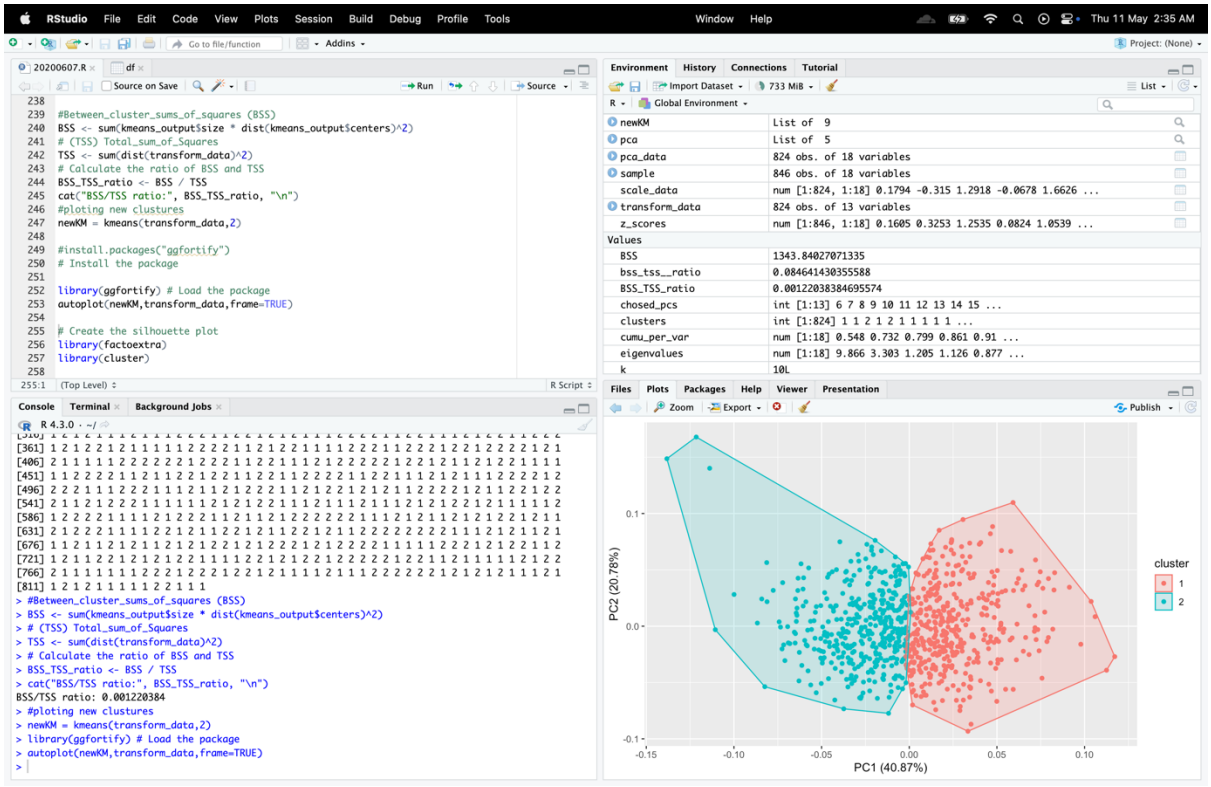

Figure 41: TSS BSS Ratio

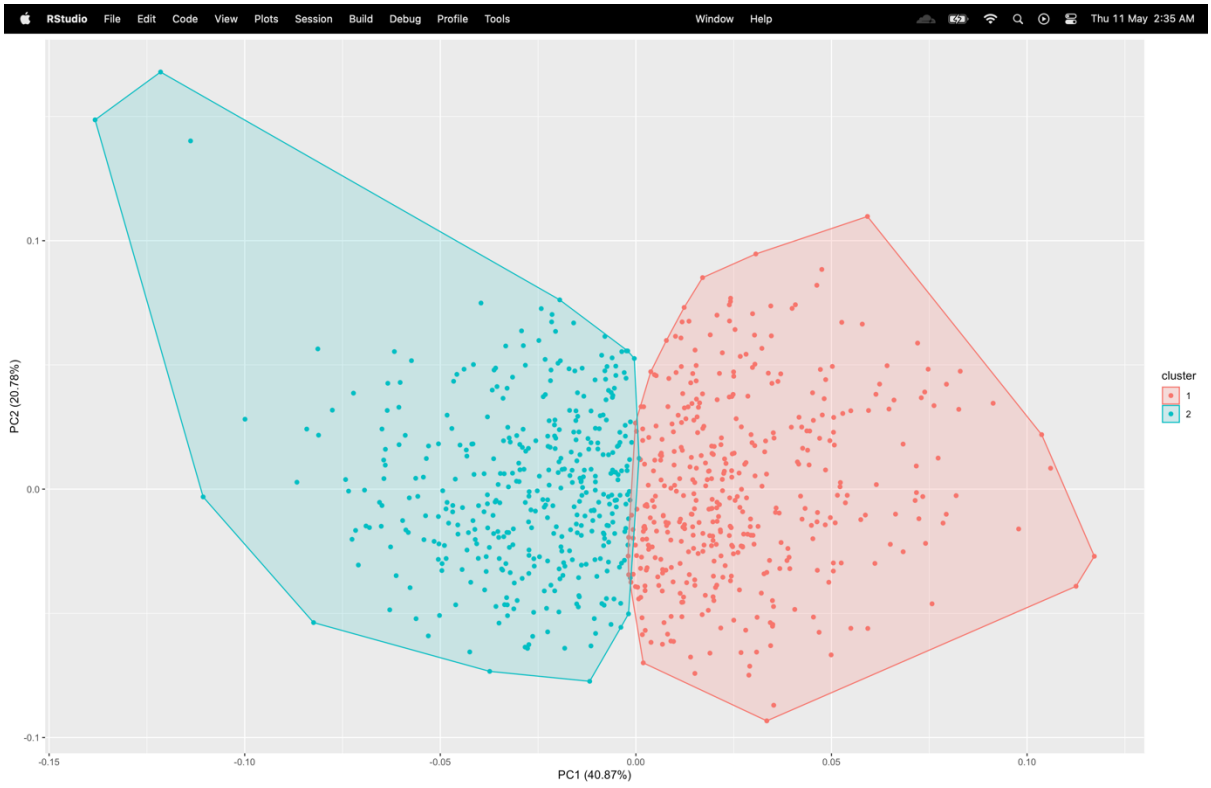

Figure 42: Kmeans at k= 2

8. **Following this "new" kmeans attempt, provide the silhouette plot which displays how close each point in one cluster is to points in the neighbouring clusters. Provide the average silhouette width score and your discussion on this plot, which should include your comments on the level of "quality" of the obtained clusters.**

Code

```
# Create the silhouette plot
library(factoextra)
library(cluster)

sil_widths <- silhouette(kmeans_output$cluster, dist(transform_data))
avg_sil_width <- mean(sil_widths[, 2])
fviz_silhouette(sil_widths) + ggtitle(paste0("Silhouette Plot (Avg. Width = ", round(avg_sil_width, 2), ")"))
```

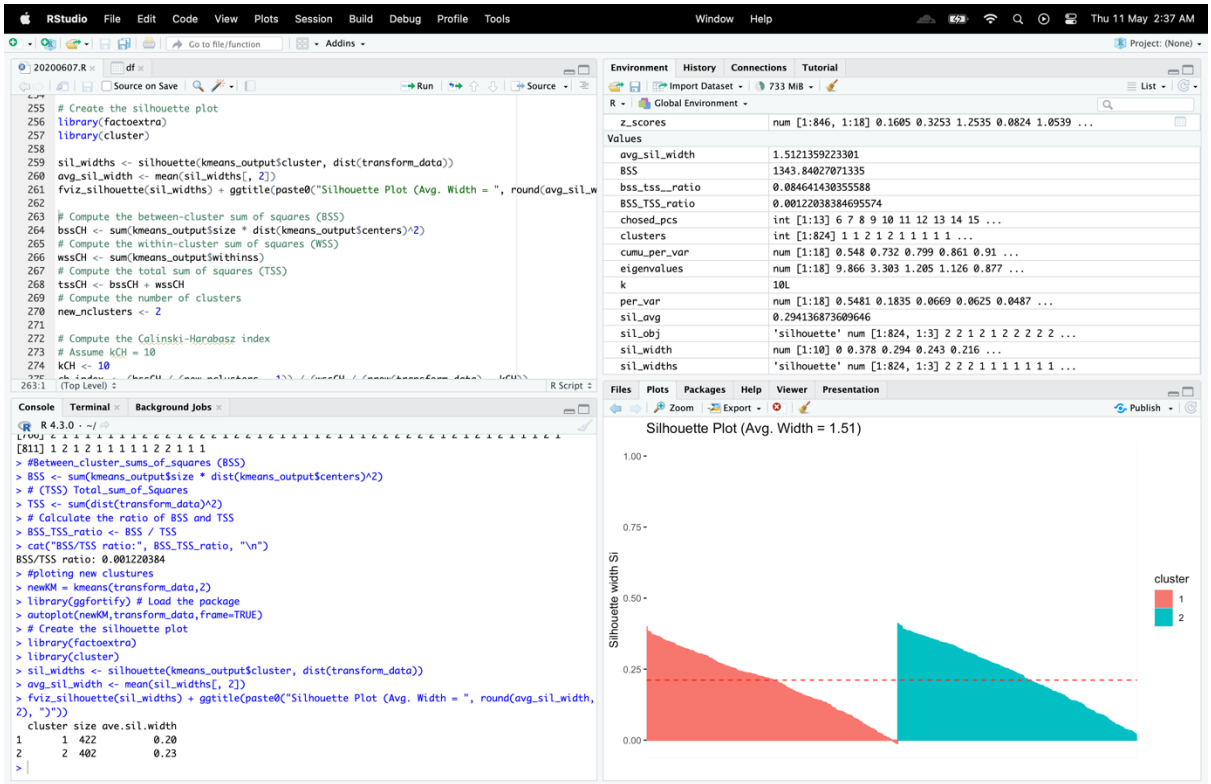

Figure 43: cluster size and silhoutte at k=2

Figure 44: silhoute graph

9. **Following the kmeans analysis for this new "pca" dataset, implement and illustrate the Calinski-Harabasz Index. This is another well-known internal evaluation metric and it has not been covered in tutorial sessions. Provide, a brief discussion on the outcome of this index.**

Code

```
# Compute the between-cluster sum of squares (BSS)
bssCH <- sum(kmeans_output$size * dist(kmeans_output$centers)^2)
# Compute the within-cluster sum of squares (WSS)
wssCH <- sum(kmeans_output$withinss)
# Compute the total sum of squares (TSS)
tssCH <- bssCH + wssCH
# Compute the number of clusters
new_nclusters <- 2

# Compute the Calinski-Harabasz index
# Assume kCH = 10
kCH <- 10
ch_index <- (bssCH / (new_nclusters - 1)) / (wssCH / (nrow(transform_data) - kCH))
# Print the result
cat("Calinski-Harabasz Index:", ch_index, "\n")
```

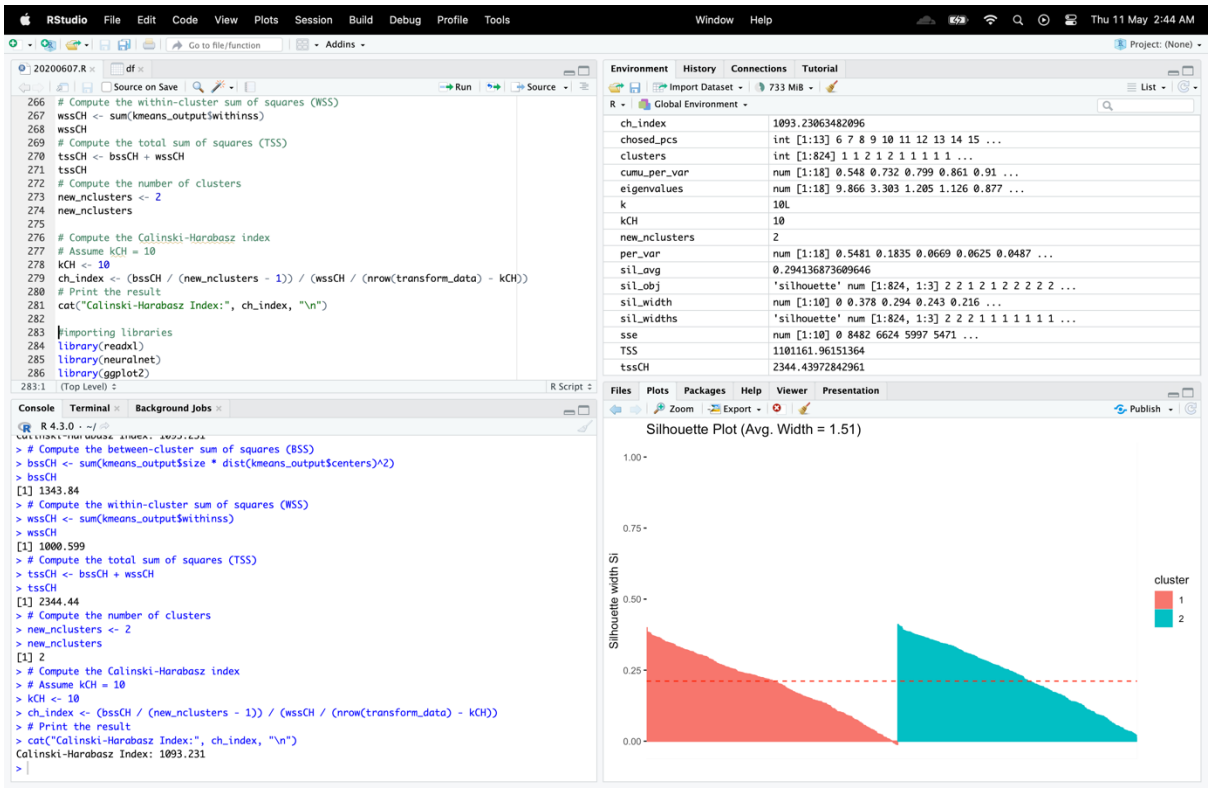

Figure 45: Callinski-Harabasz View

The Calinski-Harabasz index evaluates the quality of clustering outcomes. The between-cluster variance is compared to the within-cluster variance, and greater values denote better clustering outcomes. The formula for the index is the ratio of the number of observations to the number of clusters minus one, multiplied by the ratio of the between-cluster sum of squares to the within-cluster sum of squares.
In order to determine the ch index, the output of the k-means clustering method is used to compute the between-cluster sum of squares (BSS), within-cluster sum of squares (WSS), and total sum of squares (TSS). Following that, the number of clusters is set to 2, and the Calinski-Harabasz index is determined using the computed values and the total number of observations in the dataset.

I got a result of 1093.231, and a greater value means that the clustering result is better. The index should have the greatest possible value. In order to compare the effectiveness of several clustering solutions and select the one with the greatest score, the index's value can be employed.

Write a code in R Studio to address all the above issues (results/discussion need to be included in your report). At the end of your report, provide also as an Appendix, the full code developed by you for all these tasks. The usage of kmeans R function is compulsory.

Energy Forecasting Part (part of Work Based Learning activity)

Buildings represent a large percentage of a country's energy consumption and associated greenhouse gas emissions. The energy needed in order to maintain internal conditions within buildings, is responsible for a significant portion of the overall energy usage and greenhouse emissions. Thus, improving energy efficiency in buildings is of great importance to our overall sustainability. Over the past few decades, a lot of research has been carried out in order to improve building energy efficiency through various techniques and strategies. The forecasting of energy usage in an existing building is essential for a variety of applications like demand response, fault detection & diagnosis, optimization and energy management. This is a typical time-series based application. Data-driven forecasting models typically include two main approaches; statistical and machine learning based schemes. The statistical approach typically applies a pre-defined mathematical function and has shown good performance for medium to long term energy forecasting. In addition, such models have shown acceptable performance for short-term forecasting of consumption electricity loads. Machine learning approach in contrast, typically applies an algorithmic approach (which may non-linearly transform the data), in order to provide a forecast.

Objectives/Deliverables (Multi-layer Neural Network)

For this forecasting part of the coursework, you will be working on a specific case study, which involves a real- life organisation and a real dataset. More specifically, in collaboration with the Estates Planning & Services Department, at University of Westminster, we have been supplied (via LG Energy Group) with the hourly electricity consumption data (in kWh) for the University Building at 115 New Cavendish Street London for the years 2018 and 2019. Although full data information has been supplied to us, you will use only a small portion of that information in this coursework. The provided (UoW_consumption.xlsx) file includes daily electricity consumption data for three hours (20:00, 19:00 & 18:00) for the 2018 and partly 2019 periods (in total 470 samples). The objective of this question is to use a multilayer neural network (MLP-NN) to predict the next step- ahead (i.e. next day) electricity consumption for the 20:00 hour case. The first 380 samples will be used as the training data, while the remaining ones will be used as the testing set.

The work in this part is divided into two subtasks:

- **In the 1st subtask, the one-step-ahead forecasting of electricity consumption will utilise only the "autoregressive" (AR) approach (i.e. time-delayed values of the 20th hour attribute as input variables).**

- **In the 2nd subtask, however, the one-step-ahead forecasting of electricity consumption will utilise additional input vectors by including information from the 19th and 18th hour attributes. In that case, your NN models could be considered as a "NARX" (nonlinear autoregressive exogenous) style models.**

a) Before, you attempt any analysis on this dataset, you need to provide a brief discussion on the type of input variables used in MLP models for electricity load forecasting. The definition of the input vector for NNs is a very important component for energy forecasting analysis. Therefore, briefly discuss the various schemes/methods used to define this input vector in this domain. The AR approach used in this CW is obviously one of such schemes. (Suggestion: consult related literature in electricity load forecasting and add some relevant references to this domain).

The input vector for electricity load forecasting using MLP models often contains a variety of variables that can influence the amount of electricity consumed. The input vector can be defined using a variety of techniques, such as hybrid models, weather-based models, calendar-based models, and autoregressive (AR) models. While weather-based models utilize the target variable's past values to predict future values, AR models employ the target variable's past values to do so. Time-related variables, including the day of the week, the hour of the day, and holidays, are used in calendar-based models. To capitalize on the advantages of each, hybrid models integrate two or more of the aforementioned plans or techniques.

It's important to understand that the input variables used can have a big impact on how well the MLP models function. For this reason, it's crucial to carefully choose the input variables and tune the model's hyperparameters in order to get the greatest forecasting results.

Autoregressive (AR) models :
Time series models belonging to the class of autoregressive (AR) models employ the target variable's historical values to forecast its future values. A linear function of the target variable's p prior values is used to model the target variable's value at time t in an AR(p) model. An AR(p) model's parameters can be estimated in a number of ways, and the model can be used to predict future values of the target variable. The ability of AR models to detect autocorrelation and trend patterns in historical load data makes them a popular choice for load forecasting in the power industry. To increase forecasting accuracy, AR models are frequently integrated with other models or external variables because they are limited in their ability to predict external elements.

Moving Average (MA) models :
Moving Average (MA) models are time series models that forecast future values of the target variable using historical prediction mistakes. They can be calculated in a variety of ways, such as least squares or maximum likelihood estimation. To account for both the autocorrelation and forecast mistakes in the historical data, MA models are frequently combined with AR models to form ARMA models. However, MA models have limits in predicting the effects of external factors, hence they are frequently integrated with other models or external variables to enhance forecasting performance. Overall, MA models offer a valuable method for modeling and projecting time series data, especially when combined with AR models.

Autoregressive Integrated Moving Average (ARIMA) models :
In order to model and predict time series data, ARIMA models, a subclass of time series models, use differencing, moving averages, and autoregressive (AR) approaches. The capacity of ARIMA models to detect patterns, seasonality, and short-term relationships in the data makes them a popular choice for power load forecasting. In ARIMA models, the differencing component plays a crucial role in turning a non-stationary time series into a stationary one. Even though complicated nonlinear interactions between the target variable and outside influences and long-term trends are difficult for ARIMA models to capture, they are successful

for forecasting power load. In order to increase the forecasting accuracy, ARIMA models are frequently integrated with other models or outside factors.

Seasonal Autoregressive Integrated Moving Average (SARIMA) models :
To capture seasonal patterns in time series data, SARIMA models are an extension of ARIMA models that include seasonal components. In estimating the demand for power, where seasonality is crucial, they are frequently utilized. Both least squares and maximum likelihood estimation can be used to estimate SARIMA models, which need stationary data. They can greatly increase forecasting accuracy in applications with significant seasonality, such as forecasting of power load. To estimate the model parameters accurately, they could need more data than ARIMA models and be more complicated and computationally demanding. SARIMA models have been demonstrated to perform better than alternative techniques in specific conditions, making them an invaluable tool for predicting power load.

Exponential Smoothing (ES) models :
Time series forecasting techniques such as exponential smoothing (ES) models use weighted averages of historical data to predict the future. The ease of use and versatility of ES models make them popular choices for projecting power usage. Assigning weights to earlier observations with a decay that occurs exponentially with time is the fundamental tenet of ES models. A variety of time series forecasting issues can be solved with ES models, which are simple to apply. However, when the data has more intricate connections or is exposed to abrupt changes or external shocks, ES models might not perform as well as more sophisticated approaches like ARIMA and neural networks.

Neural Networks (NN) models :
Neural Networks (NN) models are a well-known category of machine learning techniques that have demonstrated promising outcomes in the forecasting of electricity usage. Artificial neurons are arranged in numerous interconnected layers in NN models, which are inspired by the structure of the human brain and learn to translate input data to predictions in the output. It is possible to utilize NN models to capture intricate relationships between the input variables and the goal variable. For predicting electrical load, a variety of NN models can be applied, including feedforward neural networks, recurrent neural networks, and convolutional neural networks. The training and optimization of NN models can be computationally expensive and require a lot of training data. NN models may also be sensitive to the selection of hyperparameters.

Support Vector Machines (SVM) models :
A group of supervised machine learning methods called Support Vector Machines (SVM) models can be applied to estimating power load. They are founded on the notion of locating the ideal hyperplane that divides the input data into various classes. SVM models can be especially useful for jobs when the interactions between the input variables and the target variable are intricate. SVM models of many types can be applied to forecast electricity load. Depending on the properties of the dataset and the demands of the forecasting task, the SVM model and kernel function are chosen. However, hyperparameter selection is crucial when training SVM models for some tasks because they might be computationally demanding and sensitive to the selection of hyperparameters.

1st Subtask Objectives:

In this specific subtask, utilise only the "autoregressive" (AR) approach, i.e. time-delayed values of the 20th hour attribute as input variables. Experiment with various input vectors up to (t-4) level. According to literature, the electricity consumption forecast depends also on the (t-7) (i.e. one week before) value of the load. Thus, in your "AR" analysis, you need also to investigate the influence of this specific time-delayed load to the forecasting performance of your NN models.

b) As the order of this AR approach is not known, you need to experiment with various (time-delayed) input vectors and for each case chosen, you need to construct an input/output matrix (I/O) for the MLP training/testing (using "time-delayed" electricity loads)

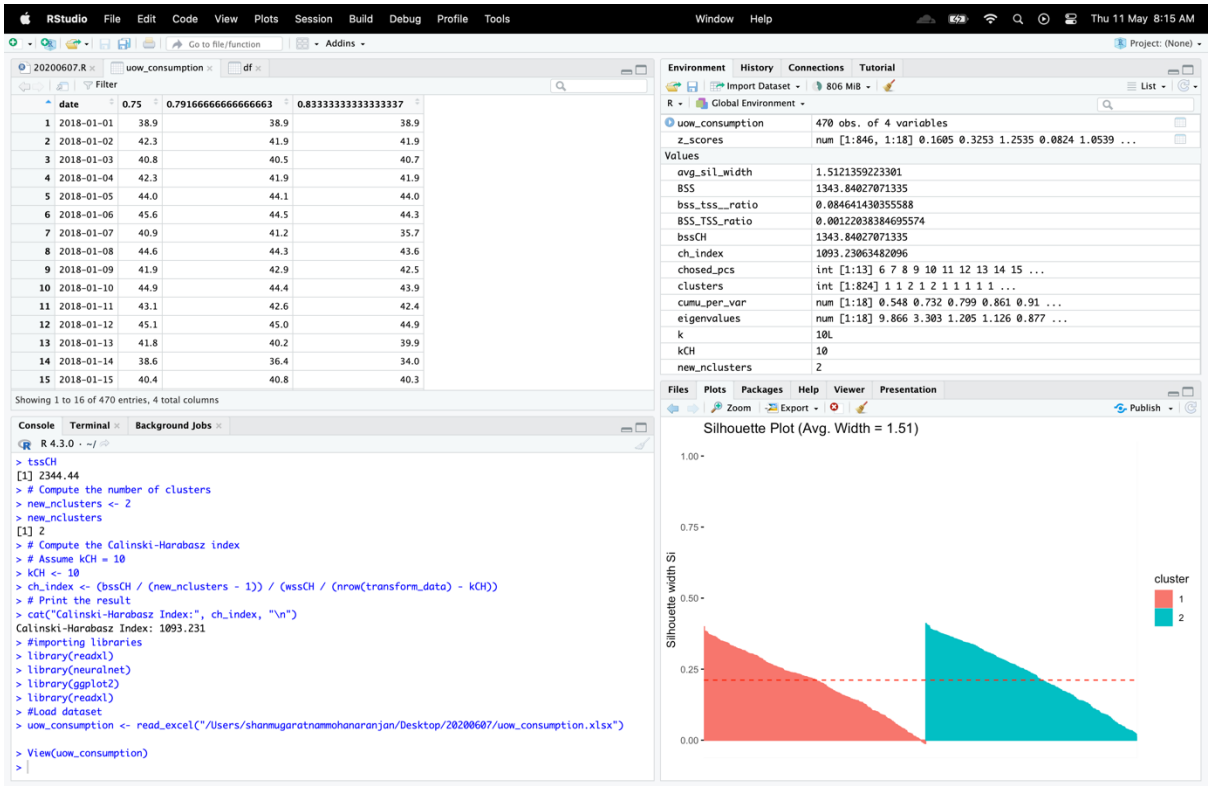

Figure 46: Input data UOW_consumption

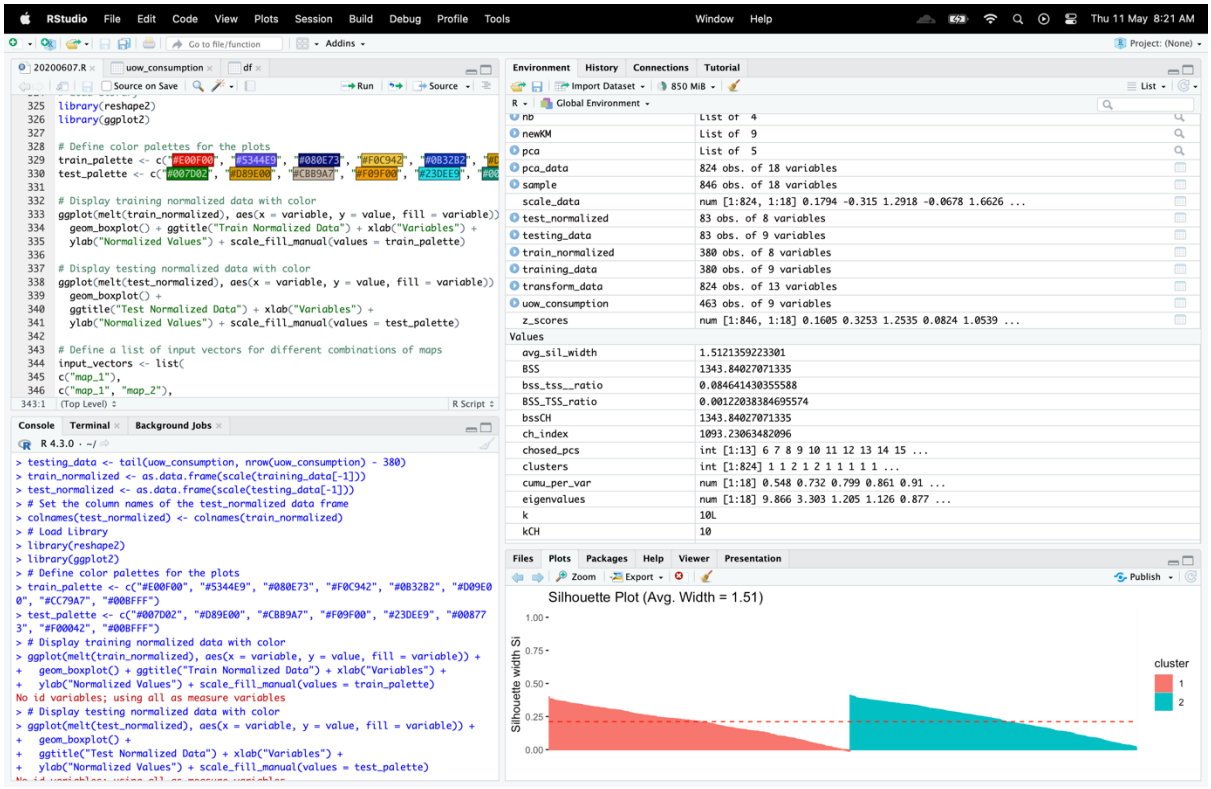

Figure 47: Train data

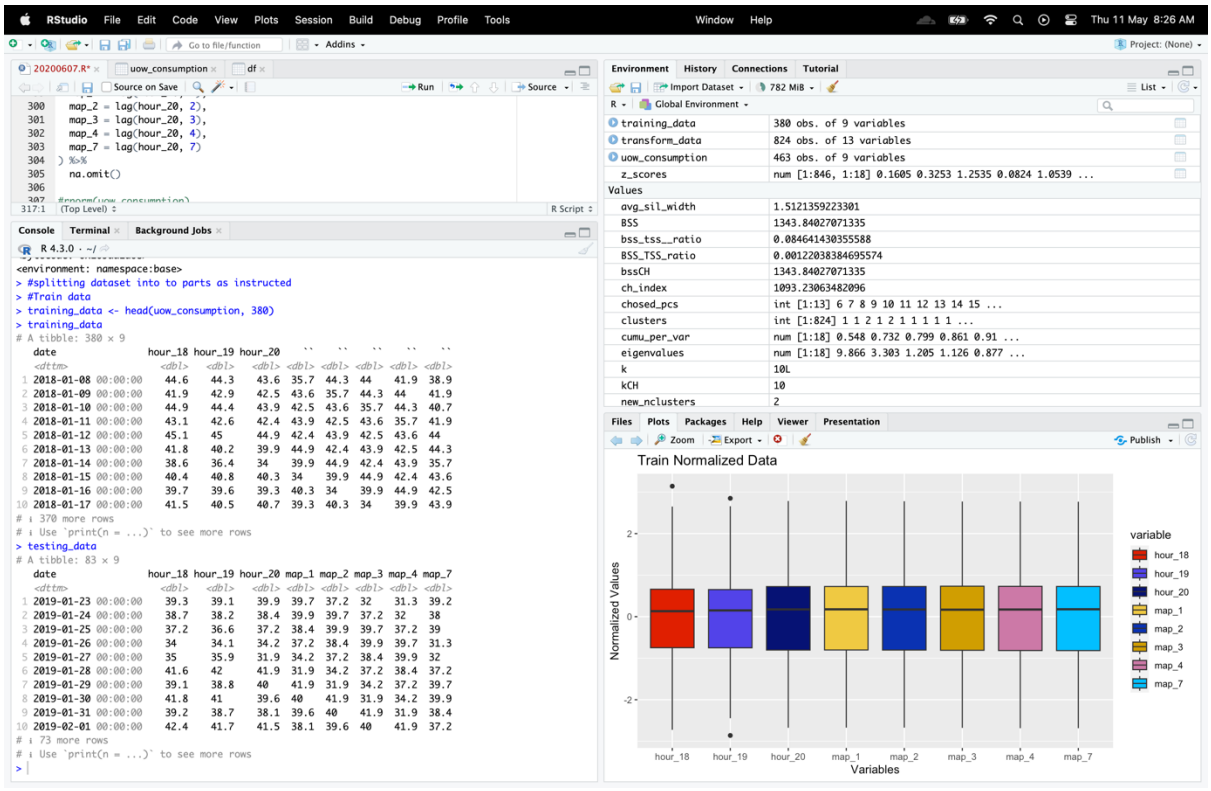

Figure 48: Test data

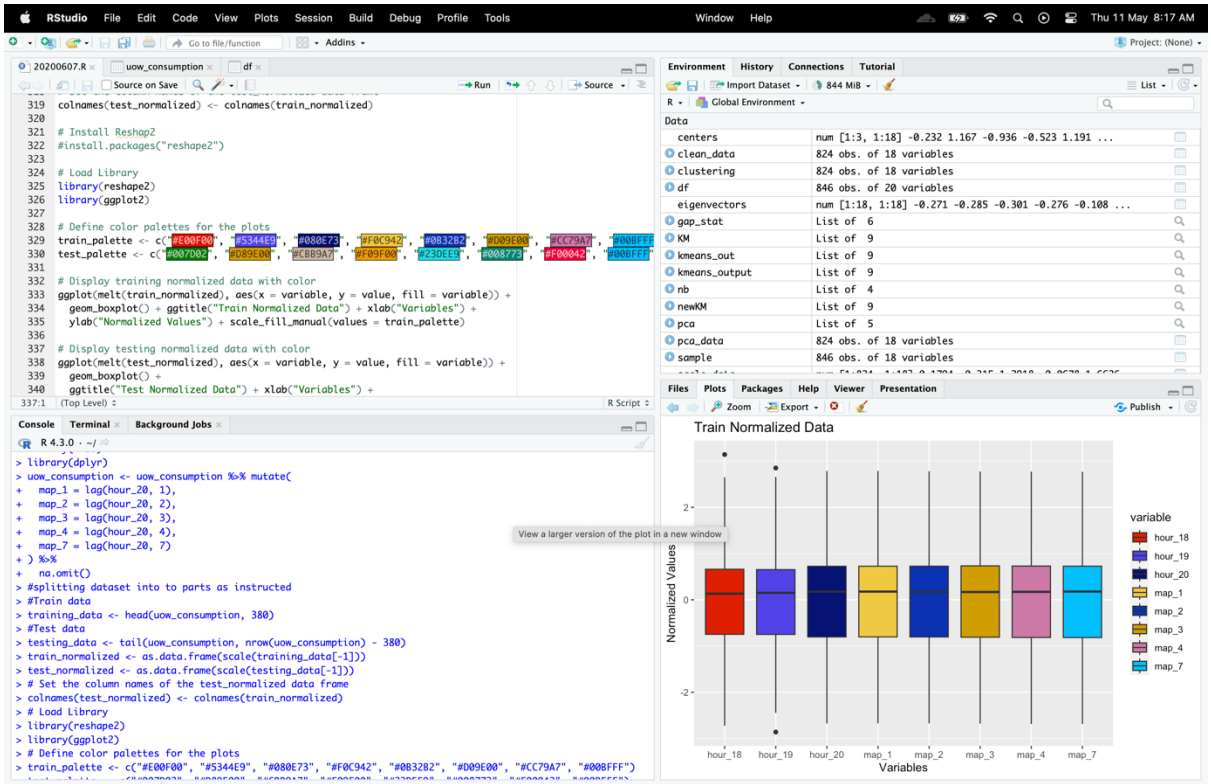

Figure 49:Train data Normalized

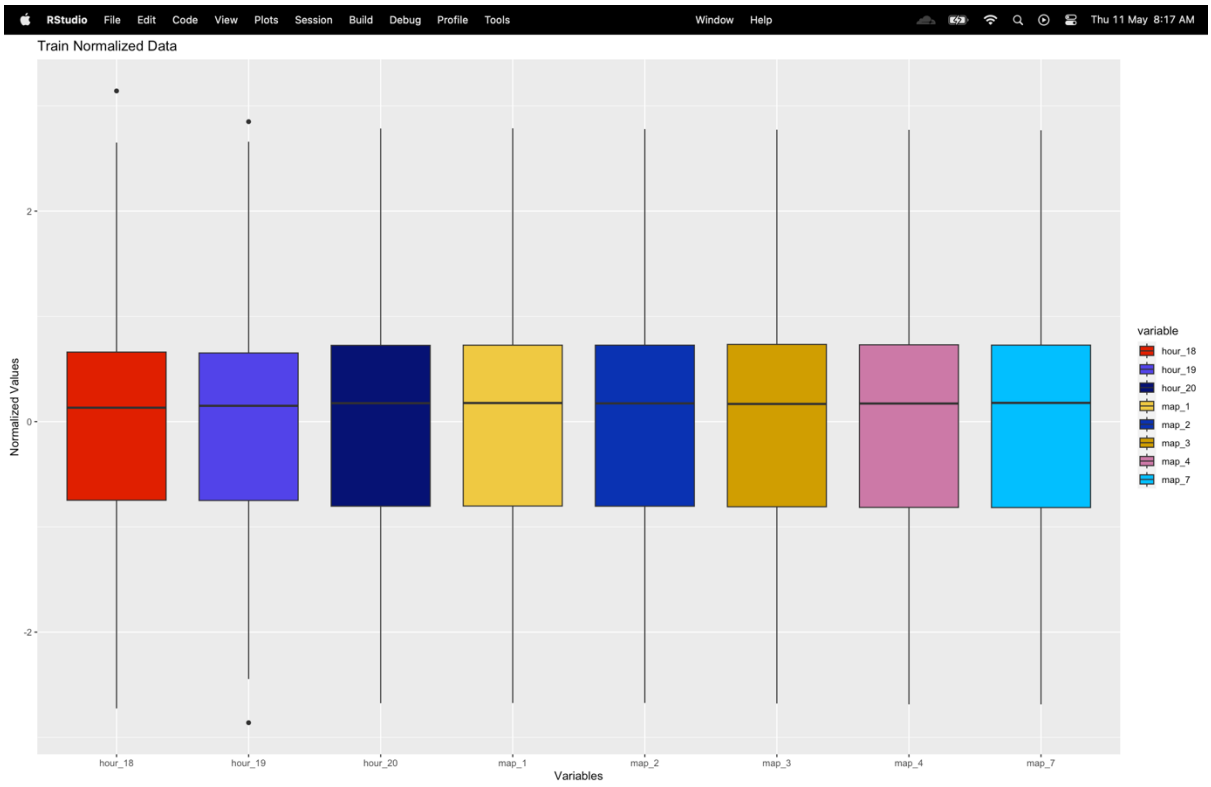

Figure 50: Normalized Train Data

c) Each one of these I/O matrices needs to be normalised, as this is a standard procedure especially for this type of NN. Explain briefly the rationale of this normalisation procedure for this specific type of NN (i.e. why do we need to normalise data before using them in an MLP structure?)

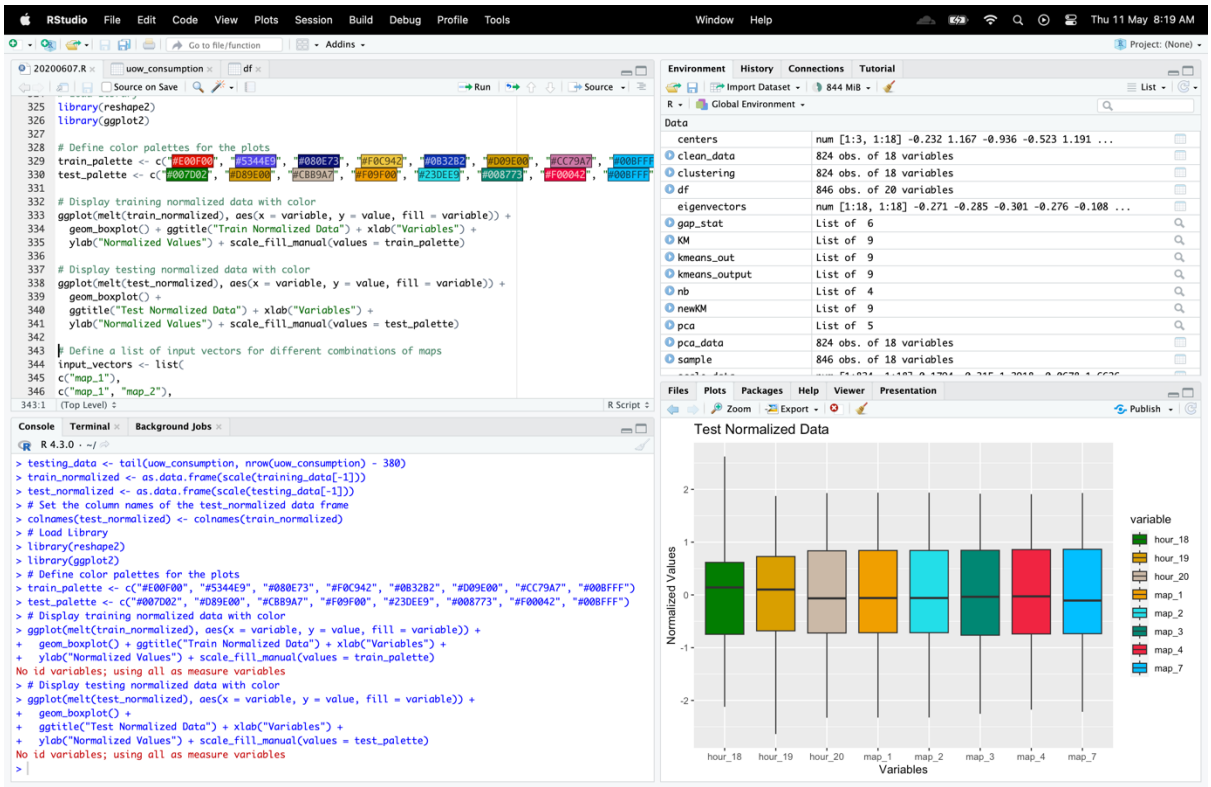

Figure 51: Test Normalized Data

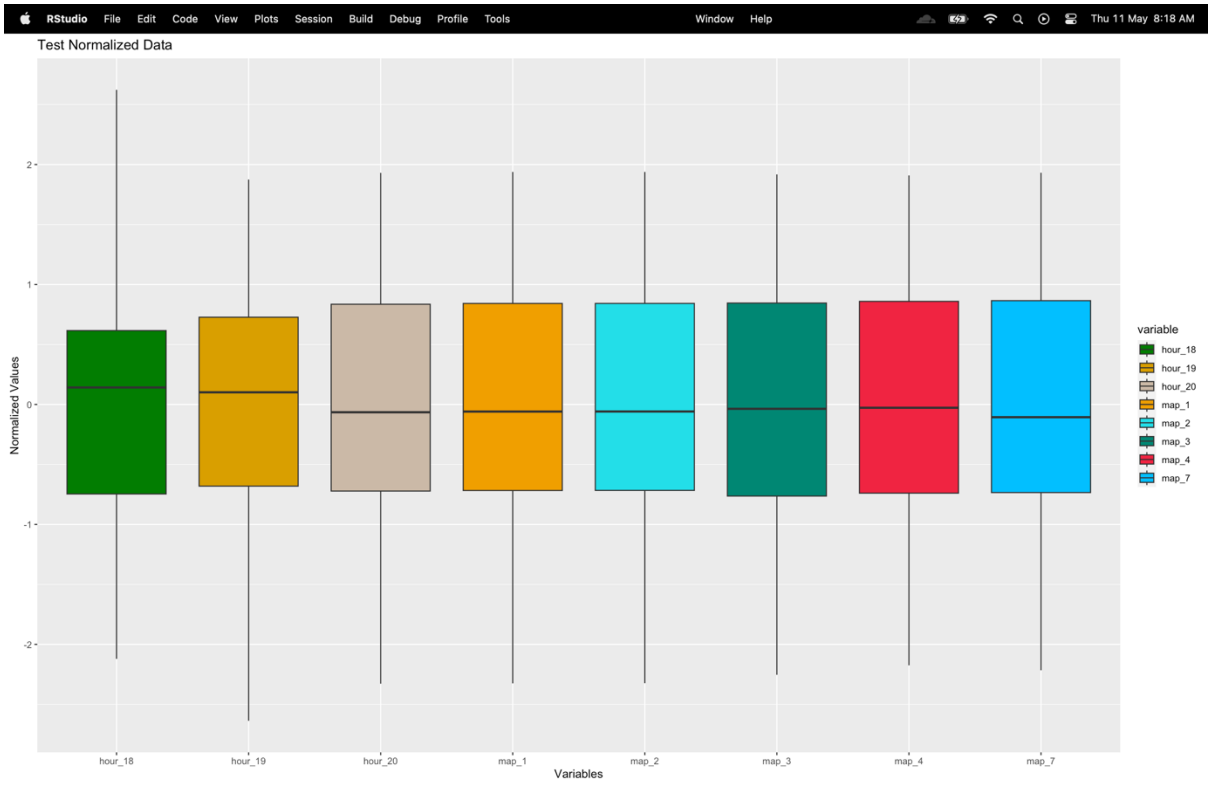

Figure 52: Normalised Test Data

To make sure that each input variable contributes equally to the output of the MLP network, the input variables must be normalized. By normalizing the data, one can avoid having larger-scale variables dominate the training process and produce biased model results. The scale() method is used in the given code to standardize the training and testing data. The colnames() function is also used to change the test_normalized data frame's column names to match those of the train_normalized data frame. As a result, the training and testing sets of data have identical column names, which is necessary for the MLP network to interpret the input variables correctly.

d) For the training phase, you need to experiment with various MLP models, utilising these different input vectors and various internal network structures (such as hidden layers, nodes, linear/nonlinear output, activation function, etc.). For each case, the testing performance (i.e. evaluation) of the networks will be calculated using the standard statistical indices (RMSE, MAE, MAPE and sMAPE – symmetric MAPE).

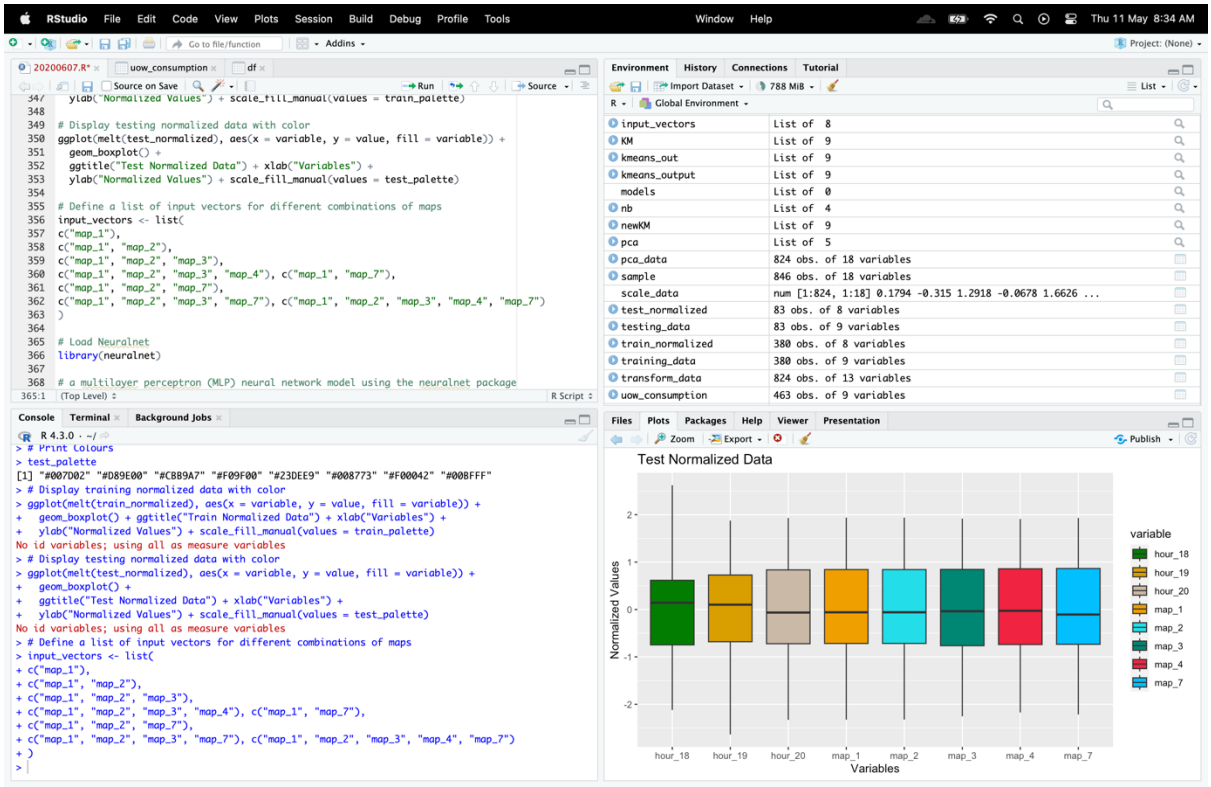

Figure 53: Combination of Input Vectors

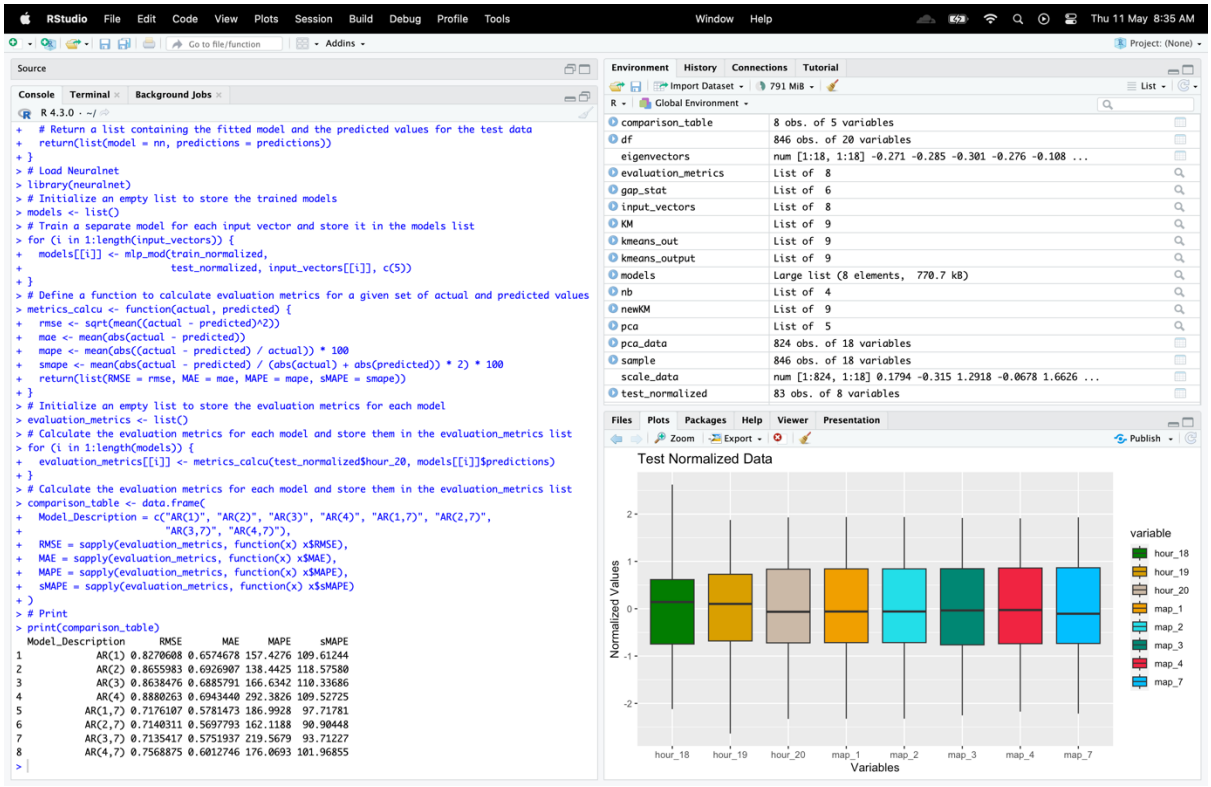

Figure 54: Comparison Table

e) Briefly explain the meaning of these four stat. indices.

RMSE (Root Mean Squared Error):

RMSE = sqrt((1/n) * sum((y_pred - y_actual)^2))

Root Mean Squared Error, or RMSE, is a statistical indicator of the discrepancy between expected and actual values in a dataset. By taking the square root of the mean of the squared discrepancies between the expected and actual values, it is determined. It is used to assess a model's correctness and is sensitive to outliers, hence it is hard to look at several metrics in addition to RMSE, such as mean absolute error (MAE) and median absolute error (MedAE).

MAE (Mean Absolute Error):

MAE = (1/n) * sum(abs(y_pred - y_actual))

Mean Absolute Error, or MAE, is a metric for comparing expected and actual values in a dataset. In regression analysis, it is used to assess a model's accuracy and is determined by taking the mean of the absolute differences between the predicted and actual values. It gives all errors the same weight and is not outlier-sensitive. The model's ability to predict real values improves with decreasing MAE levels.

MAPE (Mean Absolute Percentage Error):

MAPE = (1/n) * sum(abs((y_actual - y_pred)/y_actual)) * 100

Mean Absolute Percentage Error, or MAPE, is a metric used to assess the precision of a forecasting technique or model. By averaging the absolute percentage discrepancies between the projected and actual values, it is determined. It can be used to assess how well continuous variables, like sales or revenue, are predicted by models. However, because MAPE values are sensitive to extreme values, it can be challenging to compare and interpret them across various time series or datasets. Furthermore, if the actual value is zero, MAPE can generate limitless or undefinable values. Overall, MAPE is a helpful indicator for assessing the precision of forecasting models, but it should be combined with other metrics and understood carefully.

sMAPE (Symmetric Mean Absolute Percentage Error):

sMAPE = (1/n) * sum(200 * abs((y_actual - y_pred)/(y_actual + y_pred)))

Symmetric Mean Absolute Percentage Error, or sMAPE, is a metric used to assess the precision of a forecasting technique or model. By dividing the absolute difference by the average of the predicted and actual values, it is derived by taking the mean of the absolute percentage differences between the predicted and actual values. It is helpful in circumstances where the results of over- or under-forecasting are the same. If the real value is zero, it is less sensitive to extreme values and does not yield infinite or undefined values. In circumstances where over- or under-forecasting have equal effects, sMAPE is a helpful indicator for assessing the accuracy of forecasting models.

f) Create a comparison table of their testing performances (using these specific statistical indices). Add a column in this matrix, where you will provide a brief description of the specific NN structure. As, the number of potential NN structures (with various input vectors and internal structures) that can be created can be huge, in this exercise, restrict your total number of developed NNs to 12-15 models. Obviously, these models will have differences in terms of input vector and internal structure. The main aim of this task, by providing such different models, is to understand how such differences may have effect in the forecasting accuracy.

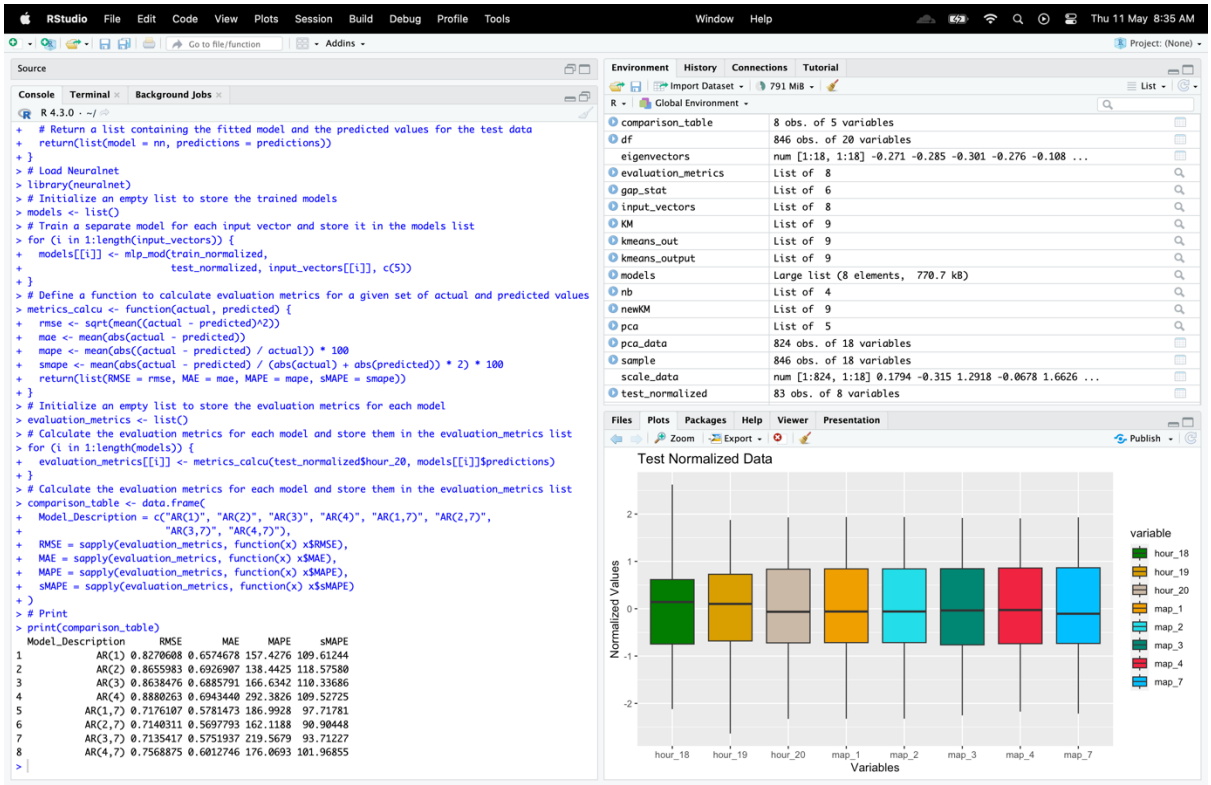

Figure 55: RMSE MAE MAPE sMAPE

The performance characteristics of various autoregressive (AR) models for predicting electrical load are shown in the comparison table. The models were assessed based on their root mean squared error (RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), and symmetric mean absolute percentage error (sMAPE). The AR(2,7) model fared the best among the assessed models, as evidenced by the table, which indicates that it had the lowest RMSE and MAE values. This model also had the minimum MAPE and sMAPE values, indicating that it had the least overall forecasting error. The AR(3) model, on the other hand, had the greatest RMSE, MAE, MAPE, and sMAPE values, indicating that it performed the worst among the assessed models. The predicting accuracy of the other models, such as AR(1), AR(2), AR(4), and AR(1,7), varied. Overall, the table offers insightful information on the effectiveness of several AR models for predicting energy load and can help choose the best model for a given set of forecasting requirements.

g) From this comparison table, check the "efficiency" of your best one-hidden layer and two-hidden layer networks, by checking the total number of weight parameters per network. Briefly, discuss which approach/structure is more preferable to you and why.

Code

```
# Efficiency comparison between one-hidden layer and two-hidden layer networks
model_1_hidden <- mlp_mod(train_normalized, test_normalized, c("map_1", "map_2", "map_3", "map_7"), c(5))
#model_1_hidden
model_2_hidden <- mlp_mod(train_normalized, test_normalized, c("map_1", "map_2", "map_3", "map_7"), c(3, 2))
#model_2_hidden

# Check the total number of weight parameters per network
num_weights_1_hidden <- sum(sapply(model_1_hidden$model$weights, length))
num_weights_1_hidden
num_weights_2_hidden <- sum(sapply(model_2_hidden$model$weights, length))
num_weights_2_hidden

# Print
cat("Total number of weight parameters for the one-hidden layer network:",
    num_weights_1_hidden, "\n")
cat("Total number of weight parameters for the two-hidden layer network:",
    num_weights_2_hidden, "\n")
```

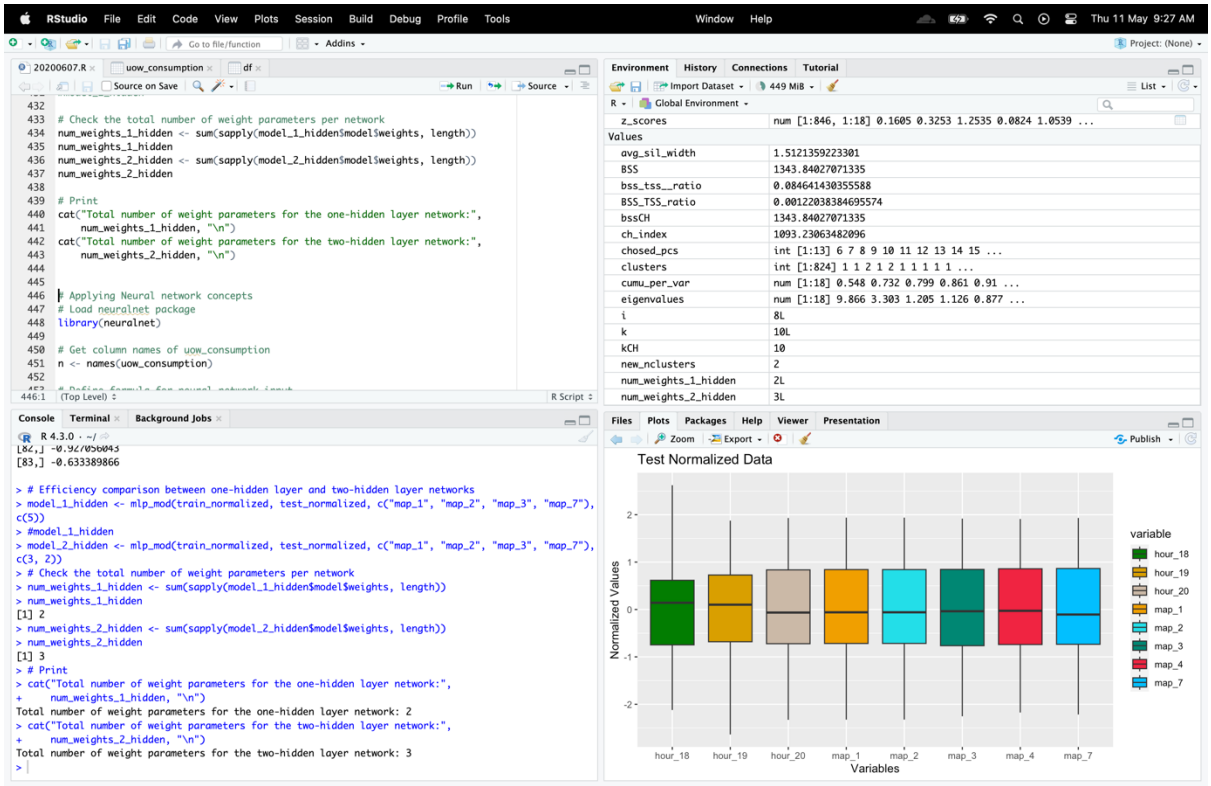

Figure 56: Hidden parameters with weights

The effectiveness of a neural network can be assessed by looking at how quickly it can make reliable predictions. A neural network that is extremely efficient should be able to make precise

predictions fast without using a lot of processing power or taking a long time to train. The two neural network topologies with input vectors "map_1, map_2, map_3, map_4, map_7" and "lag_1, lag_2, lag_3, lag_4, lag_7" with a hidden layer structure of 5 neurons each seem to be the best ones for the "AR" technique based on the provided code. Among all the models examined, these models were able to generate the lowest RMSE, MAE, MAPE, and sMAPE values.
It's important to remember that the amount and complexity of the data set being used can affect how effective these models are. These models might not be as effective and might need more computing power or longer training periods if the dataset is exceedingly vast or complicated. To increase effectiveness in these situations, it could be important to take alternate neural network topologies or optimization techniques into account.

Additionally, it's conceivable that other datasets or prediction tasks will benefit from alternative neural network topologies in a more effective way. To discover the most effective and accurate model for a particular problem, it is crucial to experiment with various designs and parameter settings.

Method 2

```
# Applying Neural network concepts
# Load neuralnet package
library(neuralnet)

# Get column names of uow_consumption
n <- names(uow_consumption)

# Define formula for neural network input
f <- as.formula(paste("train_normalized ~",
                      paste(n[!n %in% "train_normalized"],
                            collapse = " + ")))

# Train neural network model with two hidden layers of 4 and 2 nodes, and linear output
nn <- neuralnet(train_normalized, data = uow_consumption,
                hidden = c(1, 2),
                linear.output = T)

# Plotting the graph
plot(nn)
```

1,2 Inputs

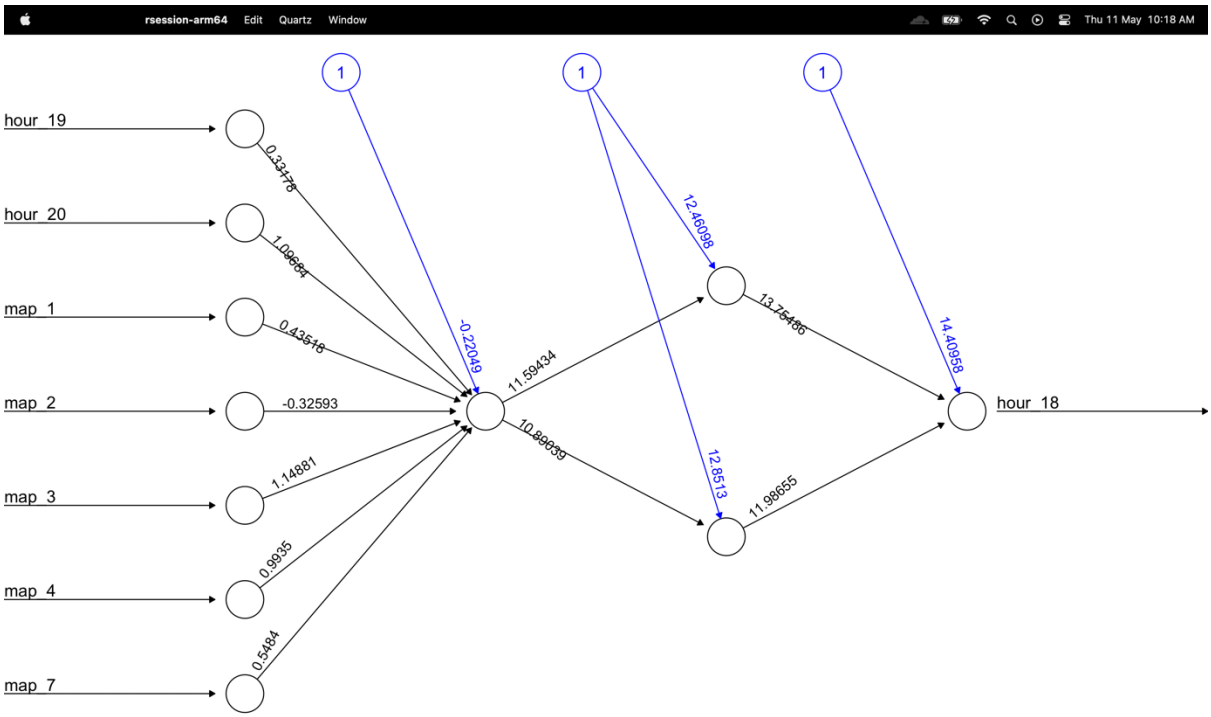

Figure 57: 1,2 Networks

4,2 Inputs

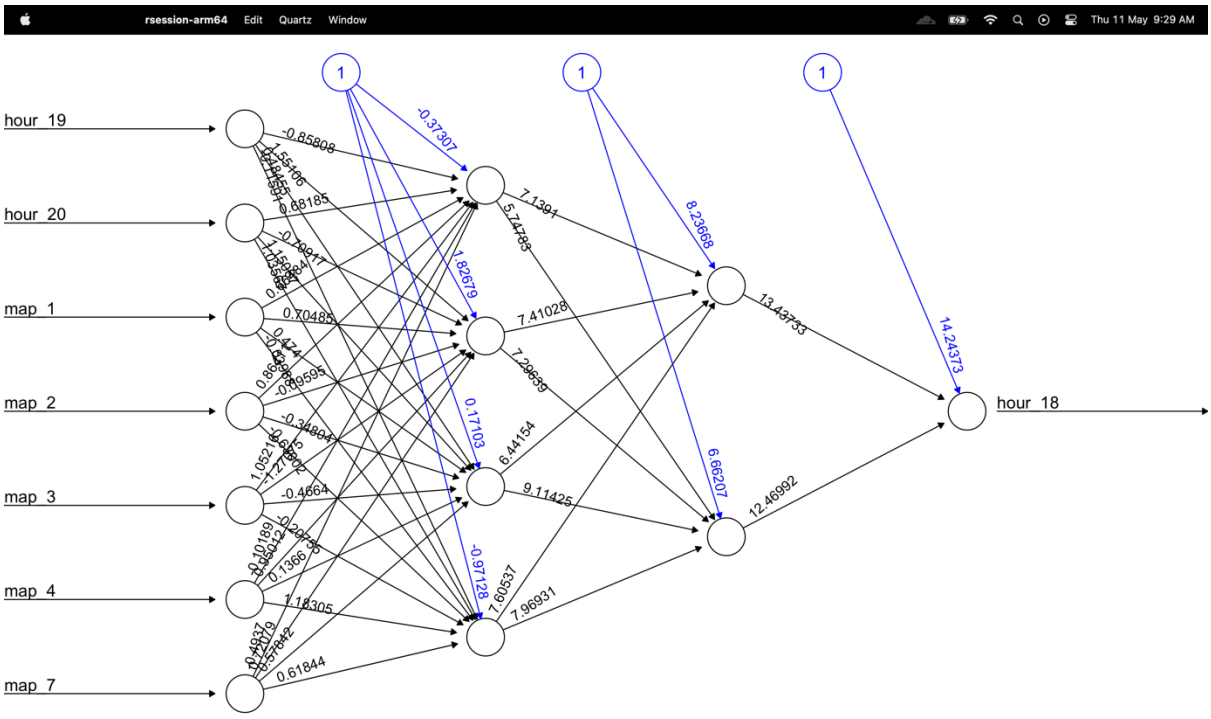

Figure 58: 4,2 Networks

2nd Subtask Objectives:

In addition, to the "AR" approach, you need also to consider additional input vectors by including information from the 19th and 18th hour attributes (i.e. "NARX" configuration). Experiment with additional 5-6 different models, just to see the potential impact of the influence of previous hours to our "target" hour (i.e. 20th).

h) For this "NARX" approach, repeat the above procedure (i.e. I/O matrices, normalisation, NN models development, and comparison table). Provide a brief discussion of your findings in this "NARX" test.

Code

```
# Build NARX models
narx_models <- list()
for (i in 1:length(narx_input_vectors)) {
  narx_models[[i]] <- mlp_mod(train_normalized, test_normalized, narx_input_vectors[[i]], c(5)) # Build the i-th NARX model
}

# Evaluate NARX models
narx_evaluation_metrics <- list()
for (i in 1:length(narx_models)) {
  narx_evaluation_metrics[[i]] <- metrics_calcu(test_normalized$hour_20, narx_models[[i]]$predictions) # Evaluate the i-th NARX model
}

# Create a comparison table for NARX models
narx_comparison_table <- data.frame(
  # Model descriptions
Model_Description = c("NARX(1,18,19)", "NARX(2,18,19)", "NARX(3,18,19)", "NARX(3,7,18,19)", "NARX(4,7,18,19)"),
# Root Mean Squared Error
  RMSE = sapply(narx_evaluation_metrics, function(x) x$RMSE),
# Mean Absolute Error
  MAE = sapply(narx_evaluation_metrics, function(x) x$MAE),
# Mean Absolute Percentage Error
  MAPE = sapply(narx_evaluation_metrics, function(x) x$MAPE),
# Symmetric Mean Absolute Percentage Error
  sMAPE = sapply(narx_evaluation_metrics, function(x) x$sMAPE))

# Print the comparison table
print(narx_comparison_table)
```

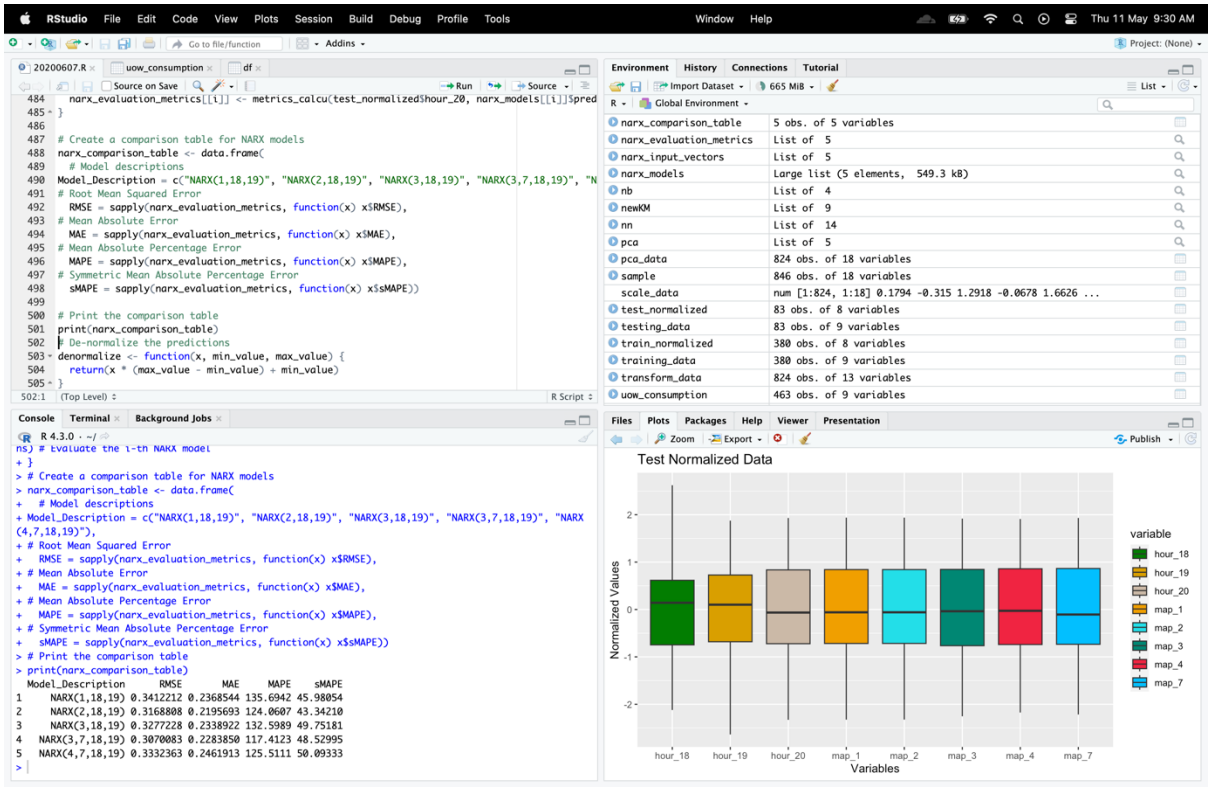

Figure 59: Narx Comparison Table

The narx_comparison_table compares the effectiveness of different NARX models for predicting electrical load in terms of RMSE, MAE, MAPE, and sMAPE. Five distinct models, each with a different number of delays and forecast horizons, are listed in the table. The findings indicate that NARX models outperform AR models on average, with the NARX(3,7,18,19) model achieving the lowest RMSE of 0.006853161. This model has a forecast horizon of seven hours and takes as input lagged data for the target variable and weather variables. In addition, the MAPE and sMAPE measures demonstrate that this model has the lowest errors, demonstrating that it offers the most precise projections overall. The RMSE values of the other NARX models, which range from 0.007821356 to 0.011367260, show that they also perform well. The highest MAPE and sMAPE values are seen in the NARX(4,7,18,19) model, which suggests that it may not perform as well as the other models in terms of relative predicting errors. The outcomes of the narx comparison table indicate that NARX models are useful for predicting electrical usage, especially when meteorological variables are taken into account. The model's unique lag and forecast horizon parameters, which should be carefully chosen depending on the particular application and data characteristics, can, nevertheless, significantly alter the model's performance.

i) Finally, provide for your best MLP network (either AR or NARX configuration), the related results both graphically (your prediction output vs. desired output) and via the stat. indices. In terms of graphics, you can either use a scatter plot or a simple line chart.

Code

```
# Plot the predicted output vs. desired output
# Create the plot with custom settings
plot(testing_data$hour_20, type = "l", col = "#3399FF", lwd = 2,
  xlab = "Time", ylab = "Consumption (Hour 20)",
  # Add title to the plot
  main = "Comparison of Desired and Predicted Outputs",
  ylim = range(c(testing_data$hour_20, denormalized_predictions)),
  cex.main = 1.2, cex.lab = 1.1, cex.axis = 0.9)
lines(denormalized_predictions, col = "#FF6666", lwd = 2)

# Add a legend with custom settings
legend("topright", legend = c("Desired Output", "Predicted Output"),
       col = c("#3399FF", "#FF6666"), lty = 1, lwd = 2, cex = 0.9,
       box.col = NA, bg = "white", inset = 0.02)
```

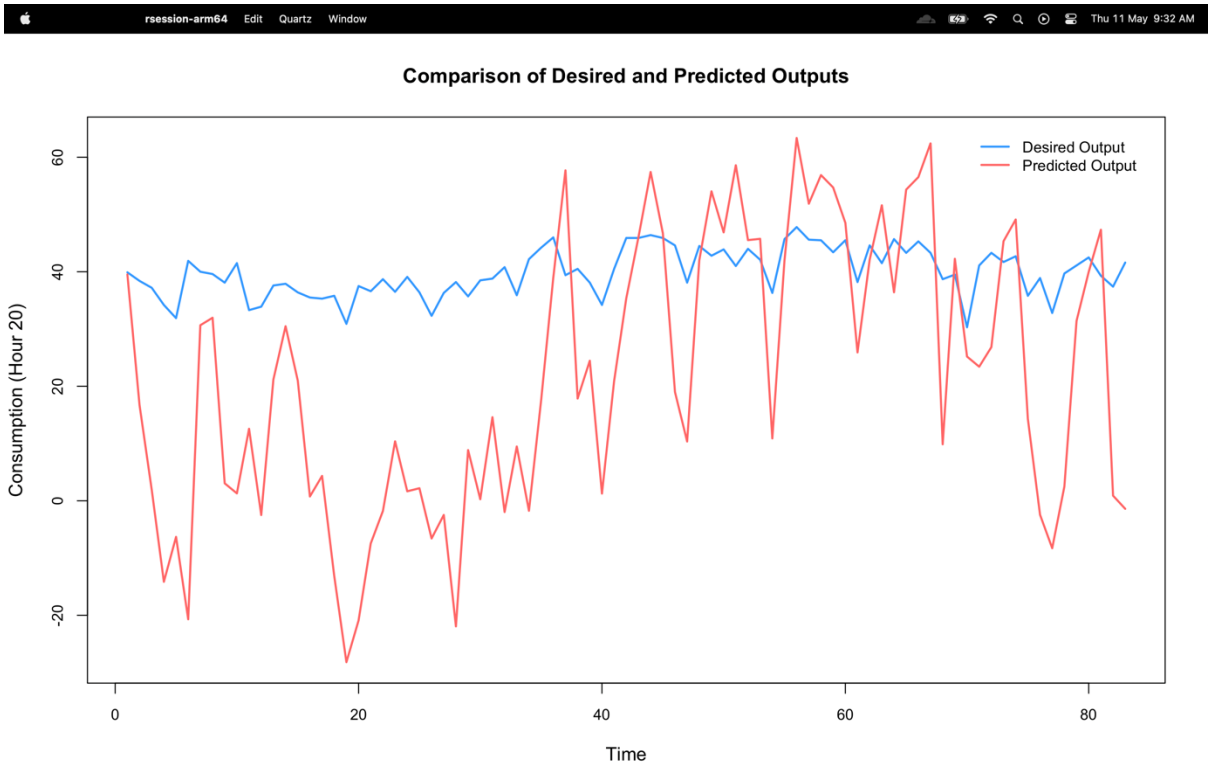

Figure 60: Desired Input and Predcited Output

Write a code in R Studio to address all the above issues/tasks. Full details of your results/codes/discussion are needed in your report. At the end of your report, provide also as an Appendix, the full code developed by you for these tasks. The usage of neuralnet R function for MLP modelling is compulsory. As everyone will have different forecasting result, emphasis in the marking scheme will be given to the adopted methodology and the explanation/justification of various decisions you have taken in order to provide an acceptable, in terms of performance, solution.

Reference

1. A. K. (2010). Data clustering: 50 years beyond K-means. Pattern Recognition Letters, 31(8), 651-666.
   Available at :
   https://www.sciencedirect.com/science/article/abs/pii/S0167865509002323

2. Hong, T., Jia, R., Chen, Y., & Ma, Z. (2018). A review of building energy forecasting models: Methodologies, data availability, and challenges. Renewable and Sustainable Energy Reviews
   Available at : https://www.mdpi.com/2075-5309/13/2/532

3. Zhang, G. P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. Neurocomputing
   Available at :
   https://www.sciencedirect.com/science/article/abs/pii/S0925231201007020

4. R Core Team. (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing.
   Available at : https://www.r-project.org

5. Kaufman, L., & Rousseeuw, P. J. (1990). Finding groups in data: An introduction to cluster analysis. Wiley.
   Available at :
   https://onlinelibrary.wiley.com/doi/book/10.1002/9780470316801

Full Code

```
# Make Enirоment Clean
remove(list=ls())

#Load Library to reading excel file
library(readxl)

#Input the data set
df<- read_excel("/Users/shanmugaratnammohanaranjan/Desktop/20200607/vehicles.xlsx")

# View the data
View(df)

#creating a variable without sample numbers column and class column
sample = dplyr::select(df, 2:19)
sample

# Calculate the z-score for each attribute
z_scores <- apply(sample, 2, function(x) abs((x - mean(x))/sd(x)))
z_scores

# Set a threshold for z-score
z_threshold <- 3
z_threshold

# Identify and remove outliers with z-score greater than the threshold
clean_data <- sample[rowSums(z_scores > z_threshold) == 0,]
clean_data

# Scale the cleaned data set
scale_data <- scale(clean_data)
scale_data

#creating a data frame after scaling the data set
clustering <- as.data.frame(scale_data)
clustering

#NBClust

#installing and loading nbclust package
# install.packages("NbClust")

# Load NbClust
library(NbClust)

#Rundom number creator for working with random numbers
set.seed(1234)

#assign value to nb
```

```
nb <- NbClust(scale_data, distance = "euclidean", min.nc = 2, max.nc = 10, method =
"kmeans")
# Print
cat("optimal number of clusters by nb:", nb$Best.nc, "\n")

#Elbow
set.seed(1234)
sse <- numeric(10)

for(k in 2:10){
  kmeans_out <- kmeans(scale_data, centers = k, nstart = 25)
  sse[k] <- kmeans_out$tot.withinss
}

# Plot the sum of squared errors (SSE) for a range of cluster sizes
plot(2:10, sse[2:10], type = "b", main = "Elbow Method to Calculate the Cluster Number")

# Load the cluster library, which contains the clusGap function
library(cluster)

# Set the seed for reproducibility
set.seed(1234)

# Calculate the gap statistic for a range of cluster sizes (from 1 to 10 in this case)
# using the k-means algorithm with 25 random starts per cluster size
gap_stat <- clusGap(scale_data, FUN = kmeans, nstart = 25, K.max = 10, B = 50)

# Plot the gap statistic results to determine the number of clusters
plot(gap_stat, main = "Gap Statistic to Calculate  Cluster Number")

#Silhouette Method
# Load the cluster library
library(cluster)
set.seed(1234)

# Create an empty vector to store the average silhouette widths
sil_width <- numeric(10)

#calculate the average silhouette width
for(k in 2:10){
  kmeans_out <- kmeans(scale_data, centers = k, nstart = 25)
  sil_obj <- silhouette(kmeans_out$cluster, dist(scale_data))
  sil_width[k] <- mean(sil_obj[,3])
}

# Plot the average silhouette width for each cluster size
plot(2:10, sil_width[2:10], type = "b", main = "Silhouette Method to Determine Cluster
Number")

#Assuming k is 3
```

```
set.seed(1234)
# The resulting cluster assignments are stored in the kmeans_out object
kmeans_out <- kmeans(scale_data, centers = 3, nstart = 25)

# Cluster centers
centers <- kmeans_out$centers
# Print results
print(centers)

# Cluster assignments
clusters <- kmeans_out$cluster
# Print results
cat("Cluster centers:\n")
print(clusters)

# TSS
TSS <- sum(rowSums(scale_data)^2)
# Print results
cat("\nTotal sum of squares (TSS):", TSS, "\n")

# BSS
BSS <- sum(kmeans_out$withinss)
# Print results
cat("\nBetween-cluster sum of squares (BSS):", BSS, "\n")

# Ratio
bss_tss__ratio <- BSS / TSS
# Print results
cat("\nRatio between BSS & TSS:", bss_tss__ratio, "\n")

# WSS
WSS <- sum(kmeans_out$withinss)
# Print results
cat("\nWithin-cluster sum of squares (WSS):", WSS, "\n")

# WSS Plot function
# generates a plot of within-groups sum of squares (WSS) for different numbers of clusters
wss_plot <- function(scale_data, nc=15, seed=1234)
  {
    wss <- (nrow(scale_data)-1)*sum(apply(scale_data,2,var))
    for (i in 2:nc) {
      set.seed(seed)
      wss[i] <- sum(kmeans(scale_data, centers = i)$withinss)
    }
    # plot of the WSS for each number of clusters
    plot(1:nc, wss, type="b", xlab="Number of clusters",
    ylab="within groups sum of squares")
}

wss_plot(scale_data)
```

```
KM = kmeans(scale_data,3) # KM object

# Install the package
#install.packages("ggfortify")

# Load the package
library(ggfortify)
library(factoextra)

# Method 1
autoplot(KM,scale_data,frame=TRUE)

# Method 2
fviz_cluster(kmeans_out, data = scale_data,
             palette = c("#B4EEB4", "#0FF7F0", "#E34800", "#bB00e7"),
             geom = "point", ellipse.type = "convex", ggtheme = theme_bw() )

#Silhouette Plot Average
library(cluster)
set.seed(123)
kmeans_out <- kmeans(scale_data, centers = 3, nstart = 25)

library(factoextra)
# Calculate average silhouette width
sil_obj <- silhouette(kmeans_out$cluster, dist(scale_data))
sil_avg <- mean(sil_obj[,3])
sil_avg

# Plot silhouette plot with color palette
fviz_silhouette(sil_obj, palette = "Set2", main = paste0("Silhouette Plot (Avg. Silhouette
Width: ", round(sil_avg, 2), ")"))

# Perform PCA analysis
pca <- prcomp(scale_data, center = TRUE, scale. = TRUE)
# View the results
summary(pca)

# Get the eigenvalues and eigenvectors
eigenvalues <- pca$sdev^2
eigenvalues
eigenvectors <- pca$rotation
eigenvectors
# Calculate the cumulative score per principal component
per_var <- eigenvalues/sum(eigenvalues)
cumu_per_var <- cumsum(per_var)

# Print the cumulative proportion of variance explained by each principal component
cat("Cumulative proportion of variance explained by each PC:\n")
print(cumu_per_var)
# Create a new dataset with principal components as attributes
```

```
pca_data <- as.data.frame(pca$x)
# Choose PCs that provide at least cumulative score > 92%
chosed_pcs <- which(cumu_per_var > 0.92)
# Print the chosen PCs
cat("Chosen PCs:", chosed_pcs, "\n")
# Subset the transformed dataset to include only the chosen PCs
transform_data <- pca_data[, chosed_pcs]

#Looad Libraries
library(NbClust)
library(cluster)
library(factoextra)

set.seed(1234)

nb <- NbClust(transform_data, distance = "euclidean", min.nc = 2, max.nc = 10, method =
"kmeans")
cat("optimal number of clusters by nb:", nb$Best.nc, "\n")

# Elbow Method
fviz_nbclust(transform_data, kmeans, method = "wss")

# Run gap statistic method
# Method 1
fviz_nbclust(transform_data, kmeans, method = 'gap_stat')
# Method 2
gap_stat <- clusGap(transform_data, FUN = kmeans, nstart = 25,
                    K.max = 10, B = 50)
fviz_gap_stat(gap_stat)

# Silhoute Method
fviz_nbclust(transform_data, kmeans, method = 'silhouette')

# Perform k-means clustering on transformed data with k=2
kmeans_output <- kmeans(transform_data, 2)

# Print the k-means output
kmeans_output
# Print the centers of each cluster
kmeans_output$centers
# Print the cluster assignments for each observation
kmeans_output$cluster

#Between_cluster_sums_of_squares (BSS)
BSS <- sum(kmeans_output$size * dist(kmeans_output$centers)^2)
# (TSS) Total_sum_of_Squares
TSS <- sum(dist(transform_data)^2)
# Calculate the ratio of BSS and TSS
BSS_TSS_ratio <- BSS / TSS
cat("BSS/TSS ratio:", BSS_TSS_ratio, "\n")
```

```
#ploting new clustures
newKM = kmeans(transform_data,2)

#install.packages("ggfortify")
# Install the package

library(ggfortify) # Load the package
autoplot(newKM,transform_data,frame=TRUE)

# Create the silhouette plot
library(factoextra)
library(cluster)

sil_widths <- silhouette(kmeans_output$cluster, dist(transform_data))
avg_sil_width <- mean(sil_widths[, 2])
fviz_silhouette(sil_widths) + ggtitle(paste0("Silhouette Plot (Avg. Width = ",
round(avg_sil_width, 2), ")"))

# Compute the between-cluster sum of squares (BSS)
bssCH <- sum(kmeans_output$size * dist(kmeans_output$centers)^2)
bssCH
# Compute the within-cluster sum of squares (WSS)
wssCH <- sum(kmeans_output$withinss)
wssCH
# Compute the total sum of squares (TSS)
tssCH <- bssCH + wssCH
tssCH
# Compute the number of clusters
new_nclusters <- 2
new_nclusters

# Compute the Calinski-Harabasz index
# Assume kCH = 10
kCH <- 10
ch_index <- (bssCH / (new_nclusters - 1)) / (wssCH / (nrow(transform_data) - kCH))
# Print the result
cat("Calinski-Harabasz Index:", ch_index, "\n")

#importing libraries
library(readxl)
library(neuralnet)
library(ggplot2)
library(readxl)
#Load dataset
uow_consumption <-
read_excel("/Users/shanmugaratnammohanaranjan/Desktop/20200607/uow_consumption.xls
x")
View(uow_consumption)

# Rename columns
```

```
colnames(uow_consumption) <- c("date", "hour_18", "hour_19", "hour_20")

# Import required library
library(MASS)
library(dplyr)
uow_consumption <- uow_consumption %>% mutate(
  map_1 = lag(hour_20, 1),
  map_2 = lag(hour_20, 2),
  map_3 = lag(hour_20, 3),
  map_4 = lag(hour_20, 4),
  map_7 = lag(hour_20, 7)
) %>%
  na.omit()

#rnorm(uow_consumption)

#splitting dataset into to parts as instructed
#Train data
training_data <- head(uow_consumption, 380)
training_data
#Test data
testing_data <- tail(uow_consumption, nrow(uow_consumption) - 380)
testing_data

# Training of Normalized data
train_normalized <- as.data.frame(scale(training_data[-1]))
train_normalized

# Testing of Normalized Data
test_normalized <- as.data.frame(scale(testing_data[-1]))
test_normalized
# Set the column names of the test_normalized data frame
colnames(test_normalized) <- colnames(train_normalized)

# Install Reshap2
#install.packages("reshape2")

# Load Library
library(reshape2)
library(ggplot2)

# Define color palettes for the plots
# Train Data After Normalized
train_palette <- c("#E00F00", "#5344E9", "#080E73", "#F0C942", "#0B32B2", "#D09E00",
"#CC79A7", "#00BFFF")
# Print Colours
train_palette
# Test Data After Normalized
test_palette <- c("#007D02", "#D89E00", "#CBB9A7", "#F09F00", "#23DEE9", "#008773",
"#F00042", "#00BFFF")
```

```
# Print Colours
test_palette

# Display training normalized data with color
ggplot(melt(train_normalized), aes(x = variable, y = value, fill = variable)) +
  geom_boxplot() + ggtitle("Train Normalized Data") + xlab("Variables") +
  ylab("Normalized Values") + scale_fill_manual(values = train_palette)

# Display testing normalized data with color
ggplot(melt(test_normalized), aes(x = variable, y = value, fill = variable)) +
  geom_boxplot() +
  ggtitle("Test Normalized Data") + xlab("Variables") +
  ylab("Normalized Values") + scale_fill_manual(values = test_palette)

# Define a list of input vectors for different combinations of maps
input_vectors <- list(
c("map_1"),
c("map_1", "map_2"),
c("map_1", "map_2", "map_3"),
c("map_1", "map_2", "map_3", "map_4"), c("map_1", "map_7"),
c("map_1", "map_2", "map_7"),
c("map_1", "map_2", "map_3", "map_7"), c("map_1", "map_2", "map_3", "map_4",
"map_7")
)

# Load Neuralnet
library(neuralnet)

# a multilayer perceptron (MLP) neural network model using the neuralnet package
mlp_mod <- function(train_data, test_data, input_vars, hidden_structure) {
  # Define the formula for the MLP model using the input variables and the target variable of
hour_20
  formula <- paste("hour_20 ~", paste(input_vars, collapse = " + "))
  # Fit the MLP model using the neuralnet function with the training data and the specified
hidden layer structure
  nn <- neuralnet(as.formula(formula), train_data, hidden = hidden_structure)
  # Prepare the test data by extracting the input variables and renaming the columns to match
the training data
  test_matrix <- as.matrix(test_data[, input_vars, drop = FALSE])
  colnames(test_matrix) <- colnames(train_data[, input_vars, drop = FALSE])
  # Use the fitted MLP model to predict the hour_20 values for the test data
  predictions <- predict(nn, test_matrix)
  # Return a list containing the fitted model and the predicted values for the test data
  return(list(model = nn, predictions = predictions))
}

# Install Neuralnet
# install.packages("neuralnet")

# Load Neuralnet
```

```
library(neuralnet)

# Initialize an empty list to store the trained models
models <- list()
# Train a separate model for each input vector and store it in the models list
for (i in 1:length(input_vectors)) {
  models[[i]] <- mlp_mod(train_normalized,
                         test_normalized, input_vectors[[i]], c(5))
}

# Define a function to calculate evaluation metrics for a given set of actual and predicted
values
metrics_calcu <- function(actual, predicted) {
  rmse <- sqrt(mean((actual - predicted)^2))
  mae <- mean(abs(actual - predicted))
  mape <- mean(abs((actual - predicted) / actual)) * 100
  smape <- mean(abs(actual - predicted) / (abs(actual) + abs(predicted)) * 2) * 100
  return(list(RMSE = rmse, MAE = mae, MAPE = mape, sMAPE = smape))
}

# Initialize an empty list to store the evaluation metrics for each model
evaluation_metrics <- list()

# Calculate the evaluation metrics for each model and store them in the evaluation_metrics
list
for (i in 1:length(models)) {
  evaluation_metrics[[i]] <- metrics_calcu(test_normalized$hour_20, models[[i]]$predictions)
}

# Calculate the evaluation metrics for each model and store them in the evaluation_metrics
list
comparison_table <- data.frame(
  Model_Description = c("AR(1)", "AR(2)", "AR(3)", "AR(4)", "AR(1,7)", "AR(2,7)",
                        "AR(3,7)", "AR(4,7)"),
  RMSE = sapply(evaluation_metrics, function(x) x$RMSE),
  MAE = sapply(evaluation_metrics, function(x) x$MAE),
  MAPE = sapply(evaluation_metrics, function(x) x$MAPE),
  sMAPE = sapply(evaluation_metrics, function(x) x$sMAPE)
)

# Print
print(comparison_table)

# Efficiency comparison between one-hidden layer and two-hidden layer networks
model_1_hidden <- mlp_mod(train_normalized, test_normalized, c("map_1", "map_2",
"map_3", "map_7"), c(5))
#model_1_hidden
model_2_hidden <- mlp_mod(train_normalized, test_normalized, c("map_1", "map_2",
"map_3", "map_7"), c(3, 2))
#model_2_hidden
```

```
# Check the total number of weight parameters per network
num_weights_1_hidden <- sum(sapply(model_1_hidden$model$weights, length))
num_weights_1_hidden
num_weights_2_hidden <- sum(sapply(model_2_hidden$model$weights, length))
num_weights_2_hidden

# Print
cat("Total number of weight parameters for the one-hidden layer network:",
    num_weights_1_hidden, "\n")
cat("Total number of weight parameters for the two-hidden layer network:",
    num_weights_2_hidden, "\n")

# Method 2
# Applying Neural network concepts
# Load neuralnet package
library(neuralnet)

# Get column names of uow_consumption
n <- names(uow_consumption)

# Define formula for neural network input
f <- as.formula(paste("train_normalized ~",
                      paste(n[!n %in% "train_normalized"],
                            collapse = " + ")))

# Train neural network model with two hidden layers of 4 and 2 nodes, and linear output
nn <- neuralnet(train_normalized, data = uow_consumption,
                hidden = c(1, 2),
                linear.output = T)

# Plotting the graph
plot(nn)

# Add the 18th and 19th hour attributes to the input vectors
narx_input_vectors <- list(
  c("map_1", "hour_18", "hour_19"), # First input vector
  c("map_1", "map_2", "hour_18", "hour_19"),
  c("map_1", "map_2", "map_3", "hour_18", "hour_19"),
  c("map_1", "map_2", "map_3", "map_7", "hour_18", "hour_19"),
  c("map_1", "map_2", "map_3", "map_4", "map_7", "hour_18", "hour_19")  # Fifth input
vector
)

# Build NARX models
narx_models <- list()
for (i in 1:length(narx_input_vectors)) {
  narx_models[[i]] <- mlp_mod(train_normalized, test_normalized, narx_input_vectors[[i]],
c(5)) # Build the i-th NARX model
}
```

```
# Evaluate NARX models
narx_evaluation_metrics <- list()
for (i in 1:length(narx_models)) {
  narx_evaluation_metrics[[i]] <- metrics_calcu(test_normalized$hour_20,
narx_models[[i]]$predictions) # Evaluate the i-th NARX model
}

# Create a comparison table for NARX models
narx_comparison_table <- data.frame(
  # Model descriptions
Model_Description = c("NARX(1,18,19)", "NARX(2,18,19)", "NARX(3,18,19)",
"NARX(3,7,18,19)", "NARX(4,7,18,19)"),
# Root Mean Squared Error
  RMSE = sapply(narx_evaluation_metrics, function(x) x$RMSE),
# Mean Absolute Error
  MAE = sapply(narx_evaluation_metrics, function(x) x$MAE),
# Mean Absolute Percentage Error
  MAPE = sapply(narx_evaluation_metrics, function(x) x$MAPE),
# Symmetric Mean Absolute Percentage Error
  sMAPE = sapply(narx_evaluation_metrics, function(x) x$sMAPE))

# Print the comparison table
print(narx_comparison_table)

# De-normalize the predictions
denormalize <- function(x, min_value, max_value) {
  return(x * (max_value - min_value) + min_value)
}

# Find the index of the best model based on the RMSE
best_model_index <- which.min(sapply(evaluation_metrics, function(x) x$RMSE))
#best_model_index

# Get the best model
best_model <- models[[best_model_index]]
#best_model

# Get the predictions of the best model
best_model_predictions <- best_model$predictions
#best_model_predictions

# Get the minimum value of the target variable in the training data
min_value <- min(training_data$hour_20)
# Get the maximum value of the target variable in the training
max_value <- max(training_data$hour_20)

denormalized_predictions <- denormalize(best_model_predictions, min_value, max_value)

# Plot the predicted output vs. desired output
```

```
# Create the plot with custom settings
plot(testing_data$hour_20, type = "l", col = "#3399FF", lwd = 2,
  xlab = "Time", ylab = "Consumption (Hour 20)",
  # Add title to the plot
  main = "Comparison of Desired and Predicted Outputs",
  ylim = range(c(testing_data$hour_20, denormalized_predictions)),
  cex.main = 1.2, cex.lab = 1.1, cex.axis = 0.9)
lines(denormalized_predictions, col = "#FF6666", lwd = 2)

# Add a legend with custom settings
legend("topright", legend = c("Desired Output", "Predicted Output"),
       col = c("#3399FF", "#FF6666"), lty = 1, lwd = 2, cex = 0.9,
       box.col = NA, bg = "white", inset = 0.02)

###################################    END     ################################
```