INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology

Department of computing

(B.Eng.) Software Engineering

Module: 5DATA001C.2 Machine Learning and Data Mining

Module Leader: Nipuna Senanayake

INDIVIDUAL COURSEWORK

Student ID -: 20200649

Student UoW ID -: w1866979

Student Name -: R.M.S.J. Bandara

Table of Contents

Table of Figures

1. Clustering Part (Part 01)

1.1. 1st Subtask Objectives (Partitioning Clustering)

1.1.1. Pre – Processing Task (a)

The code below cleans up the "vehicles.xlsx" file. The dataset is first loaded using the "read_xlsx" method from the "readxl" package. The class attribute is then deleted by choosing just columns 2 through 19. The "remove_outliers" function is used to remove outliers from a single column of the dataset, while the "boxplot" function is used to plot the dataset before outliers are removed. The "apply" function is then used to execute the "remove_outliers" function on each column of the dataset. Last but not least, the "na.omit" function eliminates rows with NA numbers. A boxplot of the cleaned dataset demonstrates the outcome of eliminating the outliers.

Code part to remove outliers

```
16 library(readxl)
17 library(cluster)
18
19 # Read the XLSX file
20 vehicles_data <- read_xlsx("/Users/sandaru/Desktop/ML/CW/ML1_CW/vehicles.xlsx")
21
22 #Output variable removed (class attribute)
23 vehicles_data <- vehicles_data[, 2:19]
24 boxplot(vehicles_data, main = "Before Outlier Removal", outcol="red")
25 View(vehicles_data)
26
27 #Create a function to eliminate outliers from a single column using the boxplot method
28 remove_outliers <- function(x) {
29   bp <- boxplot.stats(x)$stats
30   x[x < bp[1] | x > bp[5]] <- NA
31   return(x)
32 }
33
34 #Apply the function to each data frame column
35 vehicles_data <- apply(vehicles_data, 2, remove_outliers)
36
37 #Eliminate any rows with empty values (remove)
38 vehicles_data <- na.omit(vehicles_data)
39 boxplot(vehicles_data, main = "After Outlier Removal", outcol="red")
40
```

Figure 1: code part for remove outliers

Below chart is a boxplot of before removing outliers and after removing outliers

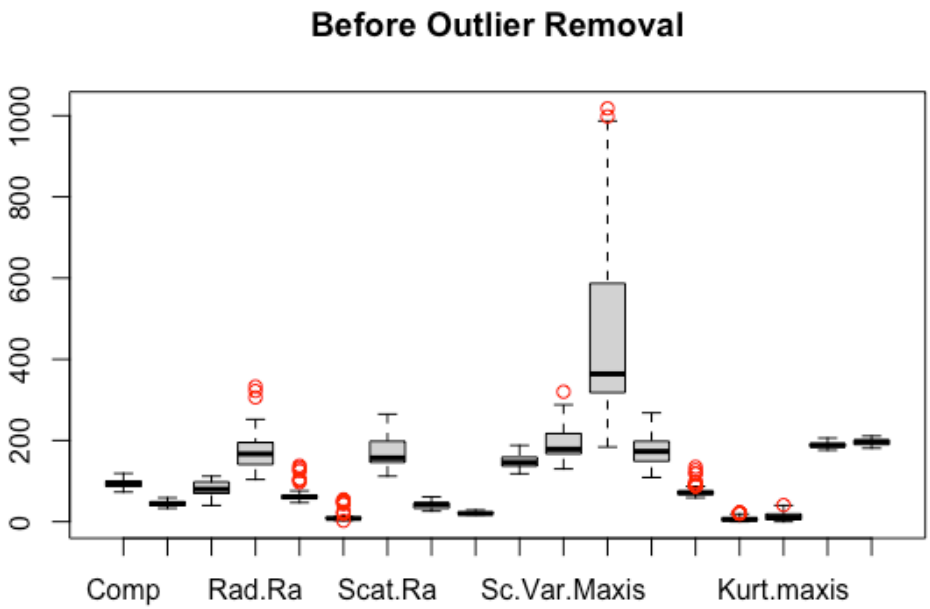


Figure 2: Before Outlier Removal

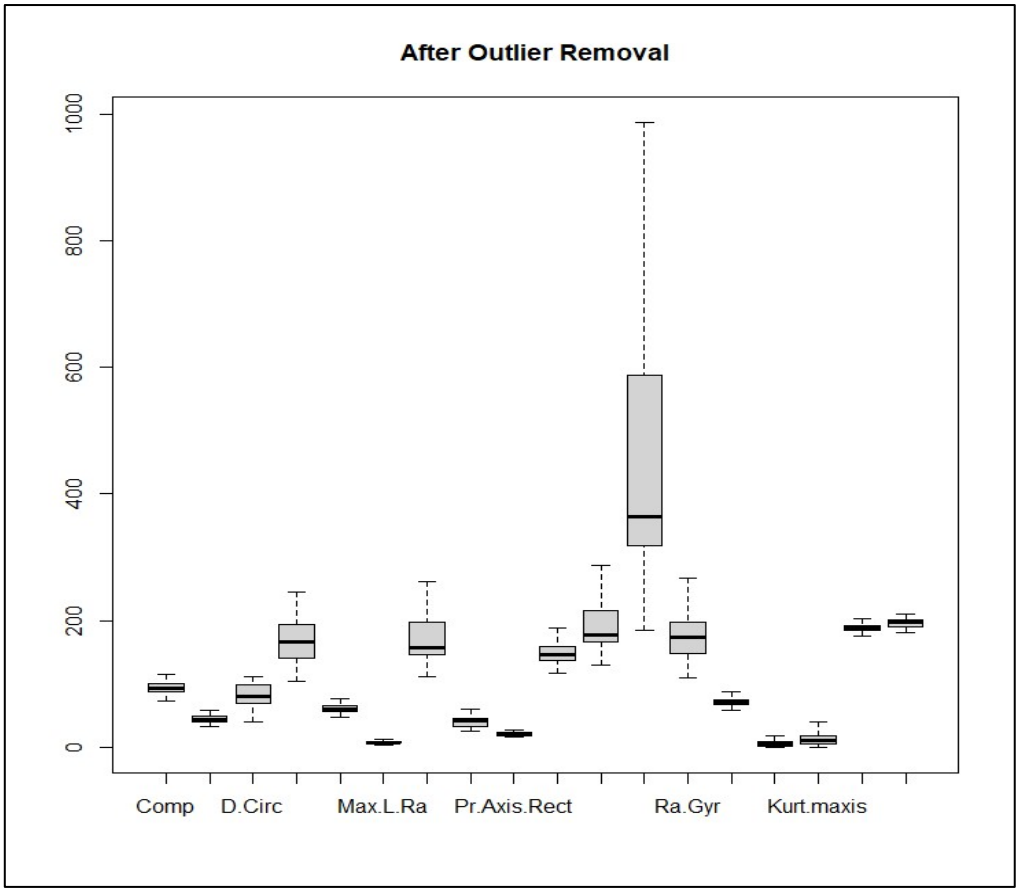


Figure 3: After Outlier Removal

Code part for scaling data set

This code scales the data from the "vehicles_data" dataset using the standardization approach. By subtracting the mean and dividing by the standard deviation of each column, the "scale ()" technique uniformizes the dataset. A boxplot depicts the distribution of the data before and after scaling, while the "head ()" method displays the top rows of the scaled dataset.

```
41  #Data scaling
42  scaled_vehicles_data <- scale(vehicles_data)
43  boxplot(scaled_vehicles_data, main = "Scale the data set")
44  head(scaled_vehicles_data)
45
```

Figure 4: code of scaling dataset

boxplot of dataset after removing outliers and scaling

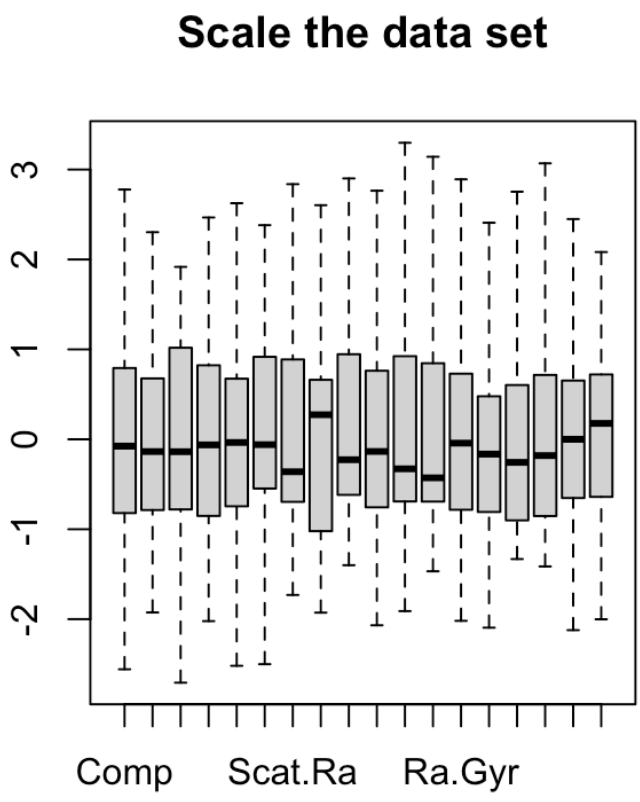


Figure 5: Scale dataset

1.1.2. Determine the number of cluster (b)

Determine the number of cluster centers via four "automated tools."

To calculate the number of clusters unutilized the techniques below.

- NBclust
- Elbow
- Gap statistics
- Silhouette method

1.1.2.1. NBclust

The following NBclust approach makes use of the Euclidean distance and the gap.

The optimal number of clusters in the supplied data set is determined by this code using the NbClust algorithm. Use the set. Seed function to ensure that the results are reproducible. Measurements of Euclidean distance are made, and K-means clustering is carried out. The index option is set to "all," which denotes that every conceivable index will be used to calculate the optimal number of clusters. The output will provide details on the suggested number of clusters based on each index, with a possible range of cluster numbers from 2 to 10.

```
45
46  set.seed(1234)
47  NBcluster <- NbClust(scaled_vehicles_data, min.nc = 2,max.nc = 10, method = "kmeans")
48  table(NBcluster$Best.n[1,])
49
```

Figure 6: NBclust

Look below. Three is the ideal cluster size.

```
> set.seed(1234)
> NBcluster <- NbClust(scaled_vehicles_data, min.nc = 2,max.nc = 10, method = "kmeans")
*** : 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:
* 9 proposed 2 as the best number of clusters
* 10 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 2 proposed 7 as the best number of clusters
* 2 proposed 9 as the best number of clusters

                   ***** Conclusion *****

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


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

Figure 7:Output Best number of clusters using NBclust

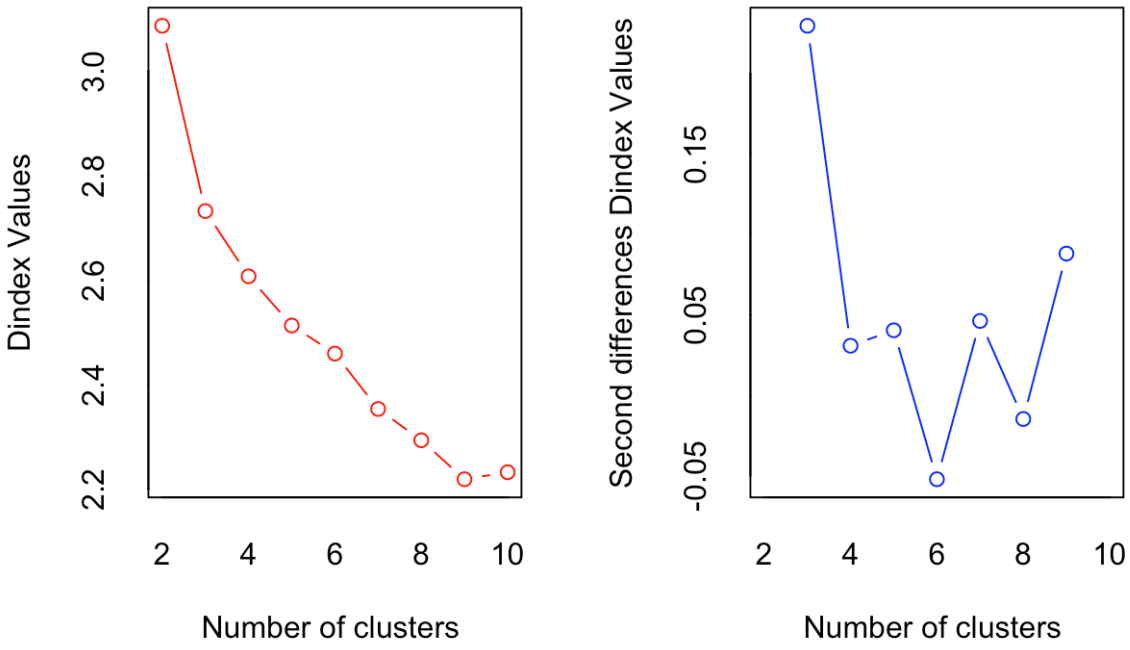


Figure 8:Graph:Number of clusters by NBClust

1.1.2.2. Elbow

“Elbow function” code section.

The main idea of the elbow approach is to exactly count the number of clusters for a given data set by locating the "elbow point" in the graph where the elbow shape is formed. Here, this is done by using the Within-cluster Sum of Squares (WSS) calculation. The calculation process was repeated ten times by this software, with each round ending with the recording of the WSS value.

```
52 #elbow_method
53 fviz_nbclust(scaled_vehicles_data,kmeans,method = "wss")
54
```

Figure 9:Code section for Elbow function

Output

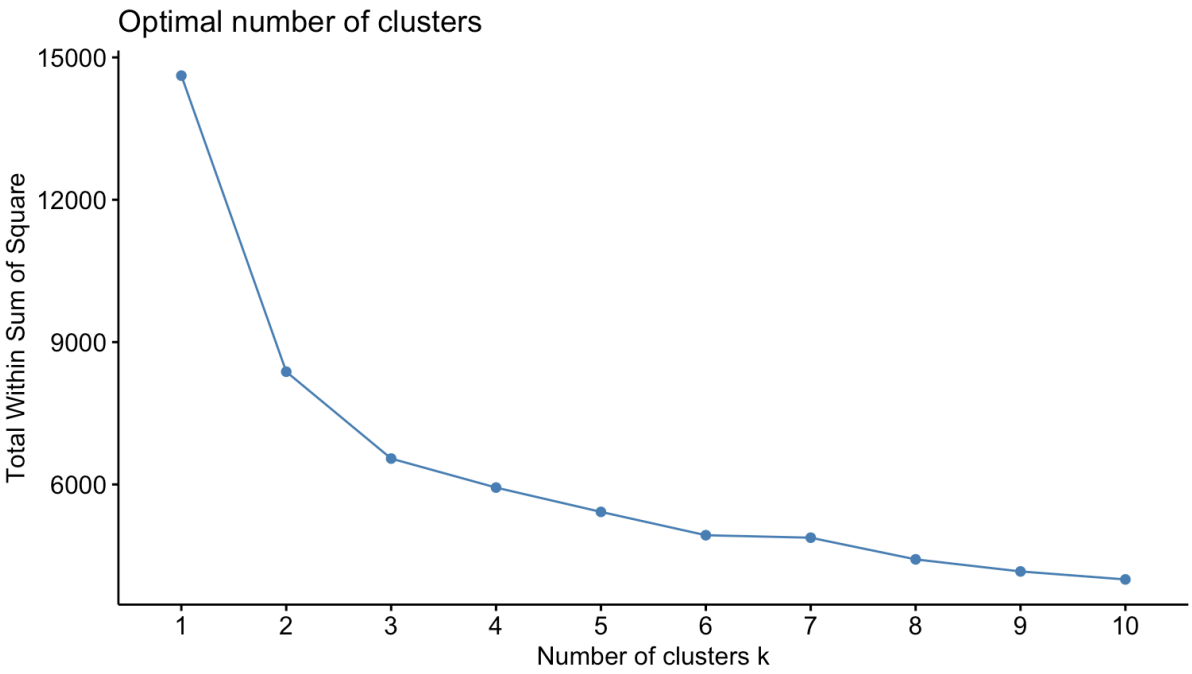


Figure 10:Elbow clustering curve

1.1.2.3. Sinlhouette

“Sinlhouette Function” code section.

The silhouette method can be used to evaluate how well a dataset is clustered. The similarity between each vehicle and the other vehicles in its cluster and the other vehicles in other clusters is used to determine the Silhouette coefficient for each vehicle. Then, in order to offer a metric for the general clustering quality, the average silhouette coefficient for each cluster is calculated. The Silhouette technique can be used to find the sweet spot by averaging the silhouette coefficient for different cluster sizes. If the average is high, the grouping is likely accurate; if it is low, the clustering needs improvement.

```
55  #silhouette_method
56  fviz_nbclust(scaled_vehicles_data,kmeans,method = "silhouette")
57
```

Figure 11: Code section for Sinlhouette Function

Output

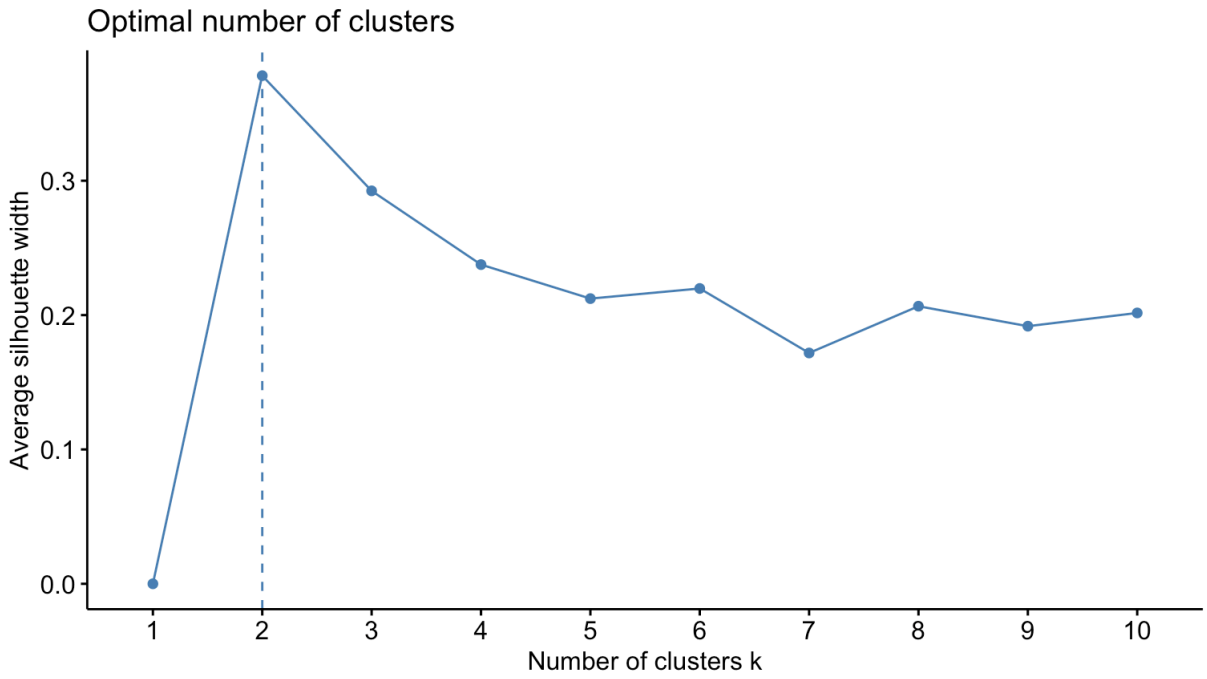


Figure 12:silhouette clustering curve

After running the code above, it was found that there should be two groups (K=2).

1.1.2.4. Gap Statistics

“Gap statistics” code section.

In the gap static approach, which compares the within-cluster dispersion of a clustering solution, data distributions with similar features but no obvious clustering trend are utilized as references. Using the gap static technique, the ideal number of clusters in a dataset can be determined. To provide a better fit for the data, other techniques and domain knowledge should be used in addition to clustering.

```
58  #gap_static_method
59  fviz_nbclust(scaled_vehicles_data,kmeans,method = "gap_stat")
60
```

Figure 13:code section for Gap statistics

Output

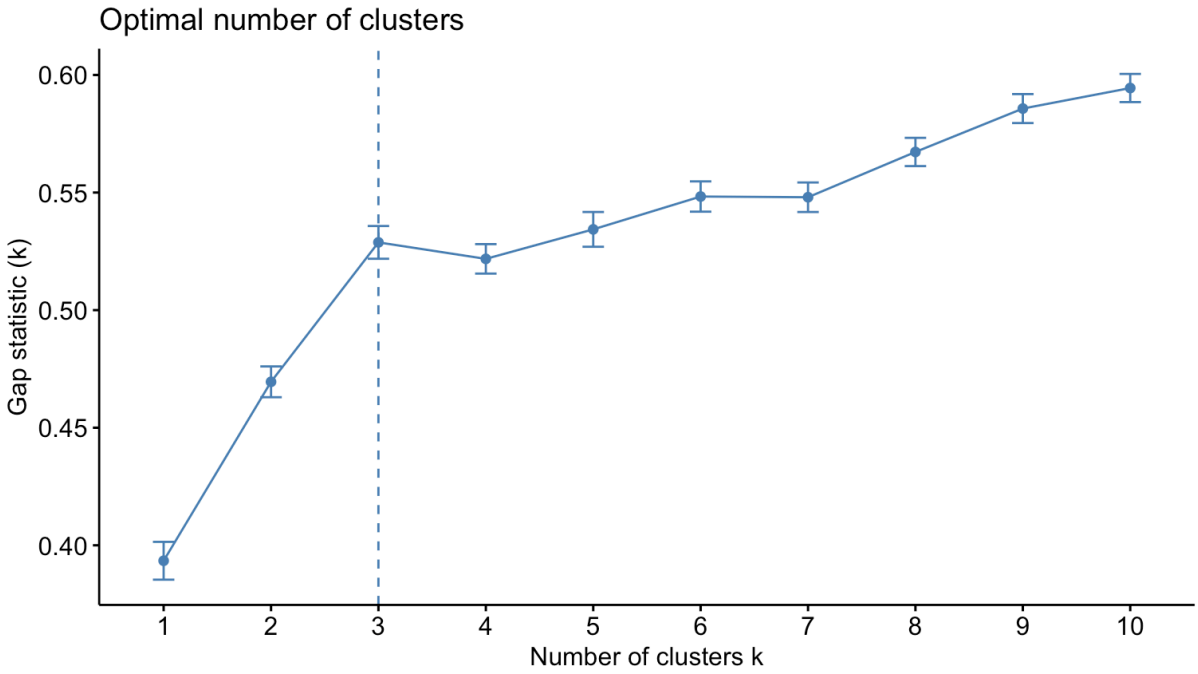


Figure 14: Gap Statistics curve

1.1.3. K-means cluster investigation (c)

K-means clustering uses the following code section when the cluster number is 2.

```
62  # kmean %2 clusters
63  k2 <-kmeans(scaled_vehicles_data, 2)
64  k2
65  autoplot(k2,scaled_vehicles_data,frame=TRUE)
66
```

Figure 15: Kmeans clustering for 2 centers

Using this approach, a preprocessed, scaled dataset with two clusters is clustered using K-means. While kmeans(scaled_vehicles_data, 2) stores the results in the variable k2, autoplot(k2, scaled_vehicles_data, frame=TRUE) uses the ggplot2 package to plot the results of the k-means clustering. The "frame=TRUE" argument indicates that a bounding box around the plot should be generated. The produced figure shows the two clusters in different colors and each point corresponds to a row in the original dataset.

Output

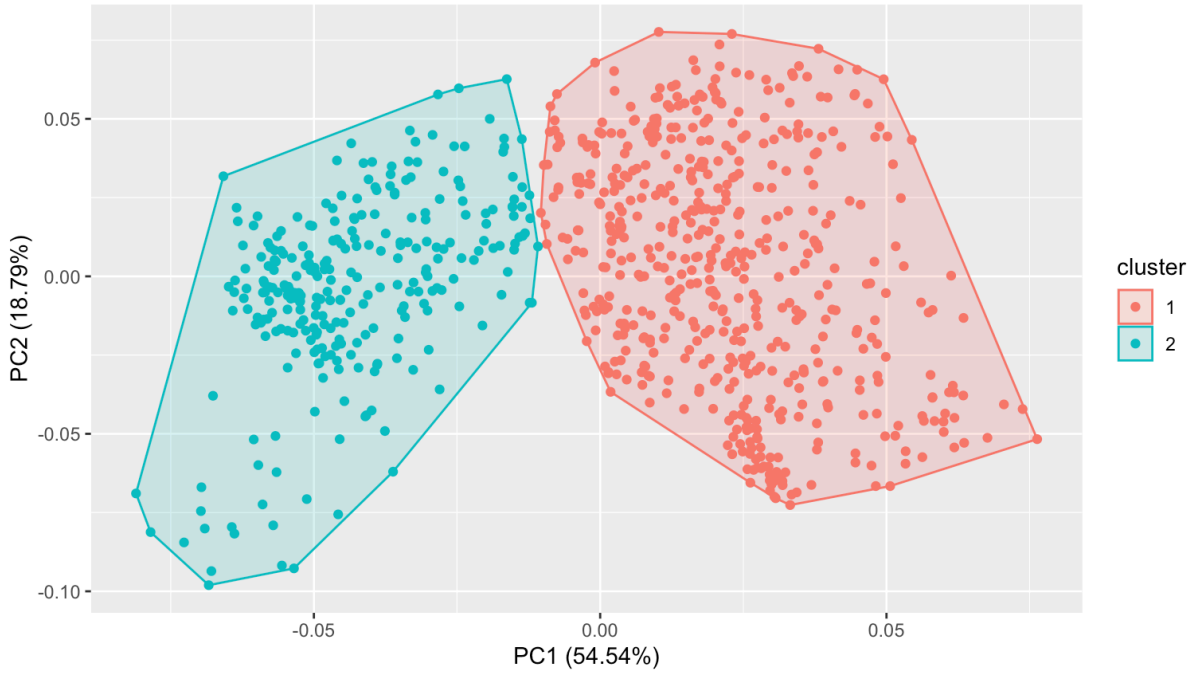


Figure 16:K2 clustering output plot

When k = 2, determine the ratio between the total squares and the square clusters.

```r
#Extract relevant data when k = 2.
# Cluster centers
clus_center <- k2$centers

#clustered outcomes
clus_assi <- k2$cluster

#Calculation BSS over TSS when k = 2
BSSk2 <- k2$betweenss #BSS
WSSk2 <- k2$tot.withinss #WSS
TSSk2 <- BSSk2 + WSSk2 #TSS

# BSS/TSS ratio
BSS_TSS_ratiok2 <- BSSk2 / TSSk2

#Explained variance as a percentage
percent_vark2 <- round(BSS_TSS_ratiok2 * 100, 3)

#Output results
cat("Cluster centers:\n", clus_center, "\n\n")
cat("Cluster assignments:\n", clus_assi, "\n\n")
cat("BSS/TSS ratio: ", round(BSS_TSS_ratiok2, 3), "\n\n")
cat("BSS: ", round(BSSk2, 3), "\n\n")
cat("WSS: ", round(WSSk2, 3), "\n\n")
cat("Explained variance as a percentage: ", percent_vark2, "%\n")

```

Figure 17: Calculation for K2

```
>
> #Output results
> cat("Cluster centers:\n", clus_center, "\n\n")
Cluster centers:
 -0.5581484 1.074135 -0.5746465 1.105885 -0.598326 1.151455 -0.5409894 1.041113 -0.1354837 0.
2607329 -0.336854 0.6482622 -0.636507 1.224932 0.606645 -1.167464 -0.6385458 1.228856 -0.5309
785 1.021847 -0.6175342 1.18842 -0.6400189 1.231691 -0.5270382 1.014264 0.04804773 -0.0924659
5 -0.05799499 0.1116091 -0.1198347 0.2306172 -0.02909619 0.05599447 -0.1073133 0.2065202

> cat("Cluster assignments:\n", clus_assi, "\n\n")
Cluster assignments:
 1 1 2 1 2 1 1 1 2 1 1 1 1 2 2 1 1 2 2 1 1 1 1 2 1 1 2 2 1 1 1 1 2 1 1 1 2 1 2 1 1 1 1 1 1 1
1 1 2 1 2 1 2 1 2 1 2 1 1 1 2 1 1 2 1 2 2 2 1 1 1 2 1 1 2 1 2 1 1 1 1 1 1 1 2 1 2 1 1 2 1 1 2
1 1 1 1 2 2 2 1 1 2 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 2
1 2 1 2 1 1 1 1 1 2 1 1 2 2 1 2 1 1 2 2 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 2 1 1 1
1 1 2 2 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 2 1 2 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 1
2 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 2 1 1 1 1 2 1
2 1 1 1 2 1 1 1 1 1 2 2 2 2 2 1 1 2 1 1 1 2 1 2 2 2 1 2 1 1 2 1 1 1 1 2 2 1 2 2 1 2 1 1 1 1 1
2 2 2 1 1 1 2 1 1 1 2 1 1 2 1 2 2 2 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 2 1 1 2 1 2 1 1 1
2 1 2 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 2 1 2 2 2 1 1 2 1 1 1 2 2 2 1
1 2 2 1 2 2 2 1 1 1 1 2 1 1 1 2 1 1 2 1 1 2 1 1 2 2 1 1 2 2 2 1 2 2 1 2 1 2 2 1 1 1 1 2 1 1
2 2 1 1 2 2 1 2 1 2 2 2 1 1 2 2 1 1 2 1 1 2 1 1 1 2 1 1 1 1 1 2 1 2 2 1 1 2 2 2 1 1 1 2 1 2 2
1 1 1 1 1 1 1 1 2 1 1 1 1 2 2 2 1 1 2 2 1 2 1 1 2 2 1 1 1 2 1 1 1 2 2 2 2 1 1 1 1 2 2 2 1 2 1
1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 2 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 1 2 1 1 2 1 1 2 1 2 1 1
1 1 2 2 2 1 2 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 2 1 2 1 1 1 1 2 2 1 1 2 1 2 1 1 2 1 2 1 1 1 1 1 2
1 1 1 2 1 2 1 1 2 1 2 1 1 1 1 1 2 2 1 1 2 2 1 1 1 2 2 2 2 1 2 1 1 2 2 1 2 1 2 1 1 2 1 1 2 2 2
1 2 1 1 2 2 2 1 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 2 2 1 1 1 2 1 1 2 2 1 1 1 2 2 2 1
2 1 2 2 1 1 2 1 2 1 1 1 2 2 1 1 2 1 1 1 1 2 2 1 1 2 1 1 2 1 1 1 1 1 1 2 2 2 1 2 1 2 2 1 2 2 1 1
1 1 2 1 1 1 1 1 1 1 1 1 2 1 1

> cat("BSS/TSS ratio: ", round(BSS_TSS_ratiok2, 3), "\n\n")
BSS/TSS ratio:  0.427

> cat("BSS: ", round(BSSk2, 3), "\n\n")
BSS:  6239.904

> cat("WSS: ", round(WSSk2, 3), "\n\n")
WSS:  8376.096

> cat("Explained variance as a percentage: ", percent_vark2, "%\n")
Explained variance as a percentage:  42.692 %
```

Figure 18: K2 output

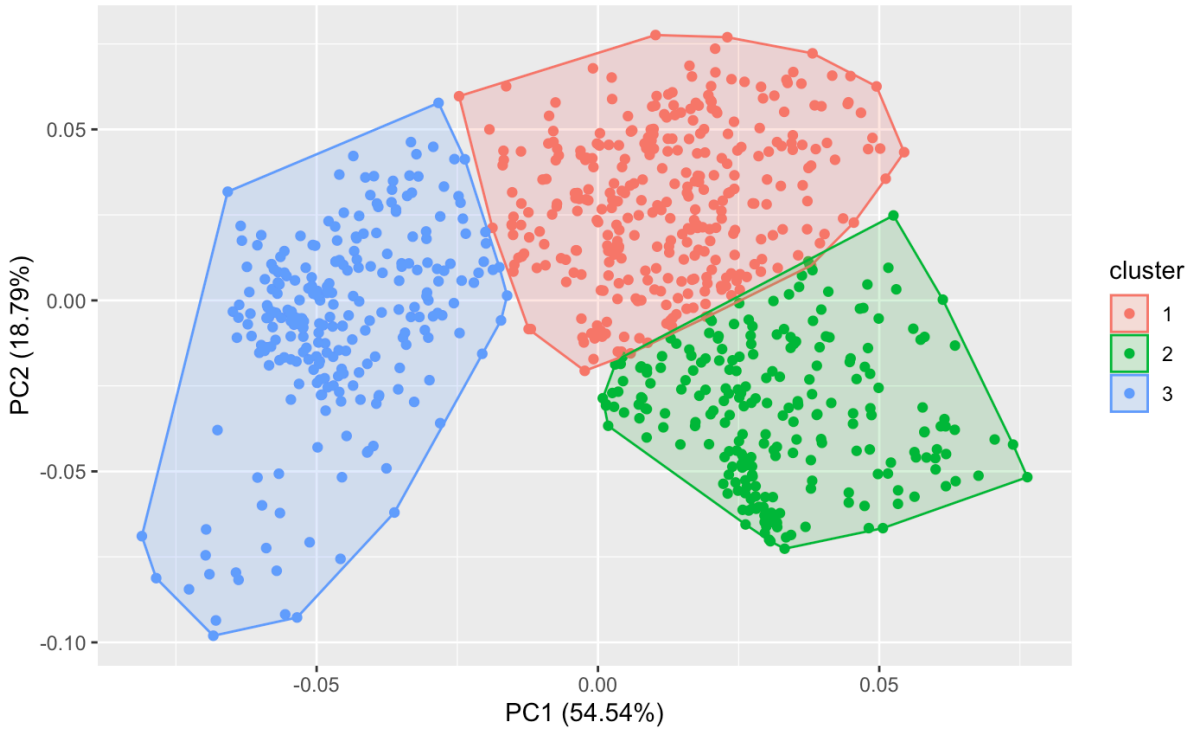
K-means clustering uses the following code section when the cluster number is 3.

```
93  #3 clusters
94  k3 <-kmeans(scaled_vehicles_data, 3)
95  k3
96  autoplot(k3,scaled_vehicles_data,frame=TRUE)
97
```

Figure 19: Kmeans clustering for three centers

On a scaled automobile dataset with three clusters, this application does cluster analysis using the k-means technique. Kmeans() is used for the clustering analysis, and the output clusters are stored in the "k3" variable. Using Autoplot(), the clustering result is represented as a scatter plot, with each point's color according to its cluster. We'll use "scaled_vehicles_data" as our input data and "frame=TRUE" to indicate the beginning and end of each cluster in this example.

Output



When k = 3, determine the ratio between the total squares and the square clusters.

```r
#Extract relevant data when k = 3
# Cluster centers
clus_center <- k3$centers

#clustered outcomes
clus_assi <- k3$cluster

#Calculation BSS over TSS when k = 3
BSSk3 <- k3$betweenss #BSS
WSSk3 <- k3$tot.withinss #WSS
TSSk3 <- BSSk3 + WSSk3 #TSS

# BSS/TSS ratio
BSS_TSS_ratiok3 <- BSSk3 / TSSk3

#Explained variance as a percentage
percent_vark3 <- round(BSS_TSS_ratiok3 * 100, 3)

# Output results
cat("Cluster centers:\n", clus_center, "\n\n")
cat("Cluster assignments:\n", cluster_assignments, "\n\n")
cat("BSS/TSS ratio: ", round(BSS_TSS_ratiok3, 3), "\n\n")
cat("BSS: ", round(BSSk3, 3), "\n\n")
cat("WSS: ", round(WSSk3, 3), "\n\n")
cat("Explained variance as a percentage: ", percent_vark3,"%\n")
```

Figure 20: K3 clustering output plot

Output

```
> # Output results
> cat("Cluster centers:\n", clus_center, "\n\n")
Cluster centers:
 -0.2285316 -0.9423859 1.163264 -0.5301921 -0.5474136 1.189408 -0.2929582 -0.9181953 1.224256 0.001398735 -1.145655 1.053282 0.3660925 -0.7525
338 0.2198741 -0.1699273 -0.5350742 0.7124052 -0.4491392 -0.7945601 1.312257 0.3136041 0.8899092 -1.224891 -0.4769669 -0.7607372 1.317075 -0.4
987139 -0.5059132 1.110503 -0.4002859 -0.8128739 1.265981 -0.4561852 -0.7977349 1.324288 -0.5520307 -0.4155658 1.096209 -0.6831445 0.992033 -
0.0306539 -0.03494371 -0.1014964 0.1386373 -0.02711354 -0.2608554 0.2752784 0.7756077 -1.062805 -0.02367658 0.6807868 -1.128556 0.1594321

> #cat("Cluster assignments:\n", cluster_assignments, "\n\n")
> cat("Cluster assignments:\n", clus_assi, "\n\n")
Cluster assignments:
 1 1 3 1 3 1 1 1 1 1 1 1 1 3 2 1 3 3 2 2 1 1 3 1 2 3 3 2 1 1 1 3 1 1 2 3 2 3 2 2 1 2 2 1 2 1 3 1 3 1 1 2 3 2 3 2 2 2 1 2 2 3 1 3 3 3 1 2 1
3 1 2 3 2 3 1 2 1 1 2 1 2 3 1 3 1 2 3 2 2 3 2 1 1 2 3 3 3 2 2 1 1 1 2 2 1 3 3 2 1 2 2 1 1 1 2 1 3 3 1 2 3 2 1 2 1 1 2 3 2 1 3 1 1 1 1 3 1 1 3
1 3 1 2 1 1 2 3 1 1 3 3 1 3 2 2 3 3 1 3 1 1 1 1 1 2 3 2 1 2 3 1 1 1 3 1 1 1 3 1 2 3 2 2 2 1 1 3 3 1 1 1 2 2 3 1 1 1 3 2 1 2 3 2 1 3 2 3 2 2 1
3 1 3 2 2 2 3 1 2 1 2 3 2 1 1 2 3 2 2 1 1 3 2 2 3 2 1 1 3 1 1 3 3 2 1 1 1 3 2 2 1 1 2 2 1 1 1 3 1 2 2 3 1 1 2 2 3 2 1 1 2 3 2 2 1 1 3 1 3 2 1
1 3 1 1 1 2 1 3 3 3 3 3 2 1 3 2 2 2 1 2 3 3 3 2 3 1 2 3 2 1 1 1 3 3 2 3 3 2 3 1 1 1 2 2 3 3 3 1 1 1 3 2 1 2 1 1 1 3 1 3 3 3 1 1 2 3 1 2 2 1 1
1 1 1 2 3 3 2 2 3 2 2 3 1 1 1 1 3 2 1 1 1 1 3 1 1 1 1 3 1 3 1 2 2 1 1 1 2 2 1 2 3 1 1 2 1 2 3 1 2 1 1 3 1 3 1 3 3 2 2 3 1 2 2 1 3 3 2 2 3 3 2
3 3 3 1 1 1 1 1 3 2 2 1 3 1 1 3 1 2 3 2 2 3 3 1 2 3 3 3 2 3 3 1 1 2 3 3 1 1 2 2 3 1 2 3 3 1 2 3 3 1 3 2 3 3 3 2 2 3 3 1 1 3 2 1 3 1 2 2 3 2 1
1 2 1 3 1 3 3 1 2 1 3 3 2 2 1 3 1 3 3 1 1 1 1 2 2 1 1 3 2 2 1 2 3 1 3 2 2 3 3 1 3 1 1 1 3 1 2 1 3 1 1 2 3 3 3 3 1 2 2 2 3 3 3 1 3 2 1 3 2 2 2
1 2 1 1 1 1 1 1 3 1 1 3 1 1 1 2 3 2 2 1 2 1 1 2 2 3 3 2 1 2 3 1 1 3 1 2 3 2 3 2 2 1 2 1 3 3 1 3 1 1 2 1 2 3 1 3 2 1 1 1 2 2 1 3 1 3 2 1 1 1
1 3 1 2 3 1 3 1 1 3 2 3 2 1 1 1 2 3 1 2 1 3 2 3 1 1 3 2 1 2 1 1 2 1 3 3 1 1 3 3 1 2 1 3 3 3 3 1 3 1 1 3 3 1 3 1 3 1 2 3 1 2 3 3 3 1 3 2 2 3 3
3 1 3 1 1 3 1 2 1 2 1 3 1 2 1 1 1 2 3 2 2 2 3 2 3 3 2 1 1 3 1 2 3 3 2 1 1 3 3 3 2 3 1 3 3 2 2 3 2 3 1 2 1 3 3 1 2 3 1 1 2 1 1 3 2 1 3 2 2 3 2
1 2 2 1 3 3 1 2 3 1 3 3 2 1 3 2 2 1 1 3 2 2 2 1 1 1 1 1 1 3 1 2

> cat("BSS/TSS ratio: ", round(BSS_TSS_ratiok3, 3), "\n\n")
BSS/TSS ratio:  0.552

> cat("BSS: ", round(BSSk3, 3), "\n\n")
BSS:  8070.492

> cat("WSS: ", round(WSSk3, 3), "\n\n")
WSS:  6545.508

> cat("Explained variance as a percentage: ", percent_vark3,"%\n")
Explained variance as a percentage:  55.217 %
```

Figure 21: K3 output

1.1.4. Code section for silhouette plot (d)

When Fit k-means model with k=2

```
125  #Fit k-means model with k=2
126  k <- 2
127  kmeans_model <- kmeans(scaled_vehicles_data, centers = k, nstart = 25)
128
129  #Create a silhouette plot
130  silhouette_plot <- silhouette(kmeans_model$cluster, dist(scaled_vehicles_data))
131
132  #Compute the average silhouette's width
133  avg_sil_width <- mean(silhouette_plot[, 3])
134
135  #Plot the silhouette plot
136  plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
137       xlab = "Silhouette Width", ylab = "Cluster", border = NA)
138
139  #As a vertical line, add the average silhouette width
140  abline(v = avg_sil_width, lty = 2, lwd =2,col="red")
```

Figure 22: K2 code section for silhouette plot

This program does a silhouette analysis to determine the appropriate k-means cluster size. The scaled vehicle dataset is subjected to a k-means clustering technique using the "kmeans" function, with the number of clusters set to 2. You can change the "nstart" option to choose how many random beginnings to use to find the best initial cluster centers. The silhouette width of each observation,

which is determined by the "silhouette" function, quantifies the degree of cluster membership of each observation. To make it simpler to see the quality of the clustering, the "abline" function inserts a vertical line at the location of the average silhouette width. The silhouette width for each observation and the total silhouette width for the clustering solution are plotted.

Output

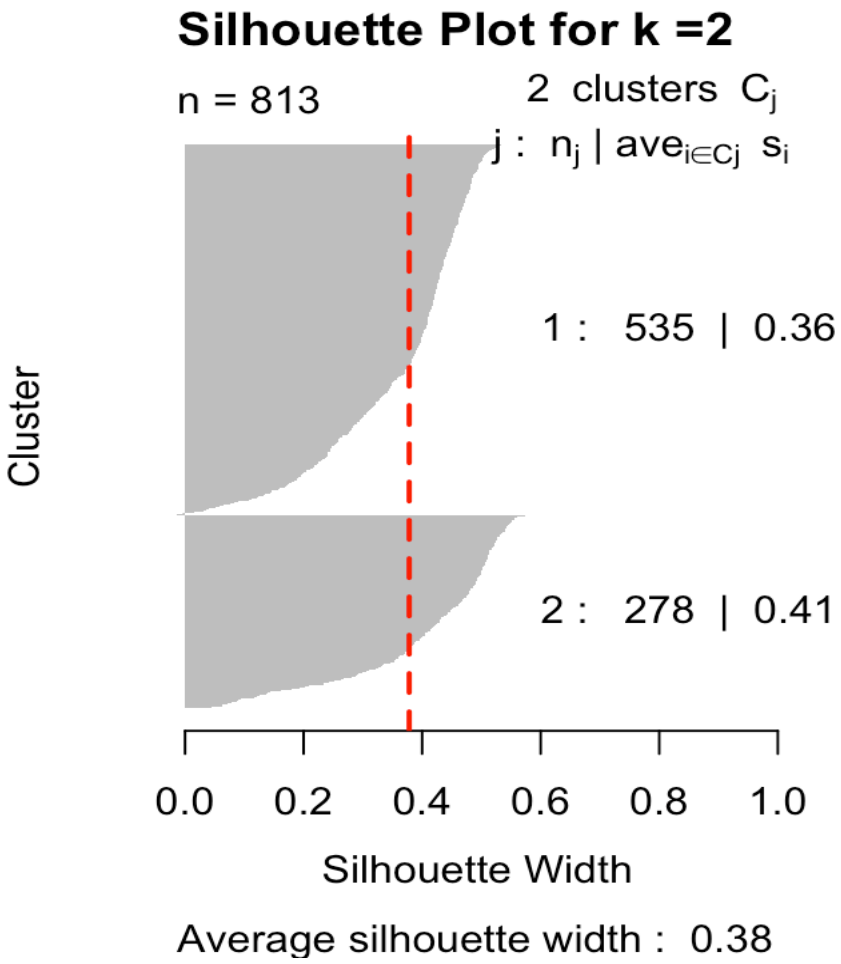


Figure 23: Silhouette plot for k2

When Fit k-means model with k=3

```
#Fit k-means model with k=3
k <- 3
kmeans_model <- kmeans(scaled_vehicles_data, centers = k, nstart = 25)

#Generate silhouette plot
silhouette_plot <- silhouette(kmeans_model$cluster, dist(scaled_vehicles_data))

#Calculate average silhouette width
avg_sil_width <- mean(silhouette_plot[, 3])

#Plot the silhouette plot
plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
     xlab = "Silhouette Width", ylab = "Cluster", border = NA)

#Add average silhouette width as vertical line
abline(v = avg_sil_width, lty = 2, lwd =2,col="red")

```

Figure 24: k2 code section for silhouette plot

The "scaled_vehicles_data" dataset is grouped in this code using the k-means algorithm. An evaluation of a silhouette plot's quality is then made of the grouping results. The notation "k - 3" indicates that three should be subtracted from the intended number of clusters when adjusting the number of clusters: "kmeans_model - kmeans (scaled_vehicles_data, centres = k, nstart = 25)", "silhouette_plot - silhouette(kmeans_model$cluster, dist(scaled_vehicles_data)"), "avg_sil_width

Output

Figure 25: silhouette plot k2

1.2. 2nd Subtask Objectives

1.2.1. Transform the dataset using PCA (e)

A statistical method for reducing dimensionality in data analysis is PCA (Principal Component Analysis). It identifies a brand-new set of variables, referred to as principal components, that fully encapsulates the data's variance. It is frequently used to show high-dimensional data in a lower-dimensional space and for feature extraction. PCA can streamline data analysis, enhance model performance, and support the discovery of underlying patterns and correlations in the data by lowering the number of variables.

In these situations, the amount of variance we wish to retain in the redesigned dataset will determine how many primary components to preserve. Because it strikes a reasonable compromise between lowering the dimensionality of the dataset and retaining enough variance to faithfully represent the original data, we decided to utilize a cutoff of 92% in this instance. We also visually examine the eigenvalue scree plot to locate the elbow point beyond which the eigenvalues start to flatten out, indicating that the variance explained by extra-large components is not particularly significant.

```
#part_02
pca <- prcomp(scaled_vehicles_data)

#The eigenvalues and eigenvectors in print
print(summary(pca))

#Calculate the total score for each of the principal components (PC)
pca_var <- pca$sdev^2
pca_var_prop <- pca_var / sum(pca_var)
pca_var_cumprop <- cumsum(pca_var_prop)

#Plot cumulative score per PC
plot(pca_var_cumprop, xlab = "Principal Component", ylab = "Cumulative Proportion of Variance Explained",
     ylim = c(0, 1), type = "b")

#Convert an existing dataset and add attributes that correspond to the primary components
pca_trans <- predict(pca, newdata = scaled_vehicles_data)

#Choose PCs that provide at least cumulative score > 92%
selected_pcs <- which(pca_var_cumprop > 0.92)
transformed_data <- pca_trans[, selected_pcs]
```

Figure 26: Code section for PCA

Output

```
> #part_02
> pca <- prcomp(scaled_vehicles_data)
>
> #The eigenvalues and eigenvectors in print
> print(summary(pca))
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11
Standard deviation     3.1332 1.8391 1.10005 1.06528 0.94325 0.80893 0.56552 0.47557 0.33278 0.27352 0.24146
Proportion of Variance 0.5454 0.1879 0.06723 0.06305 0.04943 0.03635 0.01777 0.01256 0.00615 0.00416 0.00324
Cumulative Proportion  0.5454 0.7333 0.80051 0.86355 0.91298 0.94934 0.96710 0.97967 0.98582 0.98998 0.99322
                          PC12    PC13    PC14    PC15    PC16    PC17    PC18
Standard deviation     0.20047 0.16530 0.14512 0.12420 0.10846 0.07754 0.01868
Proportion of Variance 0.00223 0.00152 0.00117 0.00086 0.00065 0.00033 0.00002
Cumulative Proportion  0.99545 0.99697 0.99814 0.99899 0.99965 0.99998 1.00000
```

Figure 27: PCA Output

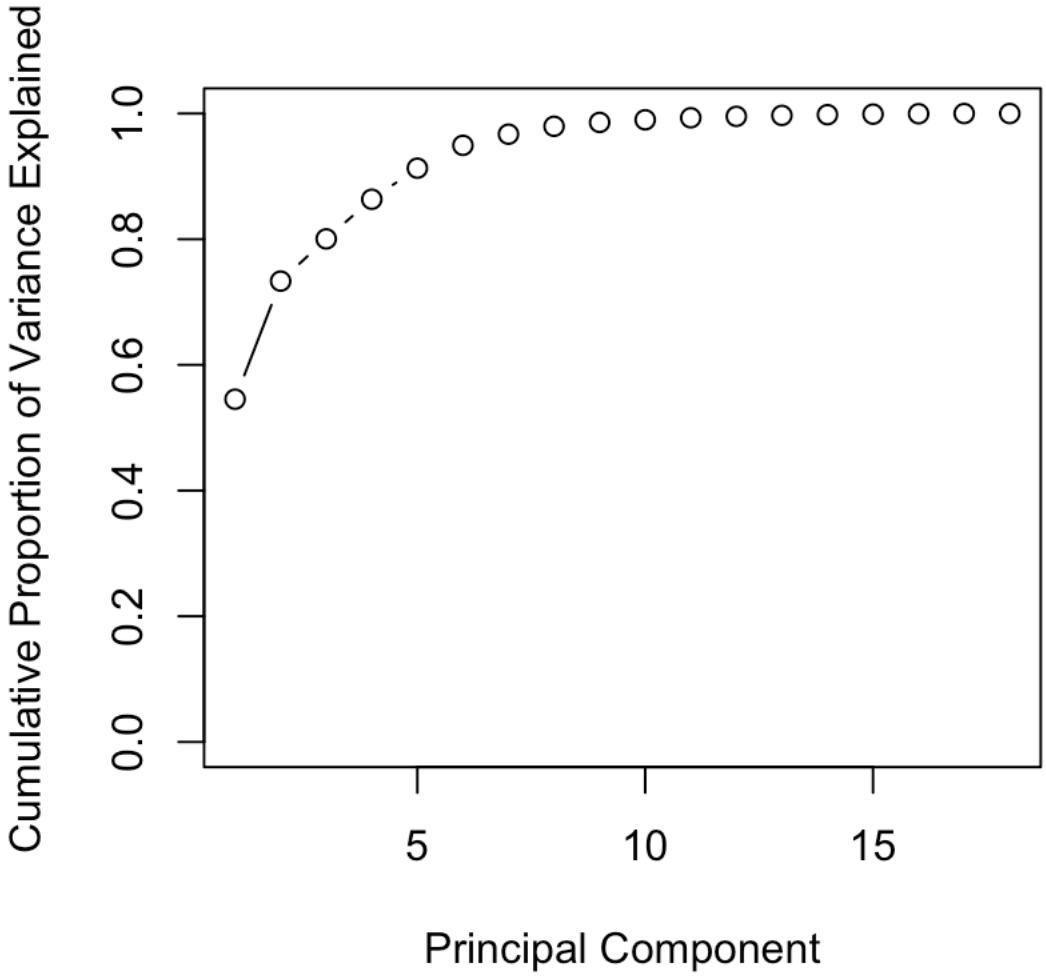


Figure 28: PCA graph

1.2.2. Determine the number of clusters in the transformed dataset. (f)

1.2.2.1. NBclust

The methods NbClust, Elbow, the Gap static, and the Sillhoutte methods are used to count the number of clusters in the altered data set.

```
184 set.seed(1234)
185 NBcluster <- NbClust(transformed_data, min.nc = 2,max.nc = 10, method = "kmeans")
186 table(NBcluster$Best.n[1,])
```

Figure 29: Performing cluster analysis to determine optimal number of clusters

This code snippet applies the k-means algorithm to some altered data in order to do a clustering analysis. The random seed is assigned to a fixed number (1234 in this case) via the function set.seed(1234). The NBcluster - NbClust function from the NbClust package utilizes the k-means algorithm to calculate the ideal number of clusters in the data. The parameters are: transformed_data, min.nc = 2, max.nc = 10, and method = "kmeans". The function table(NBcluster$Best.n[1,]) generates a table listing the frequency of each value of "Best.n" (the ideal number of clusters) in the results of the "NbClust" function. Using the kmeans algorithm, this code may be used to find the ideal number of clusters in a set of data and to display the frequency of various ideal cluster numbers for various clustering criteria.

Output

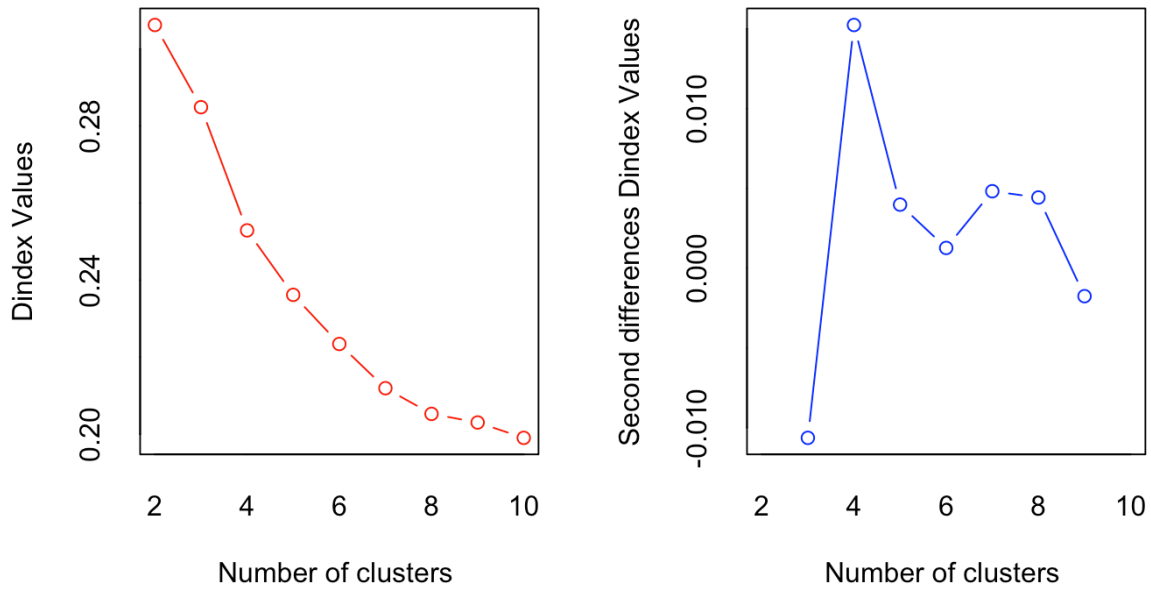


Figure 30: Plot showing the optimal number of clusters using NbClust method

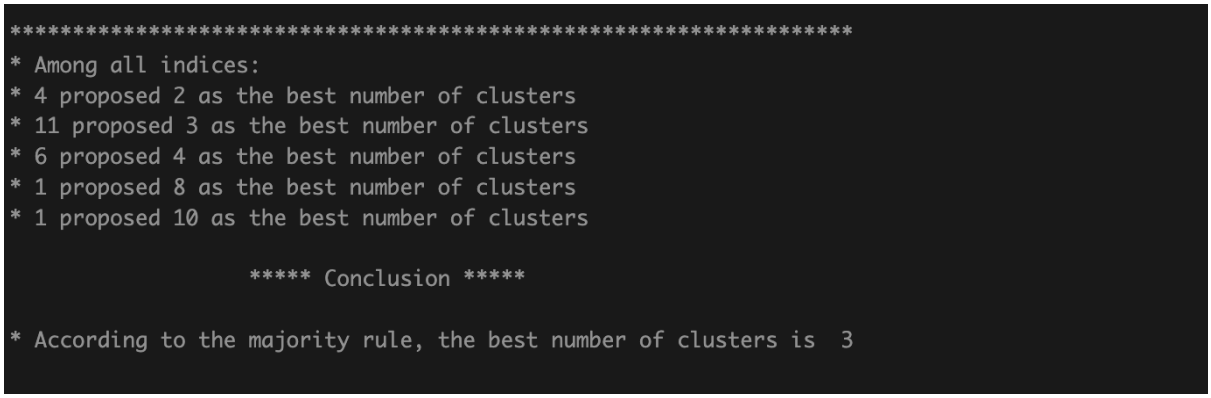


Figure 31: Output

1.2.2.2. Elbow

Based on the WSS map produced from the altered PCA dataset, the elbow approach recommends that three clusters are the ideal number.

```
190 #elbow_method
191 fviz_nbclust(transformed_data,kmeans,method = "wss")
192
```

Figure 32: plotting optimal cluster number using the elbow method

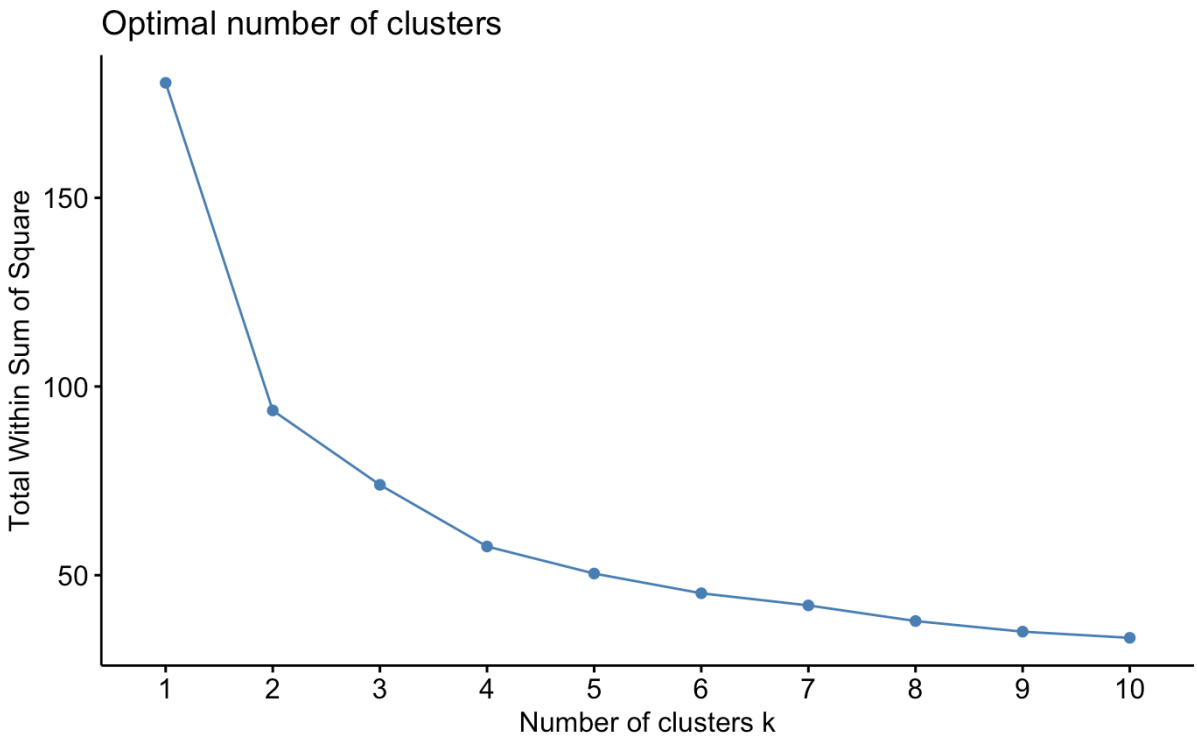


Figure 33: Elbow plot for K-means clustering

1.2.2.3. Silhouette

The clustering performance of the PCA-transformed dataset is evaluated using the Silhouette method. The average silhouette coefficient for each cluster and the silhouette coefficient for each vehicle are computed. The approach suggests successful clustering with a higher average silhouette coefficient, which aids in determining the appropriate number of clusters. It is helpful in vehicle classification and recommendation systems.

```
193 #silhouette_method
194 fviz_nbclust(transformed_data,kmeans,method = "silhouette")
```

Figure 34: Plotting the silhouette width measure for different number of clusters

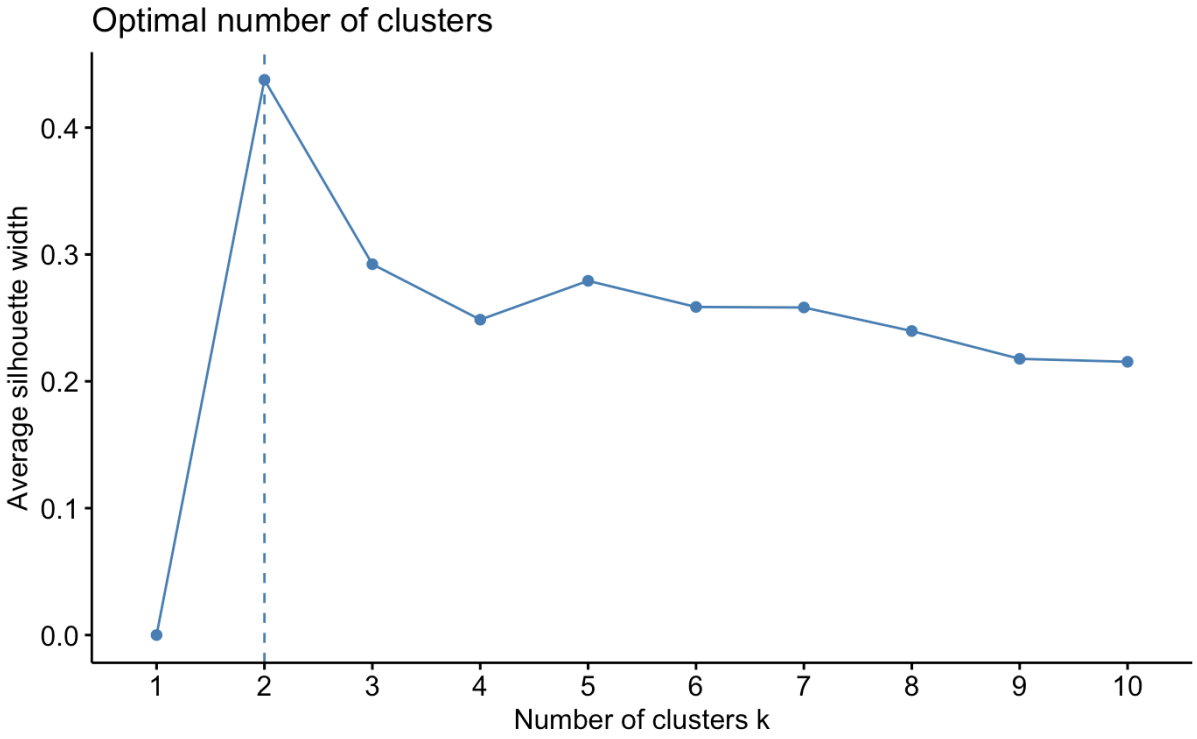


Figure 35: cluster quality analysis using silhouette method

1.2.2.4. Gap Statistics

Using PCA for dataset transformation allows for the reduction of data dimensionality while maintaining a substantial quantity of information. However, it is crucial to remember that PCA may make it more difficult to comprehend the original characteristics, particularly when there are numerous components at play. This indicates that while PCA offers advantages for dimension reduction, there is a trade-off for the interpretability of the underlying features, and this should be considered when using PCA to analyze data.

The Gap statistics approach compares the spread of distances between clusters in a clustering solution with the projected spread of a reference distribution that makes no assumptions about meaningful grouping. By analyzing this gap, the technique determines the number of clusters that maximizes the difference between the two.

```
196 #gap static_method
197 fviz_nbclust(transformed_data,kmeans,method = "gap_stat")
198
```

Figure 36:Visualizing the optimal number of clusters using the Gap statistic method

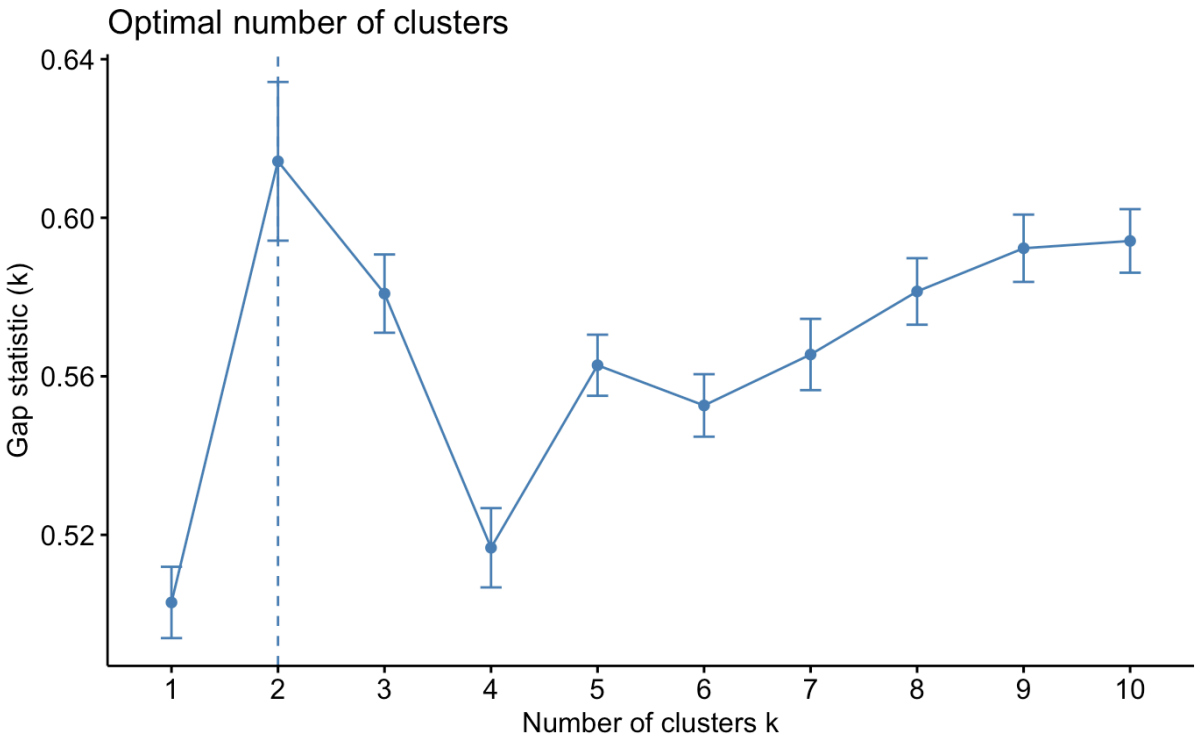


Figure 37:Gap statistics method for determining optimal k

1.2.3. K-means Clustering Investigation for transformed dataset (g)

The four automated methods were used to select the ideal k value for the k means clustering after PCA performance. The modified PCA dataset's k means clustering then started to operate using the most desired k value.

The output of the k-means clustering in R contained information about the cluster centers, the clustering outcomes, and the ratio of between-cluster sums of squares (BSS) to total sums of squares (TSS). The output was examined, the BSS and within-cluster sums of squares (WSS) indices were calculated, and they were then used to judge the quality of the clustering results. In general, when utilizing the selected k value for k-means clustering, the PCA-transformed dataset yielded well-defined and distinct clusters. This demonstrates that the dataset's inherent structure was maintained while the dimensionality of the dataset was successfully reduced by the PCA transformation.

- K2

```
#Perform k-means clustering with k=2
set.seed(1234)
kmeans_model_2 <- kmeans(transformed_data, centers = 2, nstart = 25)

#The k-means output in print
print(kmeans_model_2)
autoplot(kmeans_model_2,transformed_data,frame=TRUE)

#Calculate the within-cluster sum of squares (WSS)
wss_2 <- sum(kmeans_model_2$withinss)

#Calculate the between-cluster sum of squares (BSS)
bss_2 <- sum(kmeans_model_2$size * dist(rbind(kmeans_model_2$centers, colMeans(transformed_data)))^2)

#Calculate the total sum of squares (TSS)
tss_2 <- sum(dist(transformed_data)^2)

#Calculate the ratio of BSS to TSS
bss_tss_ratio_2 <- bss_2 / tss_2

#Print the WSS, BSS, TSS, and ratio of BSS to TSS
cat("For k=2:\n")
cat("Sum of squares within a cluster (WSS): ", wss_2, "\n")
cat("Between-cluster square sum (BSS): ", bss_2, "\n")
cat("sum of all squares (TSS): ", tss_2, "\n")
cat("Ratio of BSS to TSS: ", bss_tss_ratio_2, "\n\n")

```

Figure 38:code section for the k2 investigation

Output

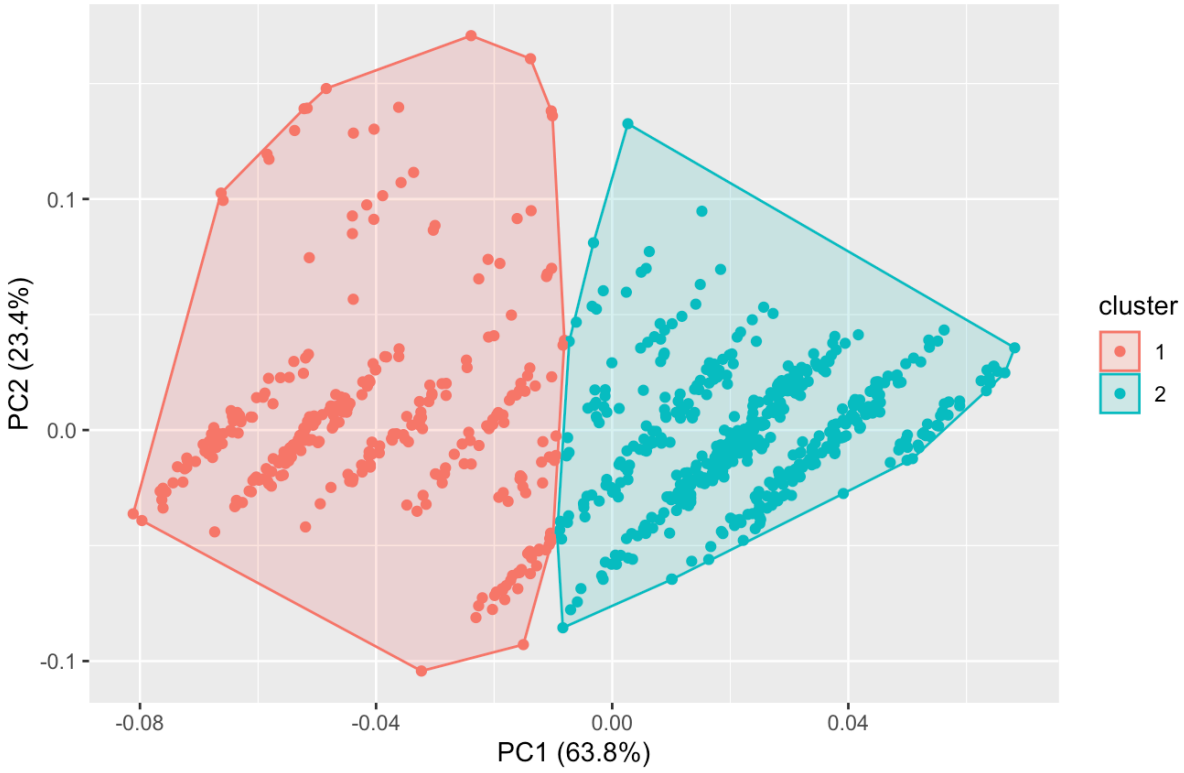


Figure 39:Kmeans clustering after PCA K2

```
> #Perform k-means clustering with k=2
> set.seed(1234)
> kmeans_model_2 <- kmeans(transformed_data, centers = 2, nstart = 25)
> #gap static_method
> fviz_nbclust(transformed_data,kmeans,method = "gap_stat")
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 100)  [one "." per sample]:
.................................................. 50
.................................................. 100
> set.seed(1234)
> kmeans_model_2 <- kmeans(transformed_data, centers = 2, nstart = 25)
>
> #The k-means output in print
> print(kmeans_model_2)
K-means clustering with 2 clusters of sizes 298, 515

Cluster means:
         PC6        PC7         PC8         PC9         PC10        PC11        PC12        PC13        PC14
1 -0.08711086 -0.1557934  0.05933291 -0.09249059 -0.009458928 -0.11804152  0.02984440  0.06281173  0.08020637
2  0.05040589  0.0901484 -0.03433244  0.05351883  0.005473322  0.06830364 -0.01726919 -0.03634543 -0.04641068
         PC15         PC16       PC17         PC18
1  0.15843351 -0.013179478  0.2996110 -0.016402714
2 -0.09167609  0.007626183 -0.1733671  0.009491279

Clustering vector:
  [1] 1 2 1 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 1 2 2 2 1 2 2 1 2 2 2 2 2 1 2 2 2 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 1
 [55] 1 2 1 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 2 2 1 2 1 2 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 1
[109] 1 1 2 2 2 1 2 2 1 2 1 1 1 2 2 1 2 2 2 2 2 2 1 2 2 1 1 2 2 2 1 2 2 1 2 1 2 1 2 2 2 2 2 1 2 2 1 1 2 1 2 2 1 1 2 1
[163] 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 2 1 2 2 1 2 2 1 2 2 1 2 2 2 1 2 1 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 1 2 2 2 2 1 2 1
[217] 2 2 2 1 2 1 2 2 1 2 2 2 2 1 2 2 2 2 2 1 2 2 2 1 1 2 1 1 2 2 1 1 2 2 1 1 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 1
[271] 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 1 2 2 2 2 2 2 1 1 1 1 2 2 1 2 2 2 1 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 1 2 1 1 2 2 2
[325] 2 2 2 2 2 1 1 1 2 2 2 1 2 2 2 1 1 2 1 2 2 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2 1 2 2 1 2 1 2 2 2 2 2 1
[379] 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 1 2 2 1 2 1 2 1 2 1 1 2 2 1 2 2 2 2 2 1 1 1 2 1 1 2 1 1 1 2 2 2
[433] 2 2 1 2 2 2 1 2 2 1 2 2 2 2 2 1 1 2 2 1 1 1 2 1 1 1 2 2 1 1 2 2 1 1 2 2 1 2 1 2 2 1 1 2 2 1 1 2 1 2 1 1 1 2 2 1 1 2
[487] 1 1 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 1 2 2 2 1 1 2 2 1 1 2 2 1 1 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 1 1 1 2 2 1 1 2 1 2 2
[541] 1 1 2 2 2 2 1 2 2 2 2 1 1 1 1 2 2 2 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 2 2 2 2 2 2
[595] 1 1 2 2 2 1 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 1 2 1 2 2 1 1 2 1 2 1 2 1 2 2 2 1 1 1 1 2 1 1 1 2 2
[649] 1 2 2 2 2 2 1 2 1 1 2 2 2 1 2 1 2 2 1 1 1 2 2 2 2 2 1 1 2 2 1 1 2 2 1 1 1 1 1 2 1 2 2 1 1 2 1 2 2 1 1 1 2 2 1 2 2 1
[703] 1 1 2 1 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 1 1 2 2 2 1 2 2 1 1 2 2 1 1 1 2 1 2 1 1 2
[757] 2 1 2 1 2 2 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 1 1 1 2 1 2 1 1 2 1 1 2 1 2 2 1 2 2 2 2 2 2 2 2 1
[811] 1 2 2
```

```
Within cluster sum of squares by cluster:
[1] 32.96546 27.28689 13.69047
 (between_SS / total_SS =  59.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
> autoplot(kmeans_model_3,transformed_data,frame=TRUE)
>
> #Calculate the squares' within-cluster sum(WSS)
> wss_3 <- sum(kmeans_model_3$withinss)
>
> #Calculate the squares' between-cluster sum (BSS)
> bss_3 <- sum(kmeans_model_3$size * dist(rbind(kmeans_model_3$centers, colMeans(transformed_data)))^2)
>
> #Calculate the sum of all the squares (TSS)
> tss_3 <- sum(dist(transformed_data)^2)
>
> #Calculate the BSS to TSS ratio.
> bss_tss_ratio_3 <- bss_3 / tss_3
>
> #Print out the WSS, BSS, TSS, and the BSS/TSS ratio
> cat("For k=3:\n")
For k=3:
> cat("Sum of squares within a cluster (WSS): ", wss_3, "\n")
Sum of squares within a cluster (WSS):  73.94282
> cat("Between-cluster square sum (BSS): ", bss_3, "\n")
Between-cluster square sum (BSS):  419.7007
> cat("sum of all squares (TSS): ", tss_3, "\n")
sum of all squares (TSS):  146717.9
> cat("Ratio of BSS to TSS: ", bss_tss_ratio_3, "\n")
Ratio of BSS to TSS:  0.002860596
```

Figure 40:code section for the k2 investigation

- K3

```r
#Perform k-means clustering with k=3
set.seed(1234)
kmeans_model_3 <- kmeans(transformed_data, centers = 3, nstart = 25)

#The k-means output in print
print(kmeans_model_3)
autoplot(kmeans_model_3,transformed_data,frame=TRUE)

#Calculate the squares' within-cluster sum(WSS)
wss_3 <- sum(kmeans_model_3$withinss)

#Calculate the squares' between-cluster sum (BSS)
bss_3 <- sum(kmeans_model_3$size * dist(rbind(kmeans_model_3$centers, colMeans(transformed_data)))^2)

#Calculate the sum of all the squares (TSS)
tss_3 <- sum(dist(transformed_data)^2)

#Calculate the BSS to TSS ratio.
bss_tss_ratio_3 <- bss_3 / tss_3

#Print out the WSS, BSS, TSS, and the BSS/TSS ratio
cat("For k=3:\n")
cat("Sum of squares within a cluster (WSS): ", wss_3, "\n")
cat("Between-cluster square sum (BSS): ", bss_3, "\n")
cat("sum of all squares (TSS): ", tss_3, "\n")
cat("Ratio of BSS to TSS: ", bss_tss_ratio_3, "\n")
```

Figure 41: Output Kmeans clutering k2

Output

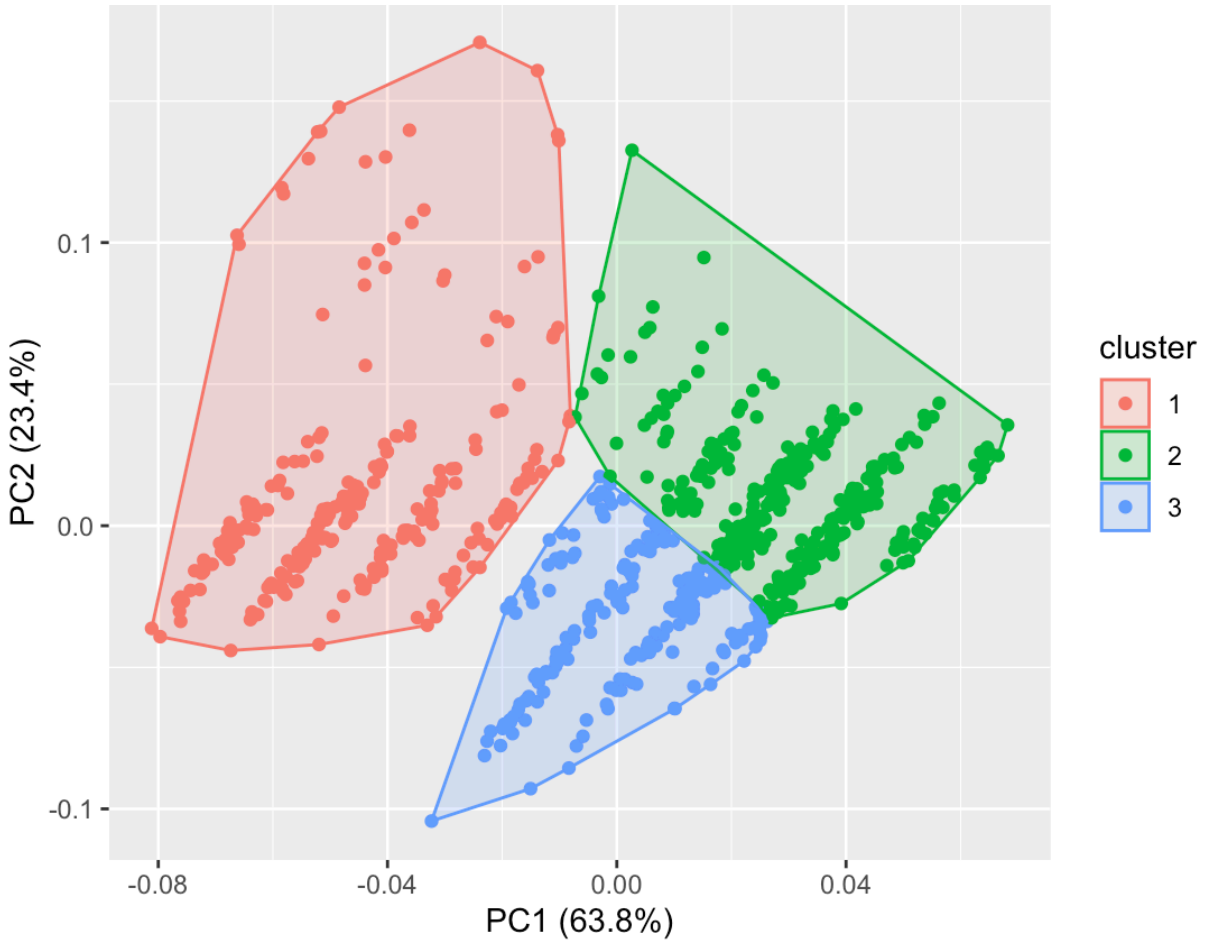


Figure 42:Kmeans clustering after PCA K3

```
> print(kmeans_model_3)
K-means clustering with 3 clusters of sizes 249, 361, 203

Cluster means:
         PC6         PC7          PC8         PC9        PC10        PC11         PC12        PC13        PC14        PC15
1 -0.06429511 -0.14710694  0.066555191 -0.09409261  0.01462570 -0.12532617  0.0529470480  0.05975654  0.09426947  0.18788103
2  0.09886444  0.14905159 -0.041712044  0.07876478  0.03944040  0.09470938 -0.0007331176 -0.05867234 -0.05796432 -0.10496631
3 -0.09694866 -0.08462067 -0.007459087 -0.02465531 -0.08807775 -0.01469887 -0.0636411798  0.03104107 -0.01255161 -0.04379083
          PC16       PC17          PC18
1 -0.028382176  0.3639999 -0.0179636541
2 -0.003847911 -0.1887212  0.0125562078
3  0.041656443 -0.1108750 -0.0002947839

Clustering vector:
  [1] 3 3 1 3 1 2 3 2 1 3 2 2 3 2 1 2 2 1 1 2 2 2 2 1 3 2 1 2 3 3 2 2 1 2 3 2 1 2 1 2 3 3 3 2 3 2 2 2 1 2 1 2 1 3 1 2 1 2 3 2 1
 [62] 3 3 1 3 1 1 1 3 2 3 1 3 2 1 2 1 2 2 3 3 2 2 2 1 3 1 3 2 1 2 2 1 2 3 2 2 2 1 1 2 3 2 3 3 2 3 3 1 1 3 2 2 3 3 2 3 2 3 1 1 2
[123] 2 1 3 3 2 3 3 3 1 3 2 1 3 2 2 3 1 2 2 1 3 1 2 2 2 2 2 1 3 2 1 1 2 1 2 2 1 1 3 1 3 2 2 2 2 2 1 2 2 2 1 2 2 2 1 3 3 2 1 2 2
[184] 1 2 2 2 3 2 1 1 2 3 2 2 3 1 3 2 2 2 2 2 2 1 2 2 1 3 2 2 2 2 2 1 3 1 2 2 2 1 3 3 2 3 1 2 3 2 2 1 3 2 2 2 1 2 3 1 3 2 3 1 2 2
[245] 1 1 2 3 2 2 1 2 2 3 2 2 2 2 3 3 1 2 2 2 1 2 1 2 2 1 3 3 2 3 1 2 2 2 2 1 2 2 2 3 2 1 2 2 2 3 2 1 1 1 1 3 3 2 1 2 2 3 1 2 1
[306] 1 1 3 1 2 2 3 2 3 2 3 1 1 2 1 1 3 2 3 2 2 2 2 1 1 1 2 3 2 1 2 3 2 3 3 2 1 2 1 1 1 2 3 2 1 3 3 2 3 2 2 3 2 2 1 1 2 2 1 2 2
[367] 1 2 3 1 3 1 3 3 2 2 3 1 2 2 3 3 1 2 1 2 3 2 2 3 3 2 2 2 3 1 2 2 3 2 2 1 2 2 3 2 1 3 1 3 1 1 2 2 1 2 2 2 3 1 1 3 2 1 1 3 1
[428] 1 1 3 3 2 3 2 1 2 2 3 1 3 2 1 3 2 2 3 3 1 1 2 2 1 1 1 3 1 1 3 2 2 1 1 3 2 3 2 1 2 2 1 1 3 2 1 1 2 1 3 1 1 1 3 2 1 1 3 3 1
[489] 2 2 1 3 2 2 1 2 2 3 2 2 1 2 1 1 2 2 3 3 1 1 2 2 3 1 2 1 1 2 2 3 3 3 2 2 3 1 2 3 3 3 1 3 1 2 2 1 1 3 1 2 2 3 1 3 2 3 1 2 3 3
[550] 1 1 1 1 2 2 2 3 1 1 1 2 1 3 2 1 3 2 3 2 2 2 3 2 2 2 3 2 1 2 2 1 2 2 2 2 1 3 2 2 2 2 2 2 2 1 1 3 3 2 1 2 2 3 2 2 1 2 1 2 2
[611] 2 2 2 1 1 3 1 2 3 2 2 2 1 2 1 2 2 3 3 2 3 2 1 2 1 2 3 2 3 1 1 3 2 1 3 1 2 3 1 3 2 2 3 3 2 3 1 2 2 2 1 2 1 3 2 1 3 3 2 2 2
[672] 2 2 1 1 3 3 1 1 2 3 3 1 1 1 1 3 1 3 2 1 1 3 1 3 1 3 2 1 2 2 1 1 1 2 1 2 2 1 1 1 3 1 2 2 1 3 2 3 2 3 1 2 2 2 2 2 2 1 2 2 3
[733] 1 2 1 1 2 3 2 1 2 3 1 1 2 3 2 1 1 1 2 1 2 1 1 2 2 1 3 1 2 2 3 1 2 2 2 1 2 3 2 2 2 1 2 2 1 2 2 1 2 2 3 2 3 1 1 1 2 1 3 1 1
[794] 3 3 1 2 3 2 2 1 2 2 2 3 3 2 2 3 3 1 2 2

Within cluster sum of squares by cluster:
[1] 32.96546 27.28689 13.69047
 (between_SS / total_SS =  59.0 %)
```

1.2.4. Silhouette plot (h)

The silhouette plot is a useful tool to evaluate the quality of the clusters. The silhouette plot shows the silhouette coefficient for each observation, which quantifies how similar each observation is to the assigned cluster in comparison to other clusters. A bigger value denotes a good match within the cluster, whereas a smaller coefficient implies that the observation might fit better in a different cluster. Negative coefficients in the silhouette plot imply that an observation might not have been located where it should have been within the cluster. The plot is an effective tool for analyzing the results and identifying potential issues even when total clustering might not be feasible. It aids in providing recommendations for further clustering algorithms or data preprocessing technique advancement.

- K2

```
#Fit k-means model with k=2
k <- 2
kmeans_Model <- kmeans(transformed_data, centers = k, nstart = 25)

#Create a silhouette plot
silhouette_plot <- silhouette(kmeans_Model$cluster, dist(transformed_data))

#Calculate average silhouette width
avg_sil_width <- mean(silhouette_plot[, 3])

#Plot the silhouette plot
plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
     xlab = "Silhouette Width", ylab = "Cluster", border = NA)

#As a vertical line, add the average silhouette width
abline(v = avg_sil_width, lty = 2, lwd =2,col="red")

```

Figure 44:code section for the silhouette plot (K2)

Output

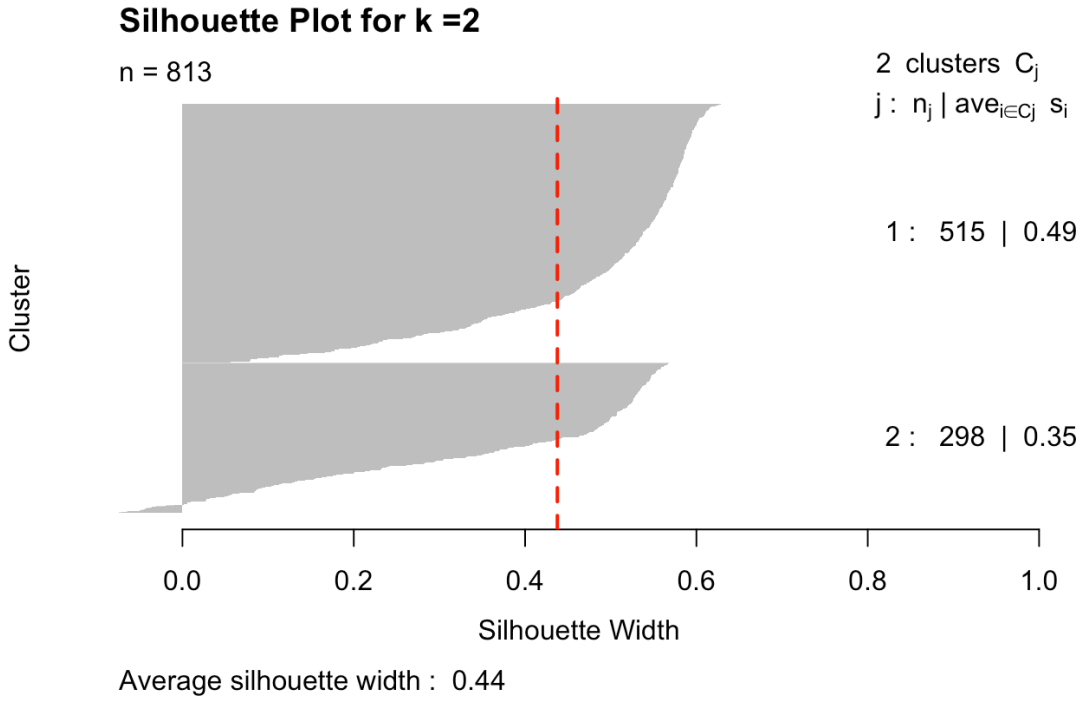


Figure 43:Output Kmeans clutering k2

- K3

```
# Fit k-means model with k=3
k <- 3
kmeans_model <- kmeans(transformed_data, centers = k, nstart = 25)

#Create a silhouette plot
silhouette_plot <- silhouette(kmeans_model$cluster, dist(transformed_data))

#Calculate average silhouette width
avg_sil_width <- mean(silhouette_plot[, 3])

#Plot the silhouette plot
plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
     xlab = "Silhouette Width", ylab = "Cluster", border = NA)

#Add average silhouette width as vertical line
abline(v = avg_sil_width, lty = 2, lwd =2,col="red")

```

Figure 45:code section for the silhouette plot (K3)

Output

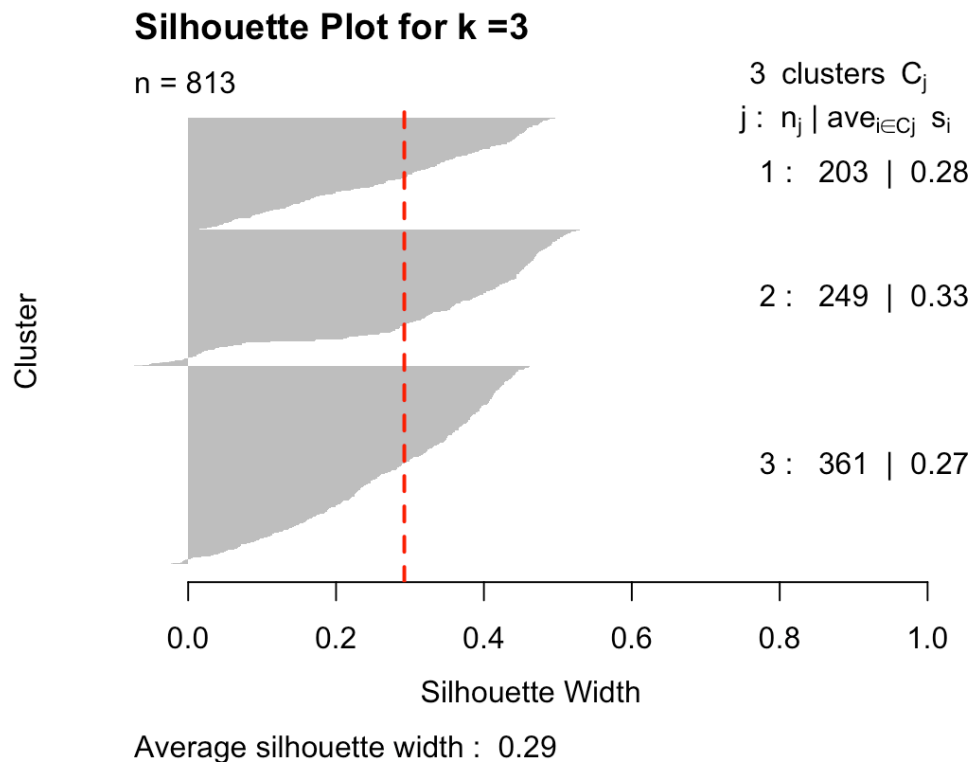


Figure 46:Silhouette plot after PCA k3

1.2.5. Canlinki Harabaza Index (i)

```
> #Calculate Calinski-Harabasz Index k =2
> calinski_harabasz_pca <- function(cluster_result, data) {
+   k2 <- length(unique(cluster_result$cluster))
+   n2 <- nrow(data)
+   BSS2 <- cluster_result$betweenss
+   WSS2 <- cluster_result$tot.withinss
+
+   ch_index2 <- ((n2 - k2) / (k2 - 1)) * (BSS2 / WSS2)
+   return(ch_index2)
+ }
>
> ch_index_pca_2 <- calinski_harabasz_pca(kmeans_Model, transformed_data)
> ch_index_pca_2
[1] 751.7775
> #Calculate Calinski-Harabasz Index k =3
> calinski_harabasz_pca <- function(cluster_result, data) {
+   k3 <- length(unique(cluster_result$cluster))
+   n3 <- nrow(data)
+   BSS3 <- cluster_result$betweenss
+   WSS3 <- cluster_result$tot.withinss
+
+   ch_index3 <- ((n3 - k3) / (k3 - 1)) * (BSS3 / WSS3)
+   return(ch_index3)
+ }
>
> ch_index_pca_3 <- calinski_harabasz_pca(kmeans_model, transformed_data)
> ch_index_pca_3
[1] 583.4431
```

Figure 47:Canlinki Harabaza

2. Energy Forecasting

2.1. 1st Subtask Objectives

2.1.1. Part (a)

The input vector selection is critical in multilayer neural network (MLP) models developed for electrical load forecasting. A number of variables in the input vector are utilized to forecast how much power will be needed in the upcoming time step. Determining the input vector is crucial for energy forecasting analysis since it directly affects prediction accuracy.

In the field of forecasting electrical demand, there are a range of designs and methods for defining the input vector for MLP models. These tactics are developed and applied in various ways. Among the regularly employed systems are:

Autoregressive (AR) approach: A time series modeling technique called autoregressive (AR) estimates a variable's future values based on its past values. This approach allows one to define a time series' present value as a linear function of its past values. An AR (1) model, for instance, would use the previous value of the time series as a predictor for the present value, but an AR (2) model would make use of the time series' two prior values. The AR technique assumes that the mistakes in the model have a fixed variance and are uncorrelated. The AR technique, which is helpful in many fields including finance, economics, and engineering, enables us to forecast the future values of a time series based on its historical values.

Moving Average (MA) approach: The Moving Average (MA) approach, a time series modeling tool, forecasts future values of a variable by averaging its past values. The "order" of the model, which determines how many prior values are used in the computation, is a parameter that affects how a time series' present value is a function of the average of its previous values.

Fourier transform approach: The Fourier transform methodology, a mathematical technique, is used to analyze time series data by converting it from the time domain to the frequency domain. The dominant frequencies in the data can be identified using this method. By employing this technique to identify seasonal patterns or cycles in the data, the precision of load forecasting models in the field of energy load forecasting can be improved.

Weather data approach: The weather data methodology uses meteorological variables like temperature, humidity, and wind speed as inputs to the forecasting model in order to estimate the required amount of power. The use of weather data can help improve the accuracy of load forecasting models since this approach recognizes that weather patterns have a significant impact on energy demand.

2.1.2. MLP training / testing using AR approach (b)

A sliding window approach is used to build an input vector for the MLP model using data on power use from the University of Westminster. One strategy for solving the problem is to select a window size, which designates the number of time steps anterior to the present one that are used as input variables. The values of the electricity usage at 18:00, 19:00, and 20:00 the day before would be used as input variables, for instance, if a window size of three is chosen. If the window size is set to 3, then this would be the situation. The amount of electricity utilized at 0:00 on the current day would be the value of the output variable.

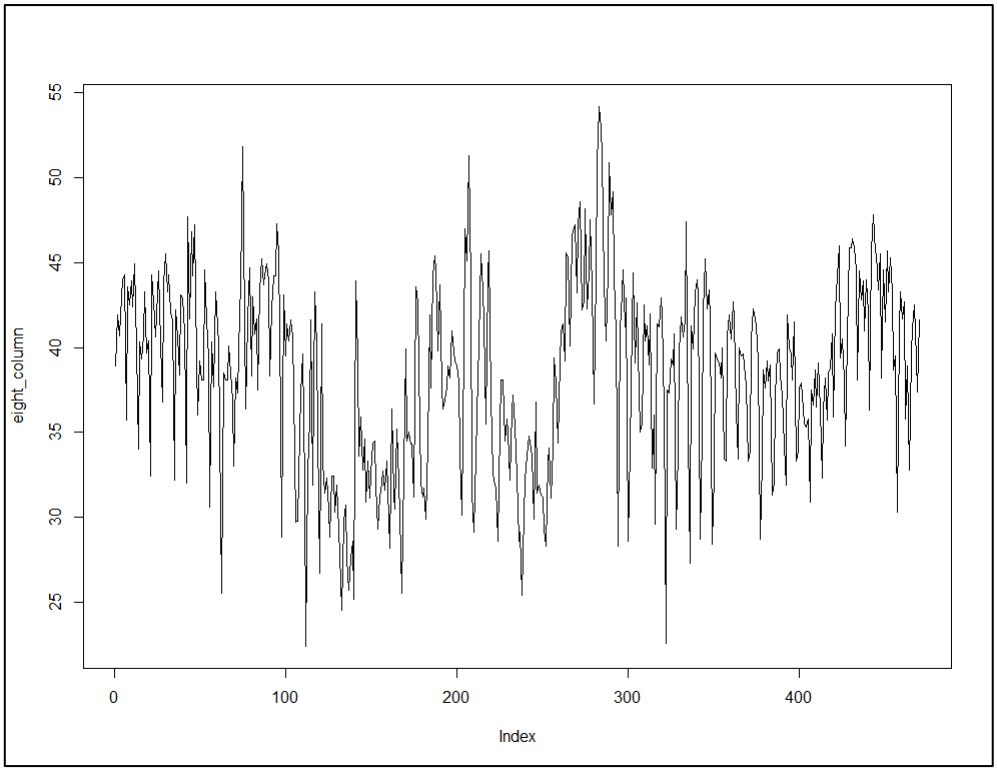
The process of creating the subsequent row of the input/output matrix (I/O matrix) continues with the iteration of the preceding phase after sliding the window forward by one time step. After it is finished, the MLP model is trained and tested using the I/O matrix. In order to determine the input vector that performs the best for the MLP model, a number of time-delayed input vectors are tested because the precise sequence for the autoregressive (AR) approach is not predefined. By performing tests with a range of time-delayed input vectors, it is feasible to identify the input configuration for the MLP model that is most advantageous. This will help to improve the model's overall performance.

```
28  # create I/O matrix
29  main_time_delayed_matrix <- bind_cols(t7 = lag(eight_column,8),
30                                        t4 = lag(eight_column,5),
31                                        t3 = lag(eight_column,4),
32                                        t2 = lag(eight_column,3),
33                                        t1 = lag(eight_column,2),
34                                        eightHour = eight_column)
35
36  time_delayed_matrix <- na.omit(main_time_delayed_matrix)
37
```

Figure 48:Code section for the MLP testing

First, a time series called "eight_column" with numerous lagged copies and different time delays ("t7," "t4", "t3," "t2," and "t1") are combined into a single data frame called "main_time_delayed_matrix" using the "bind_cols()" technique.

Then, any rows with missing values are removed using the "na.omit()" function. This is because R's "lag()" function adds "NA" values to the lagged time series' beginning. The resulting "time_delayed_matrix" can be used to train and test a forecasting model, with the "eightHour" column serving as the output or objective variable.



2.1.3. Normalization

2.1.3.1. Why normalization Important

What is "Normalization"- Normalization is the process of transforming numerical data so that it falls within a specific range, typically between 0 and 1 or -1 and 1. It is extensively used in machine learning and data analysis to improve algorithm performance and make data interpretation easier. Normalization enables more accurate comparisons and analysis by removing the impact of variable scales, units, and data ranges.

```
48  #normalization function for min-max
49  normalize <- function(x){
50    return ((x - min(x)) / (max(x) - min(x)))
51  }
```

Figure 49:code section for the min - max normalization

Using this R function, the "x" vector or matrix is min-max normalized. You can use it by taking the vector or matrix, removing the minimum "x" value from each element, and dividing the result by the difference between the maximum and minimum "x" values of the vector or matrix. A new vector or matrix with values between zero and one is produced by converting the lowest value of "x" to zero and the highest value to one. The normalized vector or matrix is returned by the "normalise"

function, which takes an integer parameter ("x") as input. The "min ()" and "max ()" functions are used to calculate the minimum and maximum "x" values, respectively. The minimum value of "x" is then subtracted from each element of the vector or matrix using the "-" operator., The difference between the greatest and least "x" values is divided by the calculated value. The final result of the function is then the normalized vector or matrix.

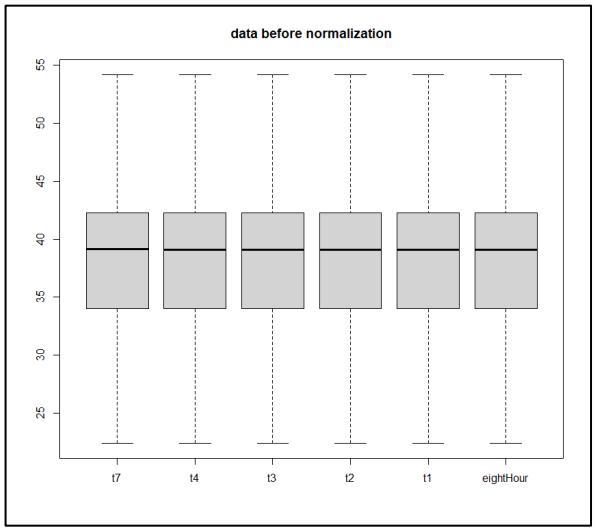


Figure 51: Before Normalization

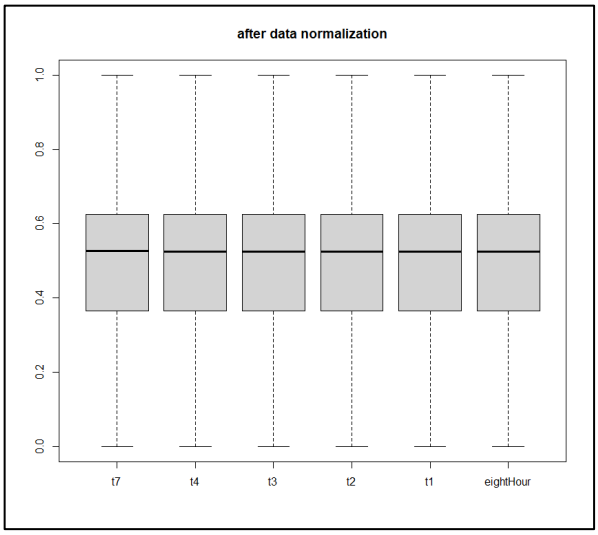


Figure 50:After Normalization

Before normalization, values between 25 and 55 may be seen in both graphs, while the final graph only contains 0 and 1.

2.1.3.2. Training Neural Networks

2.1.3.2.1. Splitting data for training and testing

```
60  #after normalizing test data, separating the data
61  train_dataset_norm <- time_delayedNorm[1:380,]
62  test_dataNorm <- time_delayedNorm[381:nrow(time_delayed_matrix),]
63
```

Figure 52:Splitting data for training

The code generates a training set and a test set using the normalized data. The first line assigns the first 380 rows to a "train_dataset_norm" variable and the rest of the rows to a "test_dataNorm" variable. It is standard procedure in machine learning to train a model on a subset of the data before testing it on the entire dataset.

2.1.3.2.2. Testing data for each time delay

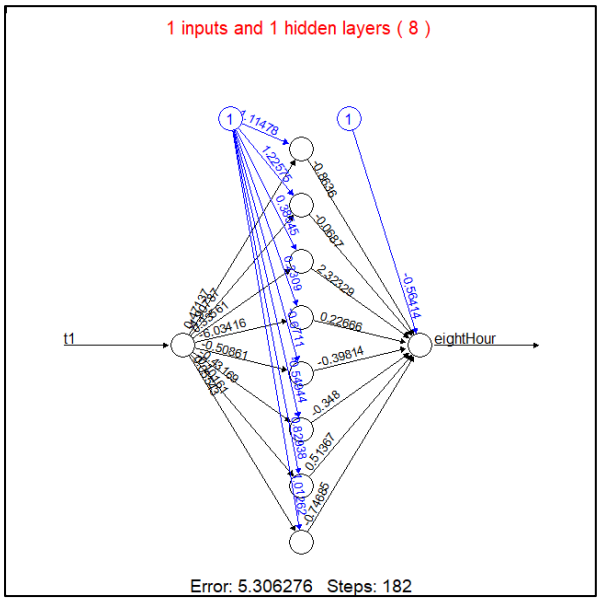
```r
#the generation of testing data for each timeDelay
t1_testing_data <- as.data.frame(test_dataNorm[, c("t1")])
t2_testing_data <- test_dataNorm[, c("t1", "t2")]
t3_testing_data <- test_dataNorm[, c("t1", "t2", "t3")]
t4_testing_data <- test_dataNorm[, c("t1", "t2", "t3", "t4")]
t7_testing_data<- test_dataNorm[, c("t1", "t2", "t3", "t4", "t7")]

```

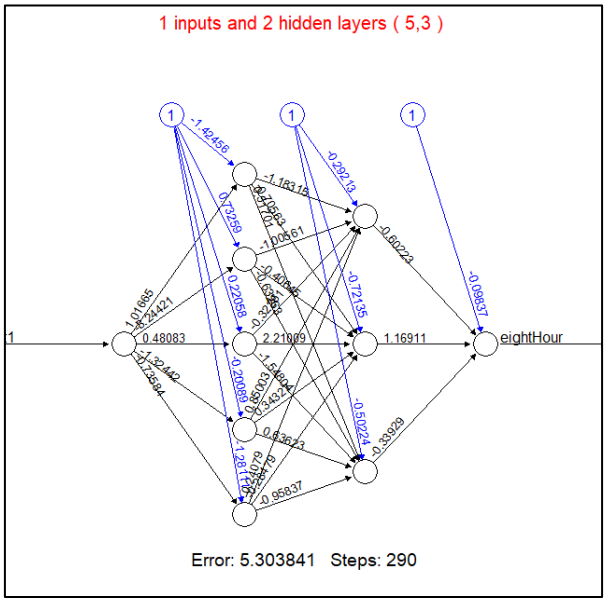
Figure 53:create testing for each time delay

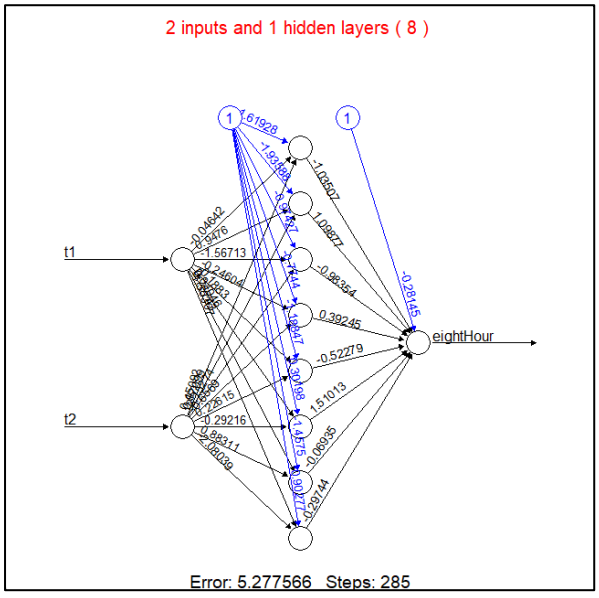
2.1.3.2.3. Training the model

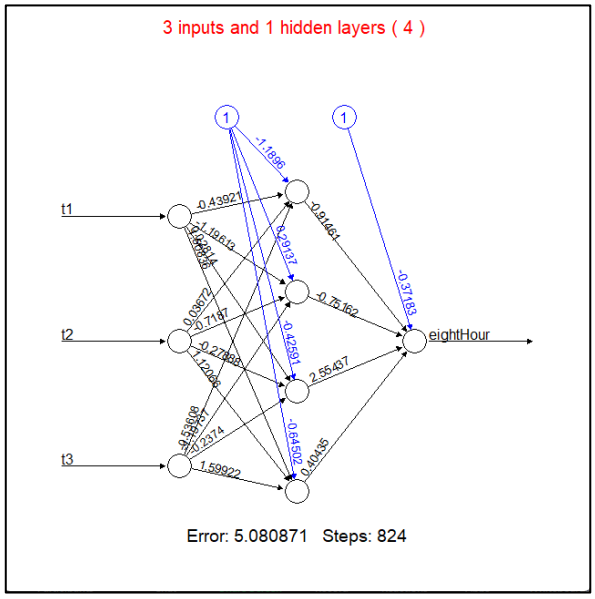
```r
#ability to train an AR model
trainModel <- function(formula, hiddenVal, isLinear, actFunc,inputs,hidden){

  my_text <- paste(inputs,"inputs and",length(hidden),"hidden layers","(",paste(hidden, collapse=","),") \n")

  set.seed(1234)
  nn <- neuralnet(formula,data = train_dataset_norm, hidden=hiddenVal, act.fct = actFunc, linear.output=isLinear)
  plot(nn)

  plot_panel <- grid.grab(wrap = TRUE)

  ## create a title grob:
  plot_title <- textGrob(my_text,
                         x = .5, y = .50,
                         gp = gpar(lineheight = 2,
                                   fontsize = 15, col = 'red',
                                   adj = c(1, 0)
                         )
  )

  #stack title and main panel, and plot:
  grid.arrange(
    grobs = list(plot_title,
                 plot_panel),
    heights = unit(c(.15, .85), units = "npc"),
    width = unit(1, "npc")
  )
  dev.new()
  dev.off()
  return(nn)
}
```

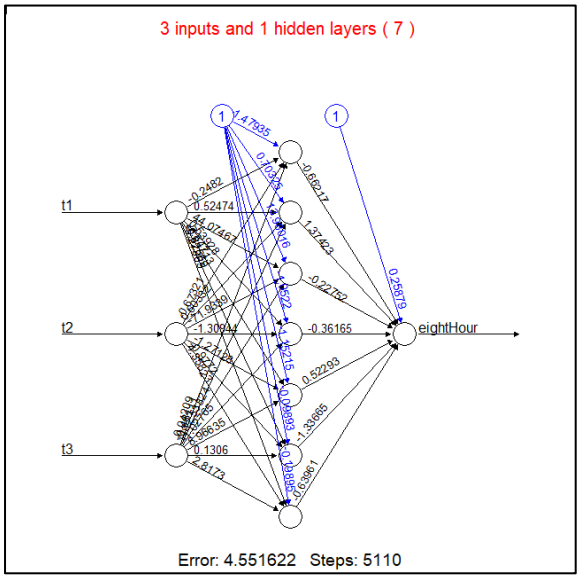
Figure 54:function to train the model

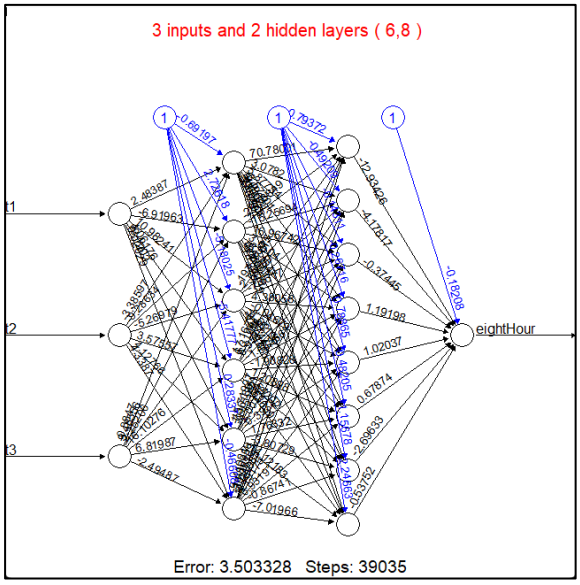

1 inputs and 1 hidden layers (8)
t1
eightHour
Error: 5.306276 Steps: 182

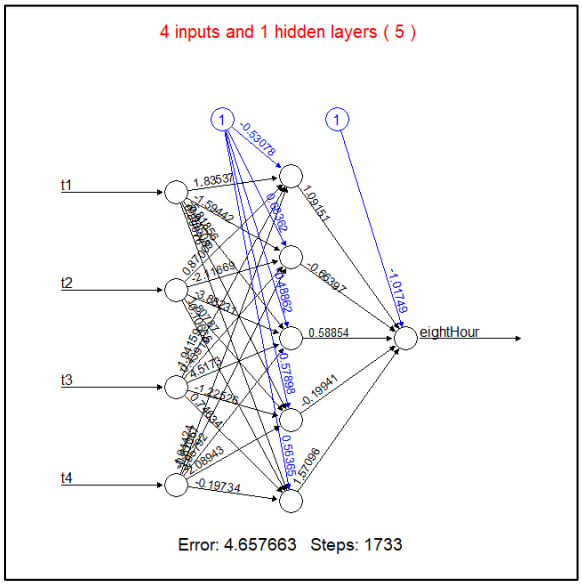

1 inputs and 2 hidden layers (5,3)
1
eightHour
Error: 5.303841 Steps: 290

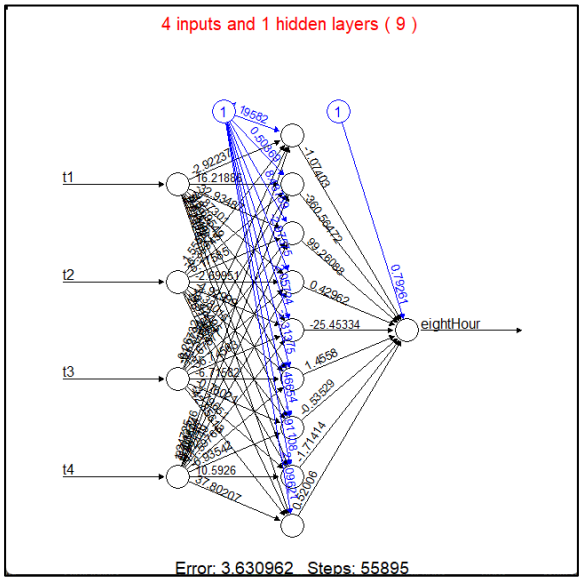

2 inputs and 1 hidden layers (8)
t1
t2
eightHour
Error: 5.277566 Steps: 285

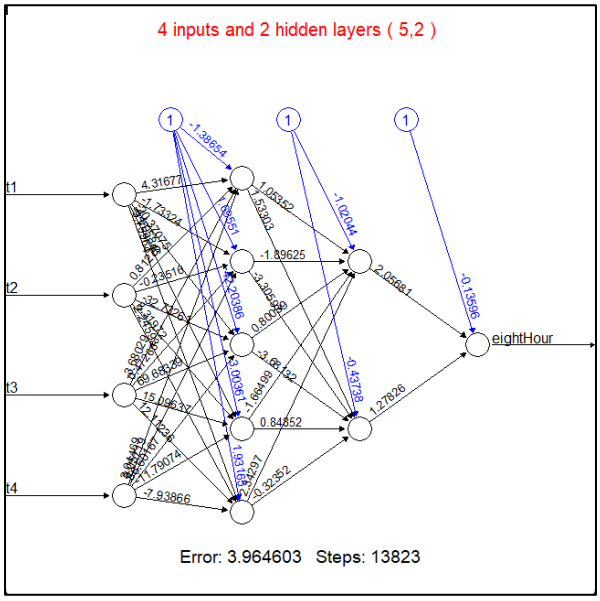

3 inputs and 1 hidden layers (4)
t1
t2
t3
eightHour
Error: 5.080871 Steps: 824

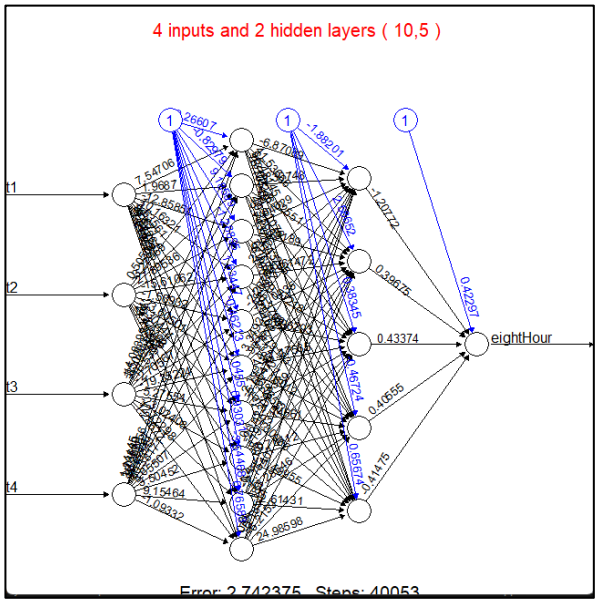

3 inputs and 1 hidden layers (7)
t1
t2
t3
eightHour
Error: 4.551622 Steps: 5110

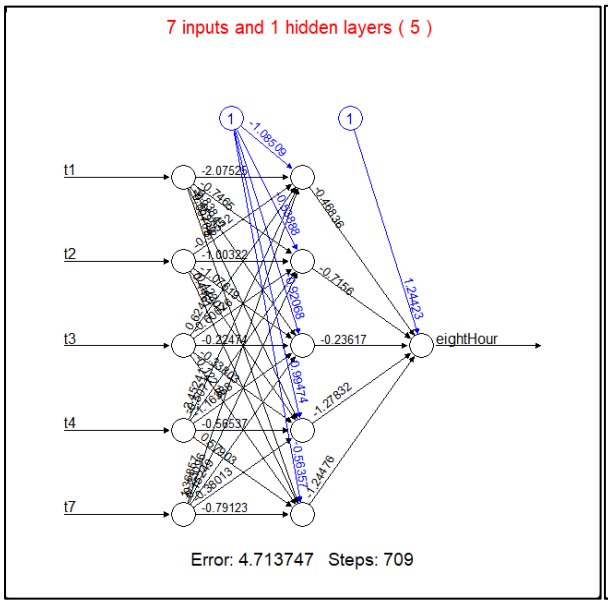

3 inputs and 2 hidden layers (6,8)
t1
t2
t3
eightHour
Error: 3.503328 Steps: 39035

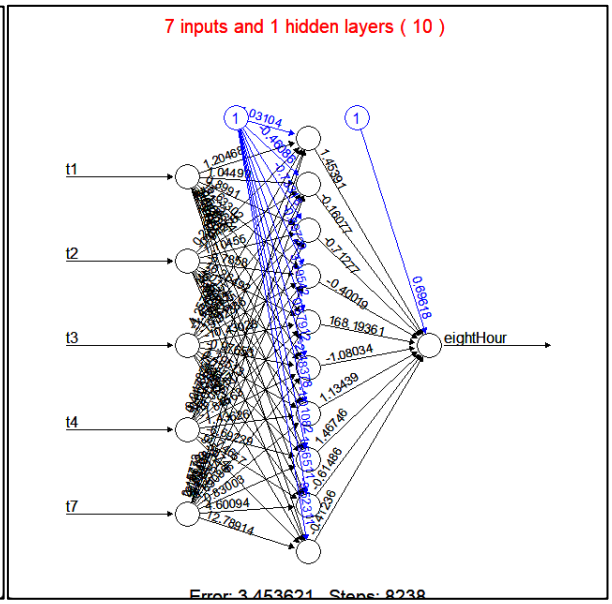

4 inputs and 1 hidden layers (5)
t1
t2
t3
t4
eightHour
Error: 4.657663 Steps: 1733

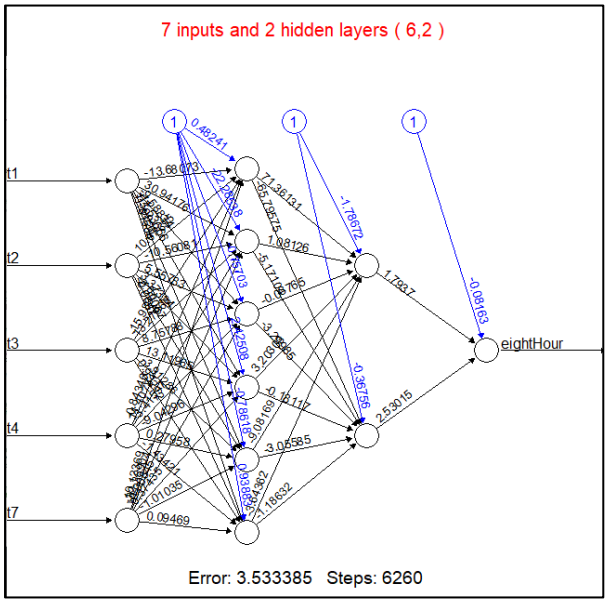

4 inputs and 1 hidden layers (9)
t1
t2
t3
t4
eightHour
Error: 3.630962 Steps: 55895

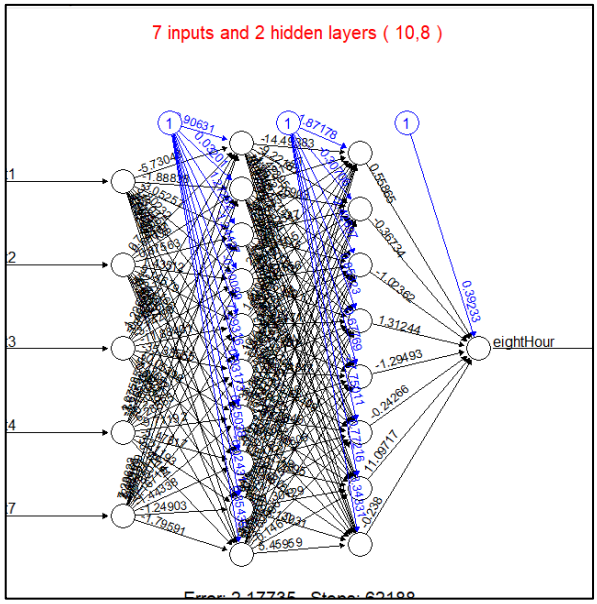

4 inputs and 2 hidden layers (5,2)
t1
t2
t3
t4
eightHour
Error: 3.964603 Steps: 13823

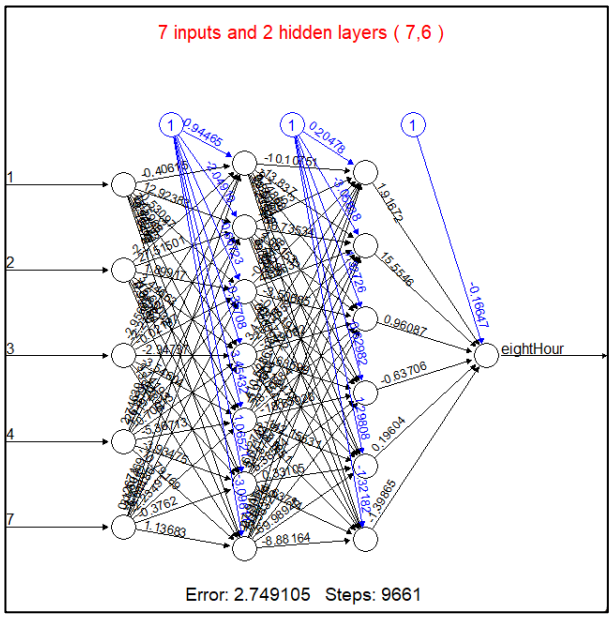

4 inputs and 2 hidden layers (10,5)
t1
t2
t3
t4
eightHour


7 inputs and 1 hidden layers (5)
t1
t2
t3
t4
t7
eightHour
Error: 4.713747 Steps: 709


7 inputs and 1 hidden layers (10)
t1
t2
t3
t4
t7
eightHour


7 inputs and 2 hidden layers (6,2)
t1
t2
t3
t4
t7
eightHour
Error: 3.533385 Steps: 6260


7 inputs and 2 hidden layers (10,8)
1
2
3
4
7
eightHour


7 inputs and 2 hidden layers (7,6)
1
2
3
4
7
eightHour
Error: 2.749105 Steps: 9661

2.1.4. RMSE, MAE, MAPE, and sMAPE

- **RMSE (Root Mean Square Error)**

In regression analysis, the term "RMSE" (Root Mean Squared Error) refers to a statistic that is widely used to measure the difference between expected and actual values. It is calculated by averaging the squared differences between the expected and actual values. RMSE is a well-liked method for evaluating and comparing the accuracy of machine learning models. Lower RMSE values indicate the model is more accurate.

- **MAE (Mean Absolute Error)**

Mean Absolute Error, sometimes known as MAE, is a unit of measurement for the discrepancy between actual and projected values in regression analysis. It is easier to read because it is in the same unit as the target variable and is derived by averaging the absolute differences between the projected and actual values.

- **MAPE (Mean Absolute Percentage Error)**

Mean Absolute Percentage Error, or MAPE, is a metric used in regression analysis to quantify the percentage difference between predicted and actual values. It is used to measure accuracy and assess how well machine learning models perform, although it has drawbacks such being undefinable at zero and sensitive to extreme values.

- **sMAPE (Symmetric Mean Absolute Percentage Error)**

In regression analysis, the percentage difference between expected and observed values is measured using the sMAPE metric. The total of the expected and actual values is used as the denominator, and it is determined by averaging the absolute percentage discrepancies between the predicted and actual values. Better accuracy is indicated by a lower sMAPE value.

2.1.5. Comparison Table

Input count	Hidden layers count	Neurons count	Activation function	linear	Accuracy	RMSE	MAE	MAPE	sMAPE
1	1	8	tanh	FALSE	98.05 %	3.953501	3.268071	0.08181146	0.08298622
	2	5,3	logistic	TRUE	98.03 %	3.951561	3.265439	0.08172144	0.08290732
2	1	8	logistic	TRUE	98.47 %	3.900452	3.253765	0.08172517	0.0824048
3	1	4	logistic	TRUE	98.96 %	3.848869	3.153246	0.07978453	0.07977575
	1	7	logistic	TRUE	97.59 %	3.865658	3.088643	0.07733229	0.07835654
	2	6,8	logistic	TRUE	98.27 %	4.438019	3.466305	0.08810957	0.08854005
4	1	5	logistic	TRUE	97.99 %	3.853526	3.160198	0.07934702	0.08003193
	1	9	logistic	TRUE	97.2 %	4.014217	3.128327	0.07796155	0.07968723
	2	5,2	logistic	TRUE	97.04 %	4.571932	3.596863	0.08996218	0.09198034
	2	10,5	logistic	TRUE	98.94 %	4.209867	3.480765	0.08761536	0.08792799
7	1	5	logistic	TRUE	99.35 %	3.661854	2.970384	0.07559341	0.07517955
	1	10	logistic	TRUE	97.63 %	4.144396	3.333606	0.08353244	0.08494433
	2	6,2	logistic	TRUE	98.07 %	4.346704	3.394495	0.08567309	0.08697417
	2	10,8	logistic	TRUE	98.01 %	5.481888	4.415487	0.1114797	0.1133419
	2	7,6	logistic	TRUE	97.87 %	4.966873	3.801418	0.0953542	0.0986283

2.1.6. Best one Hidden layer & best two hidden layer

2.1.6.1. Best One hidden layer

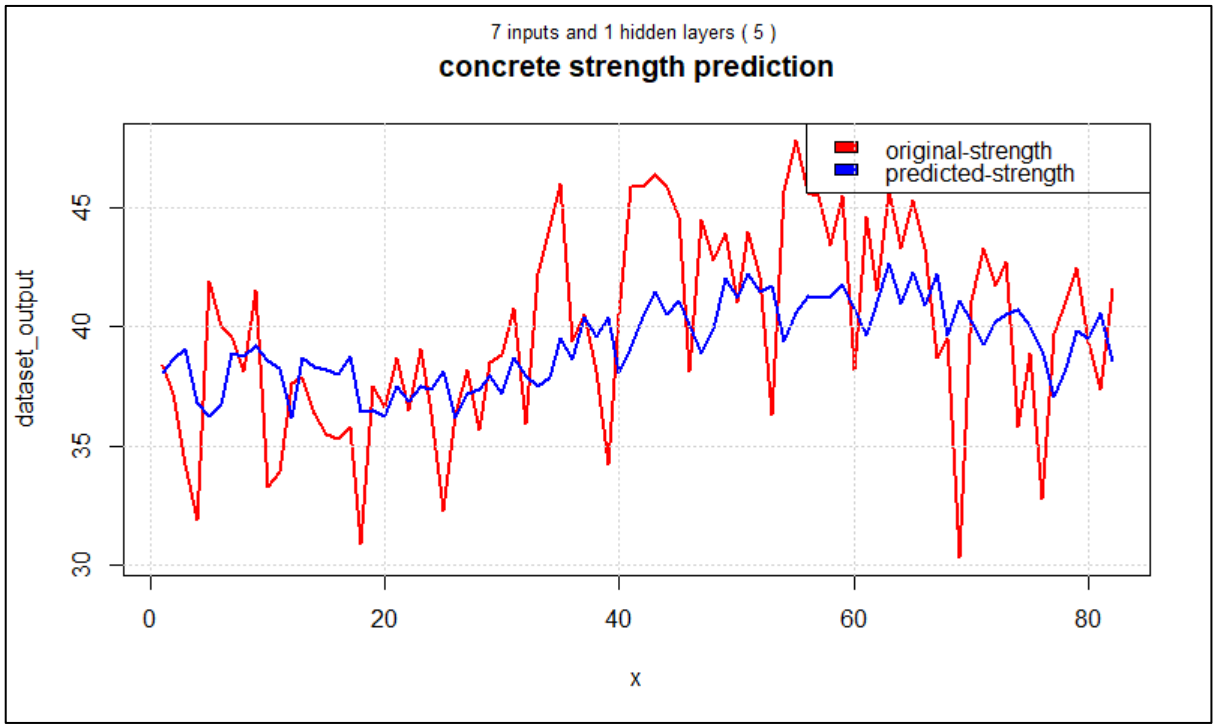


Figure 55: Best One hidden layer

2.1.6.2. Best two hidden layers

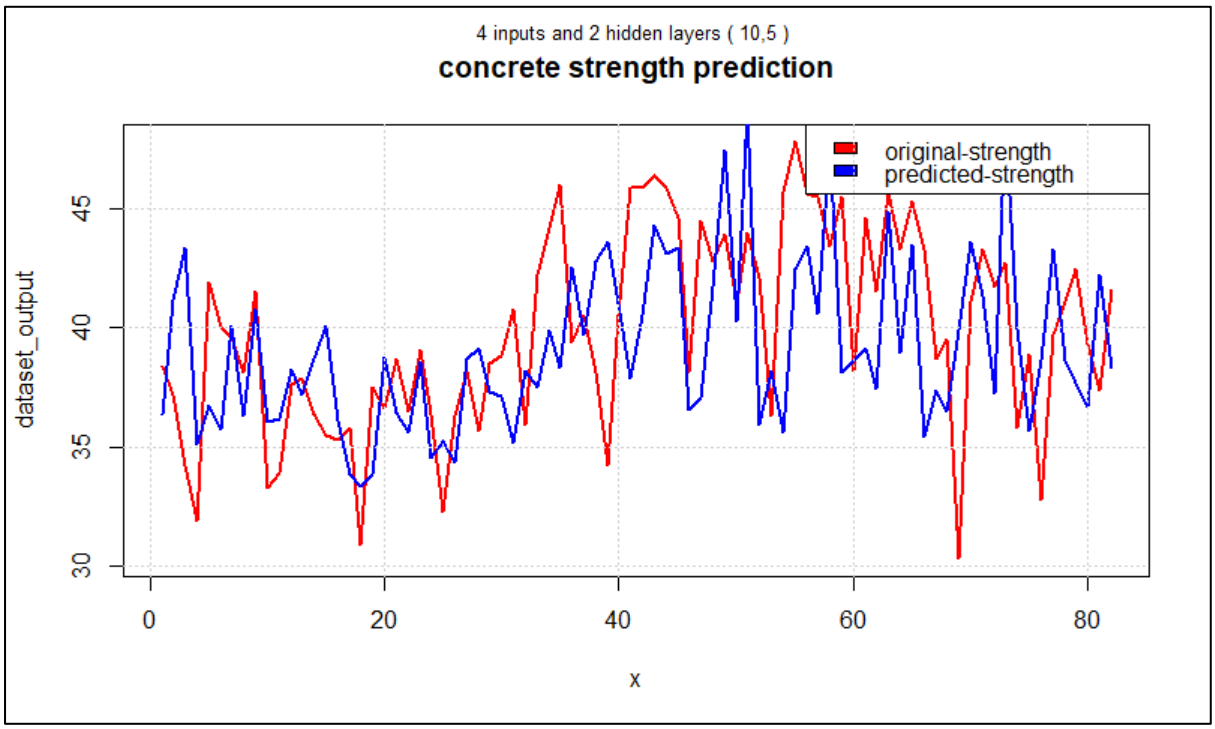


Figure 56: Best two hidden layer

2.2. 2nd Subtask Objective

2.2.1. NARX approach

```
204
205 #combining columns six and seven
206 time_delayed_matrix <- cbind(new_dataset_frame[,2:3], main_time_delayed_matrix)
207
```

Figure 57: combining six and seven hourn

This code adds two columns from the "new_dataset_frame" (which looks to be a data frame) to the left side of the "main_time_delayed_matrix" data frame using the "cbind()" method. The second and third columns are the two columns that are being added to "new_dataset_frame". These two extra columns will be present at the beginning of the resulting data frame "time_delayed_matrix," followed by the other columns of the "main_time_delayed_matrix."

2.2.2. Normalization

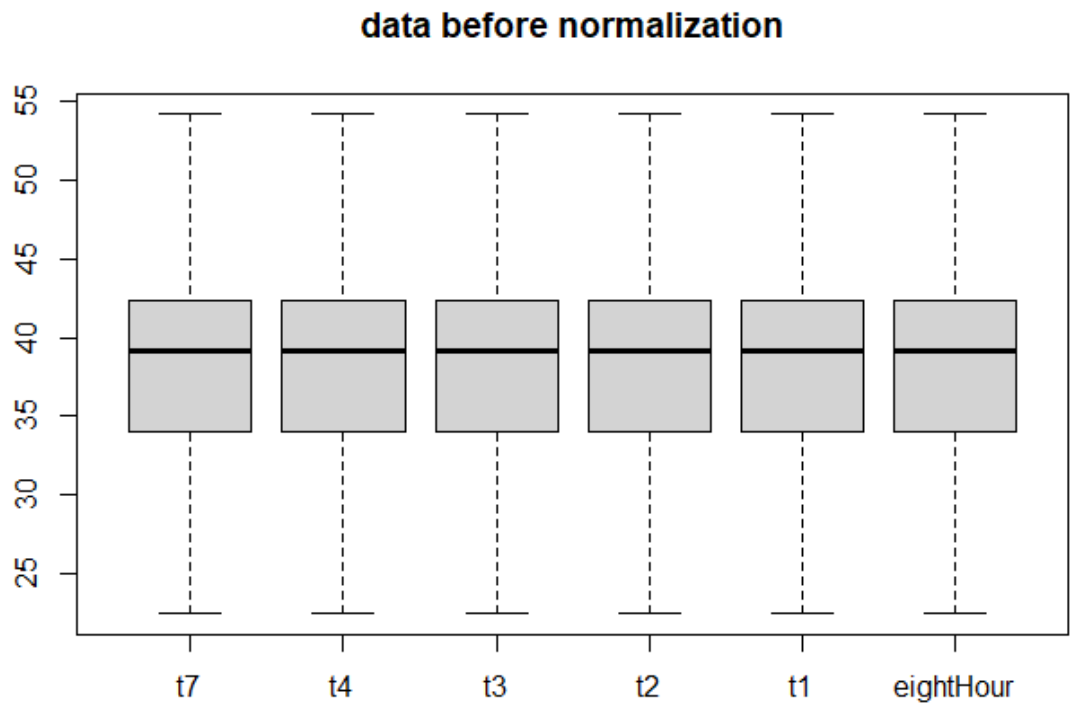
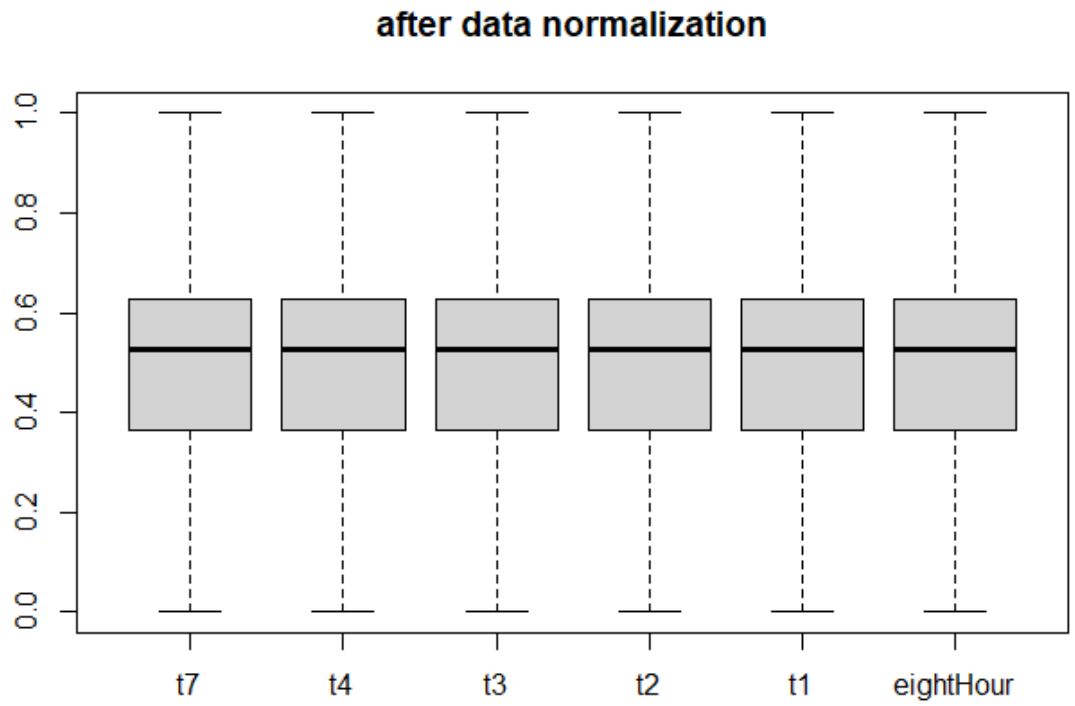
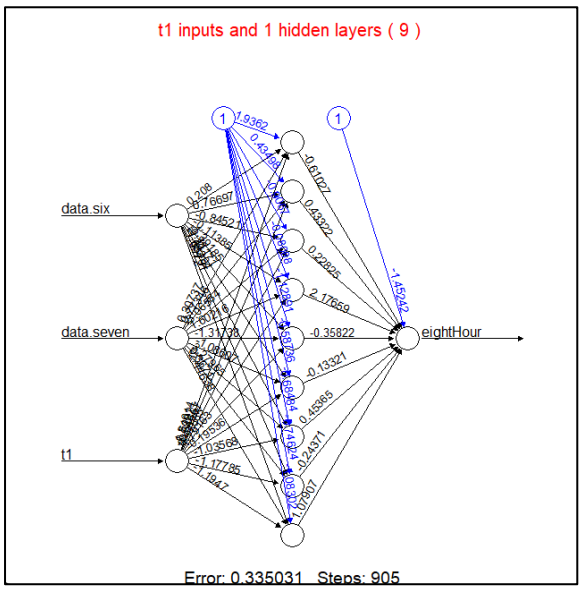
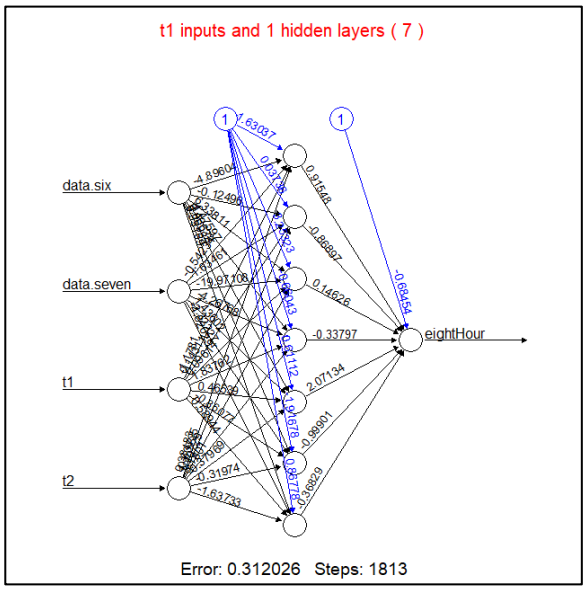
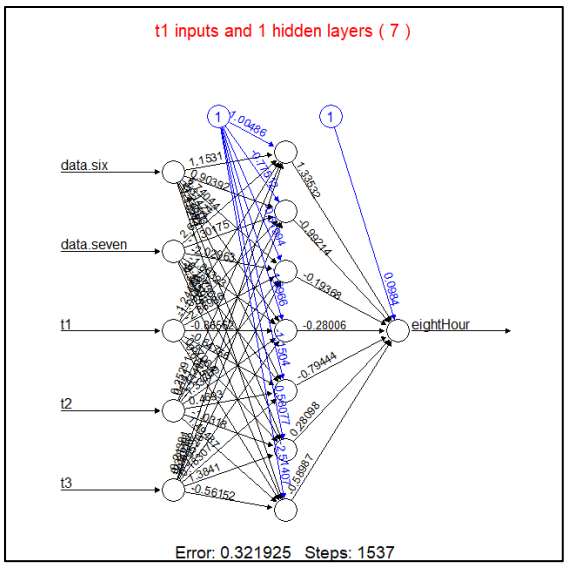


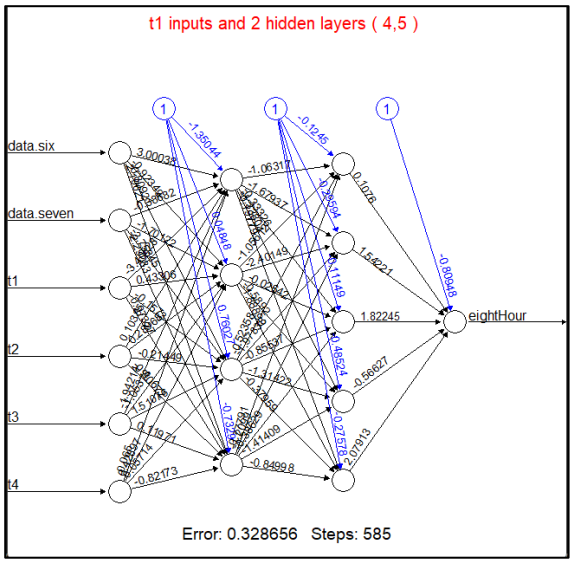
Figure 58: data before normalization

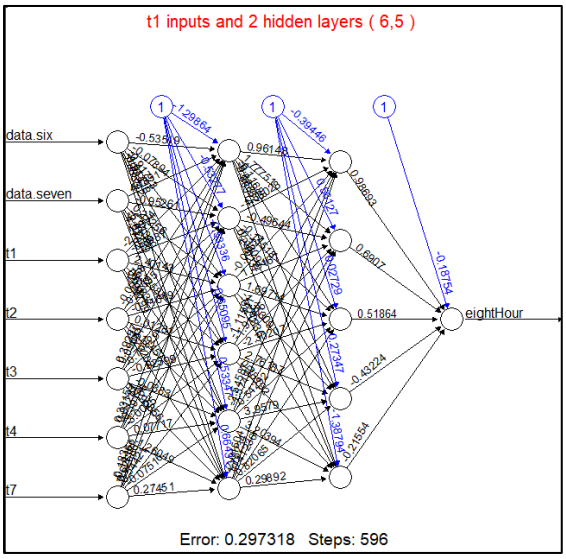

after data normalization
0.0
0.2
0.4
0.6
0.8
1.0
t7
t4
t3
t2
t1
eightHour

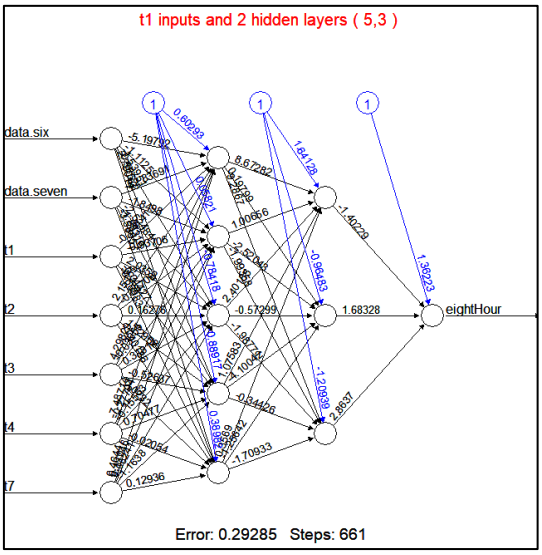

t1 inputs and 1 hidden layers (9)
data.six
data.seven
t1
eightHour
Error: 0.335031 Steps: 905


t1 inputs and 1 hidden layers (7)
data.six
data.seven
t1
t2
eightHour
Error: 0.312026 Steps: 1813


t1 inputs and 1 hidden layers (7)
data.six
data.seven
t1
t2
t3
eightHour
Error: 0.321925 Steps: 1537


t1 inputs and 2 hidden layers (4,5)
data.six
data.seven
t1
t2
t3
t4
eightHour
Error: 0.328656 Steps: 585


t1 inputs and 2 hidden layers (6,5)
data.six
data.seven
t1
t2
t3
t4
t7
eightHour
Error: 0.297318 Steps: 596


t1 inputs and 2 hidden layers (5,3)
data.six
data.seven
t1
t2
t3
t4
t7
eightHour
Error: 0.29285 Steps: 661

Input count	Hidden layers	Neurons Count	Accuracy	RMSE	MAE	MAPE	MAPE
3	1	9	94.24 %	2.661021	2.272259	0.05774834	0.05546043
4	1	7	94.21 %	2.708545	2.301916	0.05807728	0.05575574
5	1	7	94.1 %	2.700317	2.330946	0.0590455	0.05671211
6	2	4,5	94.2 %	2.658656	2.294479	0.05798417	0.05574391
7	2	6,5	94.27 %	2.643327	2.277073	0.05750137	0.05529614
7	2	5,3	93.99 %	2.748955	2.384929	0.0603456	0.05795587

2.2.3. Best MLP network

2.2.3.1. Best MLP network for AR approach

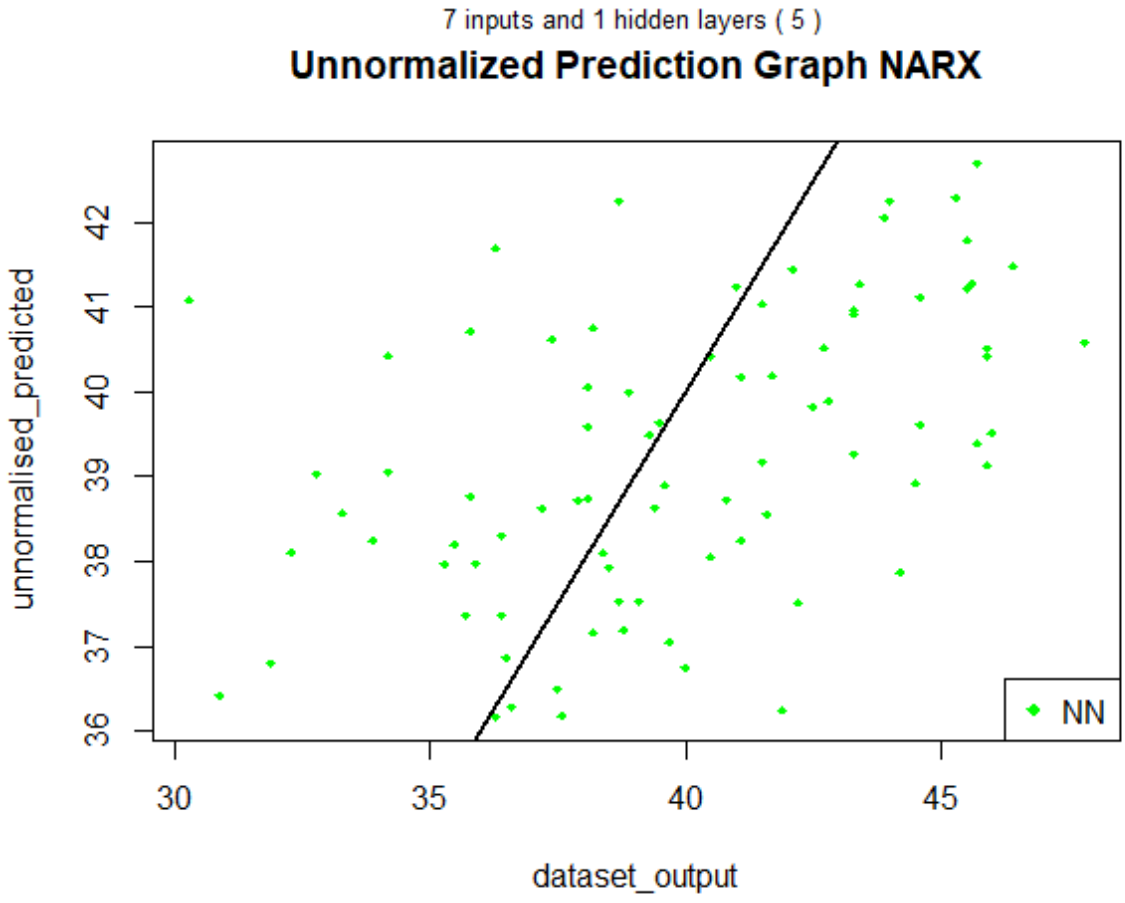


Figure 59Bset MLP network AR approach

2.2.3.2. *Best MLP network NARX approach*

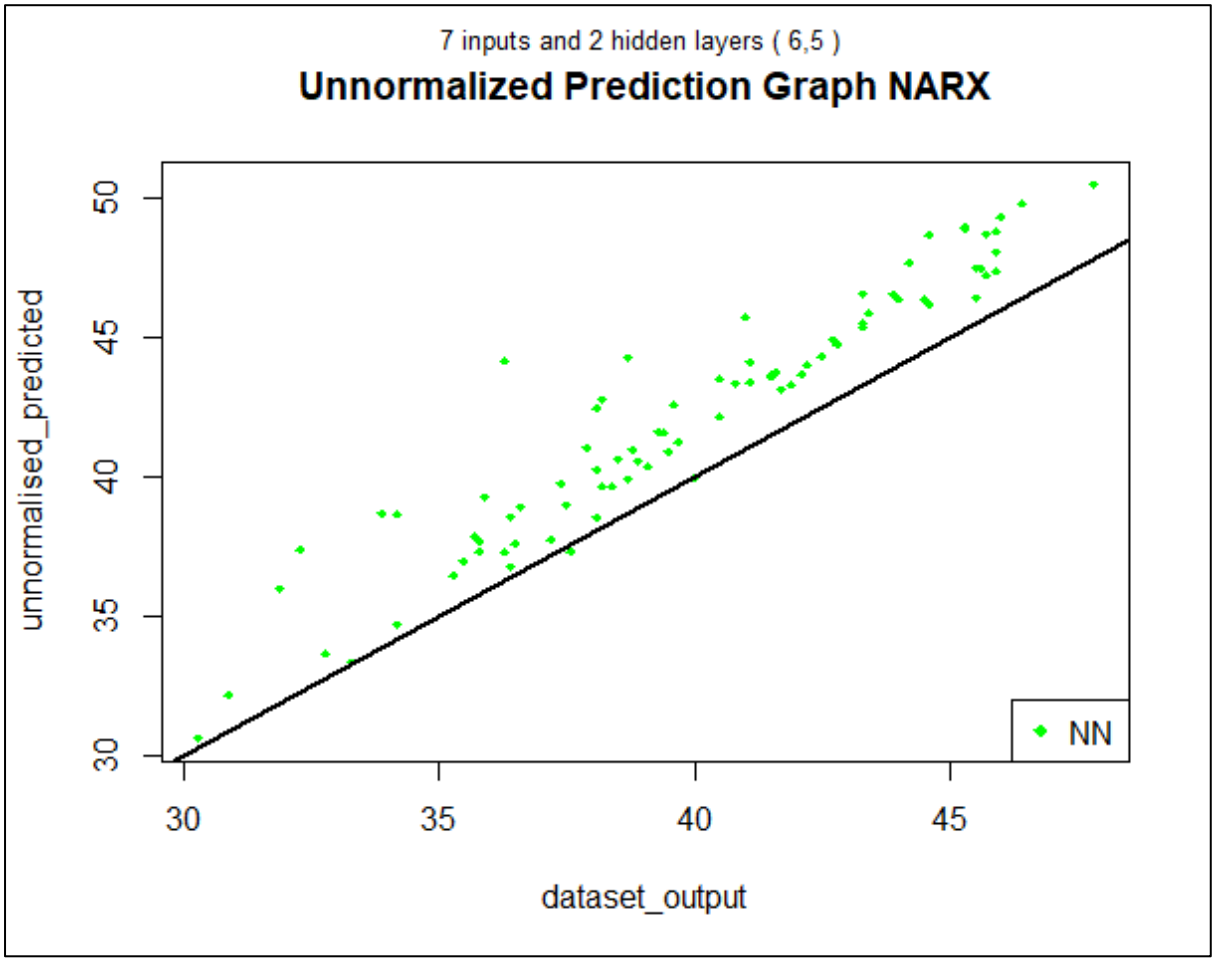


Figure 60: Best MLP network NARX approach

3. References

- A Guide to Principal Component Analysis (PCA) for Machine Learning. (no date). Available from https://www.keboola.com/blog/pca-machine-learning#:~:text=Principal%20Component%20Analysis%20(PCA)%20is,%2Dnoising%2C%20and%20plenty%20more. [Accessed 17 March 2023].

- Clustering in Machine Learning - Javatpoint. (no date). *www.javatpoint.com*. Available from https://www.javatpoint.com/clustering-in-machine-learning [Accessed 16 March 2023].

- Clustering in Machine Learning. (2018). *GeeksforGeeks*. Available from https://www.geeksforgeeks.org/clustering-in-machine-learning/ [Accessed 16 March2023].

- Ecosystem (LEDU), E. (2018). Understanding K-means Clustering in Machine Learning. *Medium*. Available from https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1 [Accessed 2 May 2023].

- Harrison, M. (2014). PCA and K-means Clustering of Delta Aircraft | R-bloggers. Available from https://www.r-bloggers.com/2014/06/pca-and-k-means-clustering-of-delta-aircraft/ [Accessed 4 May 2023].

- Plot Predicted vs. Actual Values in R (Example) | Draw Fitted & Observed. (no date). Statistics Globe. Available from https://statisticsglobe.com/plot-predicted-vs-actual-values-in-r/ [Accessed 2 May 2023].

- R Course. Forecasting Models. (no date). Available from https://www.youtube.com/watch?v=-pFBIGLWRFI&list=PLAHhszqeCDAMSNUULuTc6MuG3rHGMVRZB [Accessed 4 May 2023].

- What are Neural Networks? | IBM. (no date). Available from https://www.ibm.com/topics/neural-networks [Accessed 20 March 2023].

4. Appendix

Part1.R

```r
#part 1

library(readxl)
library(dplyr)
library(fpc)
library(MASS)
library(ggplot2)
library(ggcorrplot)
library(caret)
library(flexclust)
library(factoextra)
library(NbClust)
library(caret)
library(ggfortify)
library(FactoMineR)
library(readxl)
library(cluster)

# Read the XLSX file
vehicles_data <- read_xlsx("/Users/sandaru/Desktop/ML/CW/ML1_CW/vehicles.xlsx")

#Output variable removed (class attribute)
vehicles_data <- vehicles_data[, 2:19]
boxplot(vehicles_data, main = "Before Outlier Removal", outcol="red")
View(vehicles_data)

#Create a function to eliminate outliers from a single column using the boxplot method
remove_outliers <- function(x) {
  bp <- boxplot.stats(x)$stats
  x[x < bp[1] | x > bp[5]] <- NA
  return(x)
}

#Apply the function to each data frame column
vehicles_data <- apply(vehicles_data, 2, remove_outliers)

#Eliminate any rows with empty values (remove)
vehicles_data <- na.omit(vehicles_data)
boxplot(vehicles_data, main = "After Outlier Removal", outcol="red")

#Data scaling
scaled_vehicles_data <- scale(vehicles_data)
boxplot(scaled_vehicles_data, main = "Scale the data set")
```

```r
boxplot(scaled_vehicles_data, main = "Scale the data set")
head(scaled_vehicles_data)

set.seed(1234)
NBcluster <- NbClust(scaled_vehicles_data, min.nc = 2,max.nc = 10, method = "kmeans")
table(NBcluster$Best.n[1,])


#-------------------------------------------------------------------------------------------------------------
#elbow_method
fviz_nbclust(scaled_vehicles_data,kmeans,method = "wss")

#silhouette_method
fviz_nbclust(scaled_vehicles_data,kmeans,method = "silhouette")

#gap_static_method
fviz_nbclust(scaled_vehicles_data,kmeans,method = "gap_stat")

#-------------------------------------------------------------------------------------------------------------
# kmean %2 clusters
k2 <-kmeans(scaled_vehicles_data, 2)
k2
autoplot(k2,scaled_vehicles_data,frame=TRUE)

#Extract relevant data when k = 2.
# Cluster centers
clus_center <- k2$centers

#clustered outcomes
clus_assi <- k2$cluster

#Calculation BSS over TSS when k = 2
BSSk2 <- k2$betweenss #BSS
WSSk2 <- k2$tot.withinss #WSS
TSSk2 <- BSSk2 + WSSk2 #TSS

# BSS/TSS ratio
BSS_TSS_ratiok2 <- BSSk2 / TSSk2

#Explained variance as a percentage
percent_vark2 <- round(BSS_TSS_ratiok2 * 100, 3)

#Output results
```

```r
#Output results
cat("Cluster centers:\n", clus_center, "\n\n")
cat("Cluster assignments:\n", clus_assi, "\n\n")
cat("BSS/TSS ratio: ", round(BSS_TSS_ratiok2, 3), "\n\n")
cat("BSS: ", round(BSSk2, 3), "\n\n")
cat("WSS: ", round(WSSk2, 3), "\n\n")
cat("Explained variance as a percentage: ", percent_vark2, "%\n")

#3 clusters
k3 <-kmeans(scaled_vehicles_data, 3)
k3
autoplot(k3,scaled_vehicles_data,frame=TRUE)

#Extract relevant data when k = 3
# Cluster centers
clus_center <- k3$centers

#clustered outcomes
clus_assi <- k3$cluster

#Calculation BSS over TSS when k = 3
BSSk3 <- k3$betweenss #BSS
WSSk3 <- k3$tot.withinss #WSS
TSSk3 <- BSSk3 + WSSk3 #TSS

# BSS/TSS ratio
BSS_TSS_ratiok3 <- BSSk3 / TSSk3

#Explained variance as a percentage
percent_vark3 <- round(BSS_TSS_ratiok3 * 100, 3)

# Output results
cat("Cluster centers:\n", clus_center, "\n\n")
#cat("Cluster assignments:\n", cluster_assignments, "\n\n")
cat("Cluster assignments:\n", clus_assi, "\n\n")
cat("BSS/TSS ratio: ", round(BSS_TSS_ratiok3, 3), "\n\n")
cat("BSS: ", round(BSSk3, 3), "\n\n")
cat("WSS: ", round(WSSk3, 3), "\n\n")
cat("Explained variance as a percentage: ", percent_vark3,"%\n")

#Fit k-means model with k=2
k <- 2
kmeans_model <- kmeans(scaled_vehicles_data, centers = k, nstart = 25)
kmeans_model <- kmeans(scaled_vehicles_data, centers = k, nstart = 25)

#Create a silhouette plot
silhouette_plot <- silhouette(kmeans_model$cluster, dist(scaled_vehicles_data))

#Compute the average silhouette's width
avg_sil_width <- mean(silhouette_plot[, 3])

#Plot the silhouette plot
plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
     xlab = "Silhouette Width", ylab = "Cluster", border = NA)

#As a vertical line, add the average silhouette width
abline(v = avg_sil_width, lty = 2, lwd =2,col="red")

#Fit k-means model with k=3
k <- 3
kmeans_model <- kmeans(scaled_vehicles_data, centers = k, nstart = 25)

#Generate silhouette plot
silhouette_plot <- silhouette(kmeans_model$cluster, dist(scaled_vehicles_data))

#Calculate average silhouette width
avg_sil_width <- mean(silhouette_plot[, 3])

#Plot the silhouette plot
plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
     xlab = "Silhouette Width", ylab = "Cluster", border = NA)

#Add average silhouette width as vertical line
abline(v = avg_sil_width, lty = 2, lwd =2,col="red")

#-------------------------------------------------------------------------------------------------------------------------
#part_02
pca <- prcomp(scaled_vehicles_data)

#The eigenvalues and eigenvectors in print
print(summary(pca))

#Calculate the total score for each of the principal components (PC)
pca_var <- pca$sdev^2
pca_var_prop <- pca_var / sum(pca_var)
pca_var_cumprop <- cumsum(pca_var_prop)
```

```r
#Plot cumulative score per PC
plot(pca_var_cumprop, xlab = "Principal Component", ylab = "Cumulative Proportion of Variance Explained",
     ylim = c(0, 1), type = "b")

#Convert an existing dataset and add attributes that correspond to the primary components
pca_trans <- predict(pca, newdata = scaled_vehicles_data)

#Choose PCs that provide at least cumulative score > 92%
selected_pcs <- which(pca_var_cumprop > 0.92)
transformed_data <- pca_trans[, selected_pcs]

boxplot(transformed_data)

set.seed(1234)
NBcluster <- NbClust(transformed_data, min.nc = 2,max.nc = 10, method = "kmeans")
table(NBcluster$Best.n[1,])


#-------------------------------------------------------------------------------------------------------------------------
#elbow_method
fviz_nbclust(transformed_data,kmeans,method = "wss")

#silhouette_method
fviz_nbclust(transformed_data,kmeans,method = "silhouette")

#gap static_method
fviz_nbclust(transformed_data,kmeans,method = "gap_stat")

#Perform k-means clustering with k=2
set.seed(1234)
kmeans_model_2 <- kmeans(transformed_data, centers = 2, nstart = 25)

#The k-means output in print
print(kmeans_model_2)
autoplot(kmeans_model_2,transformed_data,frame=TRUE)

#Calculate the within-cluster sum of squares (WSS)
wss_2 <- sum(kmeans_model_2$withinss)

#Calculate the between-cluster sum of squares (BSS)
bss_2 <- sum(kmeans_model_2$size * dist(rbind(kmeans_model_2$centers, colMeans(transformed_data)))^2)

#Calculate the total sum of squares (TSS)
```

```r
#Calculate the total sum of squares (TSS)
tss_2 <- sum(dist(transformed_data)^2)

#Calculate the ratio of BSS to TSS
bss_tss_ratio_2 <- bss_2 / tss_2

#Print the WSS, BSS, TSS, and ratio of BSS to TSS
cat("For k=2:\n")
cat("Sum of squares within a cluster (WSS): ", wss_2, "\n")
cat("Between-cluster square sum (BSS): ", bss_2, "\n")
cat("sum of all squares (TSS): ", tss_2, "\n")
cat("Ratio of BSS to TSS: ", bss_tss_ratio_2, "\n\n")

#Perform k-means clustering with k=3
set.seed(1234)
kmeans_model_3 <- kmeans(transformed_data, centers = 3, nstart = 25)

#The k-means output in print
print(kmeans_model_3)
autoplot(kmeans_model_3,transformed_data,frame=TRUE)

#Calculate the squares' within-cluster sum(WSS)
wss_3 <- sum(kmeans_model_3$withinss)

#Calculate the squares' between-cluster sum (BSS)
bss_3 <- sum(kmeans_model_3$size * dist(rbind(kmeans_model_3$centers, colMeans(transformed_data)))^2)

#Calculate the sum of all the squares (TSS)
tss_3 <- sum(dist(transformed_data)^2)

#Calculate the BSS to TSS ratio.
bss_tss_ratio_3 <- bss_3 / tss_3

#Print out the WSS, BSS, TSS, and the BSS/TSS ratio
cat("For k=3:\n")
cat("Sum of squares within a cluster (WSS): ", wss_3, "\n")
cat("Between-cluster square sum (BSS): ", bss_3, "\n")
cat("sum of all squares (TSS): ", tss_3, "\n")
cat("Ratio of BSS to TSS: ", bss_tss_ratio_3, "\n")


#Fit k-means model with k=2
k <- 2
```

```r
k <- 2
kmeans_Model <- kmeans(transformed_data, centers = k, nstart = 25)

#Create a silhouette plot
silhouette_plot <- silhouette(kmeans_Model$cluster, dist(transformed_data))

#Calculate average silhouette width
avg_sil_width <- mean(silhouette_plot[, 3])

#Plot the silhouette plot
plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
     xlab = "Silhouette Width", ylab = "Cluster", border = NA)

#As a vertical line, add the average silhouette width
abline(v = avg_sil_width, lty = 2, lwd =2,col="red")

# Fit k-means model with k=3
k <- 3
kmeans_model <- kmeans(transformed_data, centers = k, nstart = 25)

#Create a silhouette plot
silhouette_plot <- silhouette(kmeans_model$cluster, dist(transformed_data))

#Calculate average silhouette width
avg_sil_width <- mean(silhouette_plot[, 3])

#Plot the silhouette plot
plot(silhouette_plot, main = paste0("Silhouette Plot for k =", k),
     xlab = "Silhouette Width", ylab = "Cluster", border = NA)

#Add average silhouette width as vertical line
abline(v = avg_sil_width, lty = 2, lwd =2,col="red")

#Calculate Calinski-Harabasz Index k =2
calinski_harabasz_pca <- function(cluster_result, data) {
  k2 <- length(unique(cluster_result$cluster))
  n2 <- nrow(data)
  BSS2 <- cluster_result$betweenss
  WSS2 <- cluster_result$tot.withinss

  ch_index2 <- ((n2 - k2) / (k2 - 1)) * (BSS2 / WSS2)
  return(ch_index2)
}
```

```r
}

ch_index_pca_2 <- calinski_harabasz_pca(kmeans_Model, transformed_data)
ch_index_pca_2


#Calculate Calinski-Harabasz Index k =3
calinski_harabasz_pca <- function(cluster_result, data) {
  k3 <- length(unique(cluster_result$cluster))
  n3 <- nrow(data)
  BSS3 <- cluster_result$betweenss
  WSS3 <- cluster_result$tot.withinss

  ch_index3 <- ((n3 - k3) / (k3 - 1)) * (BSS3 / WSS3)
  return(ch_index3)
}

ch_index_pca_3 <- calinski_harabasz_pca(kmeans_model, transformed_data)
ch_index_pca_3
```

Part-2.R

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

#Get the uow_consumption dataset loaded
data <- read_excel("/Users/sandaru/Desktop/ML/CW/ML1_CW/vehicles.xlsx", sheet = 1)

#---------------------------------------Preprocessing-------------------------------------------
names(data)[2] <- 'six'
names(data)[3] <- 'seven'
names(data)[4] <- 'eight'

# change date to numeric
date <-factor(data$date)
date <-as.numeric(date)
date

# create data frame
new_dataset_frame <- data.frame(data,data$'six',data$'seven',data$'eight')

# create dataframe for 8
eight_column <- c(new_dataset_frame$data.eight)
plot(eight_column, type = "l")

# create I/O matrix
main_time_delayed_matrix <- bind_cols(t7 = lag(eight_column,8),
                                      t4 = lag(eight_column,5),
                                      t3 = lag(eight_column,4),
                                      t2 = lag(eight_column,3),
                                      t1 = lag(eight_column,2),
                                      eightHour = eight_column)

time_delayed_matrix <- na.omit(main_time_delayed_matrix)

#dividing the data into train and test
train_data <- time_delayed_matrix[1:380,]
test_data <- time_delayed_matrix[381:nrow(time_delayed_matrix),]

#max min values
min_val <- min(train_data)
max_val <- max(train_data)
dataset_output <- test_data$eightHour

#---------------------------------------normalize-------------------------------------------
#normalization function for min-max
normalize <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}

# un-normalizing function
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}
#apply Normalization
time_delayedNorm <- as.data.frame(lapply(time_delayed_matrix[1:ncol(time_delayed_matrix)], normalize))

#after normalizing test data, separating the data
train_dataset_norm <- time_delayedNorm[1:380,]
test_dataNorm <- time_delayedNorm[381:nrow(time_delayed_matrix),]

head(time_delayed_matrix)

#before & after normalization
boxplot(time_delayed_matrix, main="data before normalization")
boxplot(time_delayedNorm, main="after data normalization")

#the generation of testing data for each timeDelay
t1_testing_data <- as.data.frame(test_dataNorm[, c("t1")])
t2_testing_data <- test_dataNorm[, c("t1", "t2")]
t3_testing_data <- test_dataNorm[, c("t1", "t2", "t3")]
t4_testing_data <- test_dataNorm[, c("t1", "t2", "t3", "t4")]
t7_testing_data<- test_dataNorm[, c("t1", "t2", "t3", "t4", "t7")]

#ability to train an AR model
trainModel <- function(formula, hiddenVal, isLinear, actFunc,inputs,hidden){

  my_text <- paste(inputs,"inputs and",length(hidden),"hidden layers","(",paste(hidden, collapse=","),") \n")

  set.seed(1234)
  nn <- neuralnet(formula,data = train_dataset_norm, hidden=hiddenVal, act.fct = actFunc, linear.output=isLinear)
```

```
  nn <- neuralnet(formula,data = train_dataset_norm, hidden=hiddenVal, act.fct = actFunc, linear.output=isLinear)
  plot(nn)

  plot_panel <- grid.grab(wrap = TRUE)

  ## create a title grob:
  plot_title <- textGrob(my_text,
                         x = .5, y = .50,
                         gp = gpar(lineheight = 2,
                                   fontsize = 15, col = 'red',
                                   adj = c(1, 0)
                         )
  )

  #stack title and main panel, and plot:
  grid.arrange(
    grobs = list(plot_title,
                 plot_panel),
    heights = unit(c(.15, .85), units = "npc"),
    width = unit(1, "npc")
  )
  dev.new()
  dev.off()
  return(nn)
}

testing_Model <- function(nnModel, testing_df,inputs,hidden){
  cat("There are",inputs,"inputs and",length(hidden),"hidden layers","(",paste(hidden, collapse=","),") \n")
  my_text <- paste(inputs,"inputs and",length(hidden),"hidden layers","(",paste(hidden, collapse=","),") \n")

  nnresults <- compute(nnModel, testing_df)
  predicted <- nnresults$net.result
  unnormalised_predicted <- unnormalize(predicted, min_val, max_val)
  devia = ((dataset_output - unnormalised_predicted)/dataset_output)
  modelAccuracy = 1 - abs(mean(devia))
  accuracy = round(modelAccuracy * 100 , digits = 2)

  plot( dataset_output,unnormalised_predicted, col = 'green', main = "Unnormalized Prediction Graph NARX", pch = 18, cex = 0.7)
  mtext(my_text,  side = 3, line = 2, cex = 0.8)
  abline(0,1,lwd=2)
  legend("bottomright", legend = 'NN', pch = 18, col = 'green')
  dev.new()
  dev.off()


  x = 1:length(dataset_output)
  plot(x, dataset_output, col = "red", type = "l", lwd=2,
       main = "concrete strength prediction")
  mtext(my_text,  side = 3, line = 2, cex = 0.8)
  lines(x, unnormalised_predicted, col = "blue", lwd=2)
  legend("topright", legend = c("original-strength", "predicted-strength"),
         fill = c("red", "blue"), col = 2:3, adj = c(0, 0.6))
  grid()
  dev.new()
  dev.off()


  rmse = rmse(dataset_output, unnormalised_predicted)
  mae = mae(dataset_output, unnormalised_predicted)
  mape = mape(dataset_output, unnormalised_predicted)
  smape = smape(dataset_output, unnormalised_predicted)

  cat("Model Accuracy:", accuracy, "%\n")
  cat("RMSE:", rmse, "\n")
  cat("MAE:", mae, "\n")
  cat("MAPE:", mape, "\n")
  cat("sMAPE:", smape, "\n")
  cat("\n\n")

  return(unnormalised_predicted)
}


#t1 Train models using various hidden layer sizes.
hidden_layers_count <- list( c(8),c(5, 3))

for (i in seq_along(hidden_layers_count)) {
  model <- trainModel(eightHour ~ t1, hidden_layers_count[[i]], isLinear = FALSE, "tanh",1,hidden_layers_count[[i]])
  pred <- testing_Model(model, t1_testing_data,1,hidden_layers_count[[i]])

}
```

```r
# t2 Train models using various hidden layer sizes.
t2_training <- trainModel(eightHour ~ t1 + t2, c(8),isLinear = TRUE, "logistic",2,c(8))
test_t2_predict <- testing_Model(t2_training, t2_testing_data,2,c(8))



#t3 Train models using various hidden layer sizes.
hidden_layers_count <- list( c(4),c(7),c(6,8))

for (i in seq_along(hidden_layers_count)) {
  model <- trainModel(eightHour ~ t1 + t2 + t3 ,hidden_layers_count[[i]],isLinear = TRUE, "logistic",3,hidden_layers_count[[i]])

  pred <- testing_Model(model, t3_testing_data,3,hidden_layers_count[[i]])

}


#t4 Train models using various hidden layer sizes.
hidden_layers_count <- list( c(5),c(9),c(5,2),c(10,5))

for (i in seq_along(hidden_layers_count)) {
  model <- trainModel(eightHour ~ t1 + t2 + t3 + t4,hidden_layers_count[[i]],isLinear = TRUE, "logistic",4,hidden_layers_count[[i]])
  pred <- testing_Model(model, t4_testing_data,4,hidden_layers_count[[i]])

}


#t7 Train models using various hidden layer sizes.
hidden_layers_count <- list( c(5),c(10),c(6,2),c(10,8),c(7,6))

for (i in seq_along(hidden_layers_count)) {
  model <- trainModel(eightHour ~ t1 + t2 + t3 + t4 + t7,hidden_layers_count[[i]],isLinear = TRUE, "logistic",7,hidden_layers_count[[i]])
  pred <- testing_Model(model, t7_testing_data,7,hidden_layers_count[[i]])

}
#-----------------------------------------substask 02--------------------------------------#

#combining columns six and seven
time_delayed_matrix <- cbind(new_dataset_frame[,2:3], main_time_delayed_matrix)

time_delayed_matrix <- na.omit(time_delayed_matrix)

#dividing the data into train and test
train_data <- time_delayed_matrix[1:380,]
test_data <- time_delayed_matrix[381:nrow(time_delayed_matrix),]

#max min values
min_val <- min(train_data)
max_val <- max(train_data)
dataset_output <- test_data$eightHour

# apply Normalization
time_delayedNorm <- as.data.frame(lapply(time_delayed_matrix[1:ncol(time_delayed_matrix)],
                                         normalize))
#splitting data after normalize test data
train_dataset_norm <- time_delayedNorm[1:380,]
test_dataNorm <- time_delayedNorm[381:nrow(time_delayed_matrix),]

#view before & after normalization
boxplot(time_delayed_matrix, main="data before normalization")
boxplot(time_delayedNorm, main="after data normalization")

#the generation of testing data for each timeDelay
t1_testing_data <- test_dataNorm[, c("data.six","data.seven", "t1")]
t2_testing_data <- test_dataNorm[, c("data.six","data.seven", "t1", "t2")]
t3_testing_data <- test_dataNorm[, c("data.six","data.seven", "t1", "t2", "t3")]
t4_testing_data <- test_dataNorm[, c("data.six","data.seven", "t1", "t2", "t3", "t4")]
t7_testing_data<- test_dataNorm[, c("data.six","data.seven", "t1", "t2", "t3", "t4", "t7")]

#the input characteristics and the appropriate test data should be defined.
inputs <- c( "t1", "t2", "t3","t4","t7")
test_data <- list(t1_testing_data, t2_testing_data, t3_testing_data,t4_testing_data,t7_testing_data)


#Train models using various hidden layer sizes.

h1 <- list(c(9))
h2 <- list(c(7))
h3 <- list(c(7))
h4 <- list(c(4,5))
h5 <- list(c(6,5),c(5,3))

hidden_layers_count <- list(h1,h2,h3,h4,h5)

```

```
250
251
252 for (i in seq_along(inputs)) {
253   for(j in seq_along(hidden_layers_count[[i]])){
254
255     current_inputs <- inputs[1:i]
256     # cat(current_inputs)
257
258     # Get the current hidden layer configuration
259     currentHidden <- hidden_layers_count[[i]][[j]]
260
261     # Train the model
262     formula <- as.formula(paste("eightHour ~ data.six + data.seven +", paste(current_inputs, collapse="+")))
263
264
265     model <- trainModel(formula, currentHidden,isLinear = TRUE, "logistic",current_inputs,currentHidden)
266
267     # Test the model
268     test_predict <- testing_Model(model, test_data[[i]],(length(current_inputs)+2),currentHidden)
269
270   }
271 }
272
```