INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTING

(B.Eng.) in Software Engineering

Module code: 5COSC021C

Machine Learning and Data Mining

Module Leader: Mr. Nipuna Senanayake

Student Name : Gorakapitiyage Induranga Kawishwara

Student ID: 20200688

UoW ID : w1913278

Table of Contents

TABLE OF FIGURES

TABLES

1. Partitioning Clustering

1.1. 1st Subtask Objectives

1.1.1. Pre-processing Task

Loading several libraries is part of the code for data analysis. After that, the "read_excel()" method from the "readxl" library is used to read an Excel file called "vehicles.xlsx" The -1 number is used to get rid of the first column of the data set, and the data set is split using the data variable to get rid of the string columns. Before outlier removal, a boxplot is made to show how the data is spread out, and a code called "remove_outliers()" is made to find outliers in a single column. The "boxplot.stats()" method is used to figure out the boxplot's median, quartiles, and limits. Values outside these limits are replaced with "NA," which stands for "missing value." This function is applied to each column of the dataset using the "apply()" function. This creates a new dataset in which outlier numbers are replaced with "NA." Finally, a boxplot is made to show how the data are spread out after the outliers have been taken out.

Code part to get data from data file.

```
19
20 Vehicles_dataset <- read_excel("F://SEMESTER 02//5DATA001C.2 Machine Learning and Data Mining//CW//MyCW//vehicles.xlsx")
21
22
23
24 # Delete the first column.
25
26 Vehicles_dataset <- Vehicles_dataset[-1]
27
28
29 # Examine the data type to determine which columns are strings.
30
31
32 string_cols <- sapply(Vehicles_dataset, class) == "character"
33
34 #Use the subset function to eliminate the string column(s).
35
36 data <- Vehicles_dataset[, !string_cols]
37
38 View(data)
39
40 boxplot(data, main = "Before Outlier Removal", outcol="red")
41
42
43 # Before removing the outliers, count the rows and columns and print the results.
44
45 rows_of_dataset <- nrow(data)
46 columns_of_dataset <- ncol(data)
47
48 # print the results
49 cat("Rows to delete before outliers   :", rows_of_dataset, "\n")
50 cat("columns to delete before outliers:", columns_of_dataset, "\n")
```

Figure 1 : get data from file

Code part to remove outliers.

```r

# Create a function to remove outliers from a single column using the boxplot method.

outliers_remove <- 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.

whithout_outliers_dataset <- apply(data, 2, outliers_remove)

# Remove any rows with missing values

whithout_outliers_dataset <- na.omit(whithout_outliers_dataset)


# After removing the outliers, count the rows and columns and print the results.

rows_of_dataset <- nrow(whithout_outliers_dataset)
columns_of_dataset <- ncol(whithout_outliers_dataset)

# print the results
cat("Rows to delete before outliers   :", rows_of_dataset, "\n")
cat("Columns to delete before outliers:", columns_of_dataset, "\n")


boxplot(whithout_outliers_dataset,main = "After Outlier Removal")

```

Figure 2:Code part to remove outliers

Below chart is a boxplot of before and after removing outliers

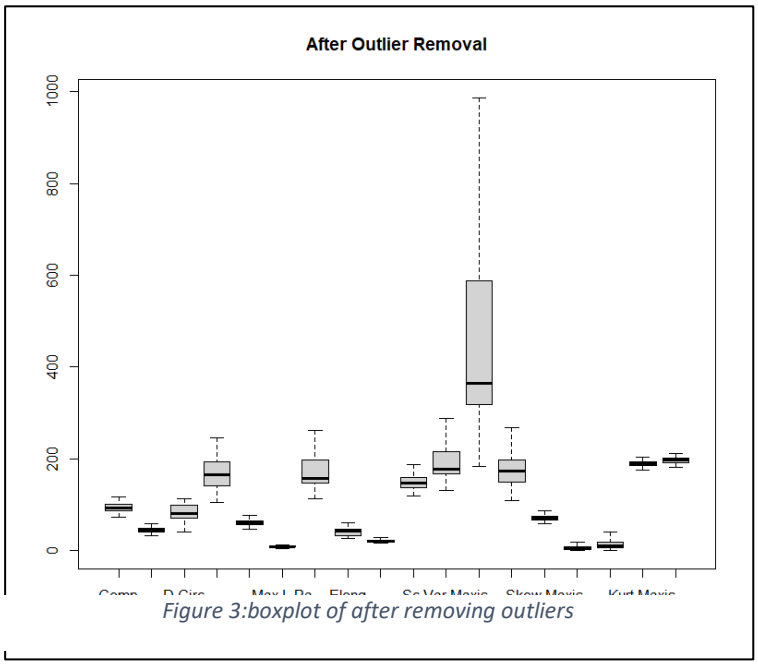


Figure 3:boxplot of after removing outliers

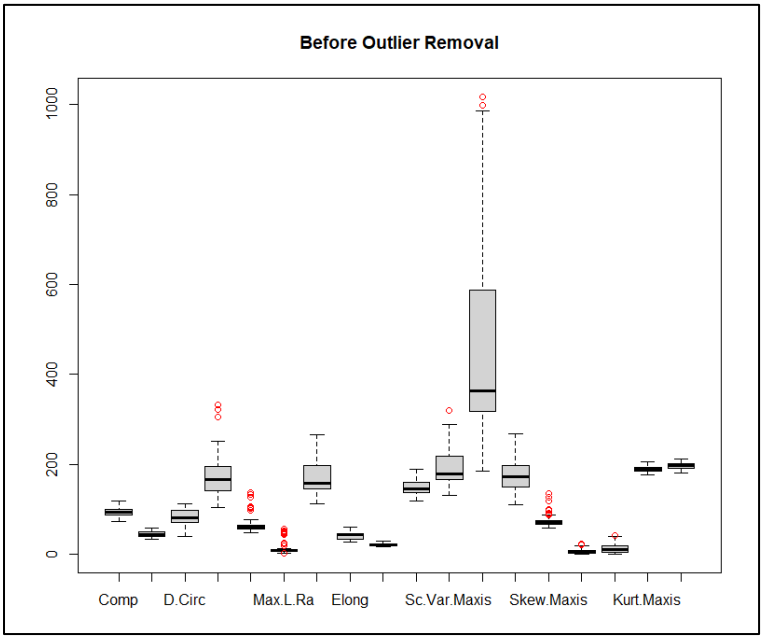


Figure 4:boxplot of before removing outliers

Code part to scaling data

The "scale ()" method is used to make the "whithout_outliers_dataset" have a mean of zero and a standard deviation of one. The "head ()" method shows the first few rows of the scaled dataset, and a boxplot shows how the variables in the scaled dataset are spread out. The center of the boxplot is zero, and the spread between the two quartiles is equal to two standard deviations. This makes it useful for finding outliers, skewness, and other general characteristics of how the variables are distributed.

```
83
84  # Scale up the data set.
85
86  scaled_Vehicles_dataset <- scale(whithout_outliers_dataset)
87  head(scaled_Vehicles_dataset)
88
89  boxplot(scaled_Vehicles_dataset)
90
91
92
93
```

Figure 5:scaling_code

Below chart is a boxplot of dataset after removing outliers and scaling

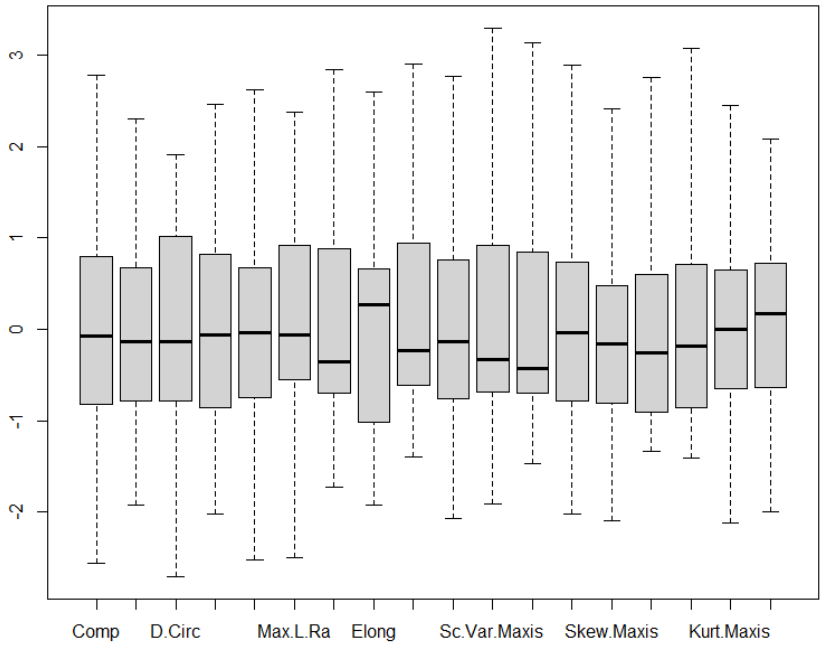


Figure 6:after removing outliers and scaling

1.1.2. Determine the number of cluster centers via four “automated tools.”

Having completed procedures like missing value management and outlier elimination, we have a clean data set suitable for subsequent processing, such as clustering. One of these techniques is called k-means, and it requires specifying the desired number of groups. The maximum number of clusters that can be created is different for each dataset. Some methods for counting the number of groups in a dataset are the Elbow method, the Gap statistics, and the silhouette method. Finding the most effective clusters in a dataset based on their shared characteristics is the primary focus of this stage.

To calculate the number of cluster centroids, I utilized the techniques below.

- NBclust
- Elbow method
- Gap statistics
- Silhouette method
-

1.1.2.1. Code section for NBclust

The Euclidean distance and the gap is used in the following NBclust method

```
94
95
96  # Nbclust method
97
98  # Set the random seed to 1234 for reproducibility
99
100 set.seed(1234)
101
102 NBcluster <- NbClust(scaled_Vehicles_dataset, min.nc = 2,max.nc = 10, method = "kmeans")
103
104 table(NBcluster$Best.n[1,])
105
```

Figure 7:NBclust code

See below. The best number of clusters is 3.

```
> set.seed(1234)
>
> NBcluster <- NbClust(scaled_Vehicles_dataset, 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 8:NBclust_output_1

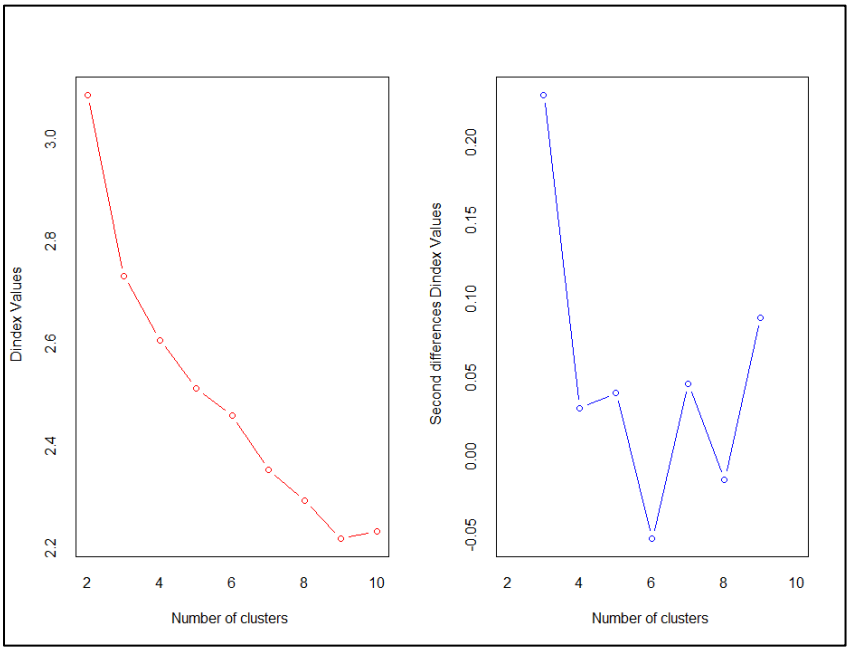


Figure 9:NBclust_output2

1.1.2.2. Code section for elbow function

The basic idea behind the elbow method is to find the exact number of clusters in a set of data by finding the "elbow point" in the graph where the elbow shape is made. To finish this job, the Within-cluster Sum of Square (WSS) calculation is used.

```
105
106  # elbow method
107
108  fviz_nbclust(scaled_Vehicles_dataset,kmeans,method = "wss")
109
```

Figure 10:elbow code

Output

By turning on the code part, which is 2, you can get the right cluster hub number for the dataset, which is shown below.

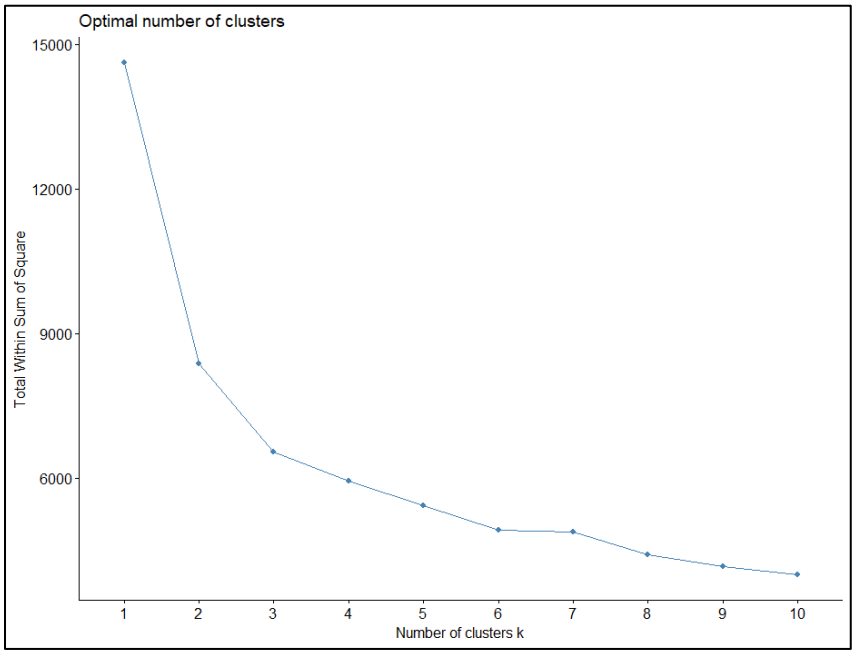


Figure 11:elbow_output

1.1.2.3. Code section for Silhouette method

The silhouette method can be used to figure out how well a dataset is clustered. To use this method, the Silhouette method is used to figure out how similar each vehicle is to its own cluster compared to other clusters. This is how the Silhouette coefficient for each vehicle is calculated. As a result, it calculates the average silhouette coefficient for each cluster to give a measure of the strength of clustering. By figuring out the average silhouette coefficient for different numbers of clusters, the Silhouette method can also be used to find the best number of clusters. If the average is higher, it means that the grouping is more accurate, and if it's lower, it means that the grouping is less accurate and needs to be fixed.

```
109
110 # silhouette method
111
112 fviz_nbclust(scaled_Vehicles_dataset,kmeans,method = "silhouette")
113
```

Figure 12:silhouette_code

Output

Running the code part shows that the dataset has two optimal cluster centers, as seen below.

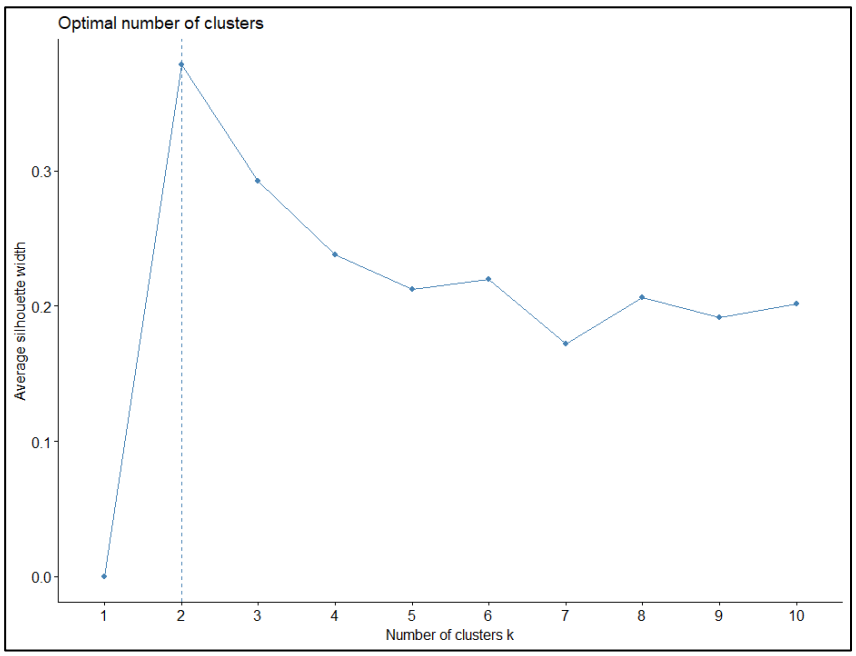


Figure 13:silhouette_output

1.1.2.4. Code section for Gap statistics method

The basic idea behind the static gap method is to compare the within-cluster dispersion of a clustering solution that references a data distribution with similar features but no clear clustering pattern. The static gap method can be used to find the best number of groups in a data set. But for the clustering solution to be more appropriate for the data, it should be used along with other methods and domain knowledge.

```
113
114  # gap static method
115
116  fviz_nbclust(scaled_Vehicles_dataset,kmeans,method = "gap_stat")
117
118
```

Figure 14:Gap

Output

When the code part is run, it shows that the dataset has 3 good centres for clusters, as shown below.

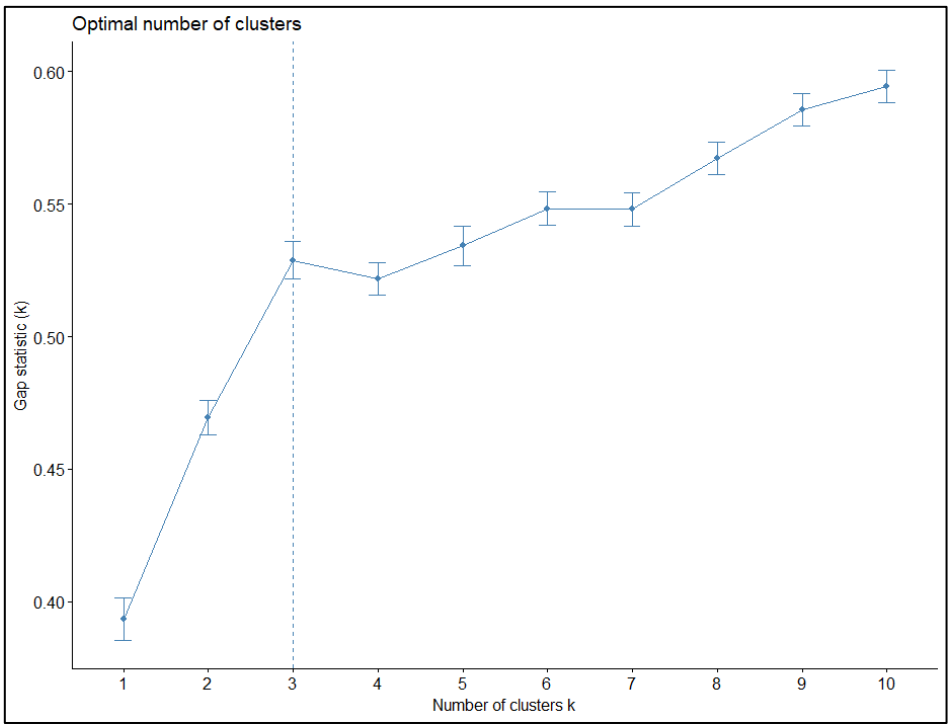


Figure 15:Gap_output

1.1.3. Kmeans clustering investigation

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

```
121
122  # 2 clusters
123  k2 <-kmeans(scaled_Vehicles_dataset, 2)
124  k2
125  autoplot(k2,scaled_Vehicles_dataset,frame=TRUE)
126
```

Figure 16:clustering_code(k=2)

Above code is used to do k-means clustering with two groups on the scaled_Vehicles_dataset. The clustering is done with the "kmeans()" method, and the resulting object is saved in the "k2" variable. For the plotting, the "autoplot()" function from the "ggfortify" package is used. The first input to the function is the "k2" object, the second is the original data set, and the frame is set to "TRUE" to put a bounding box around each cluster. The result is a plot that shows the two groups in different colours, with each point labelled according to the row in the original data set. This picture can help you see how well the k-means clustering algorithm separates the data into two different groups.

Output

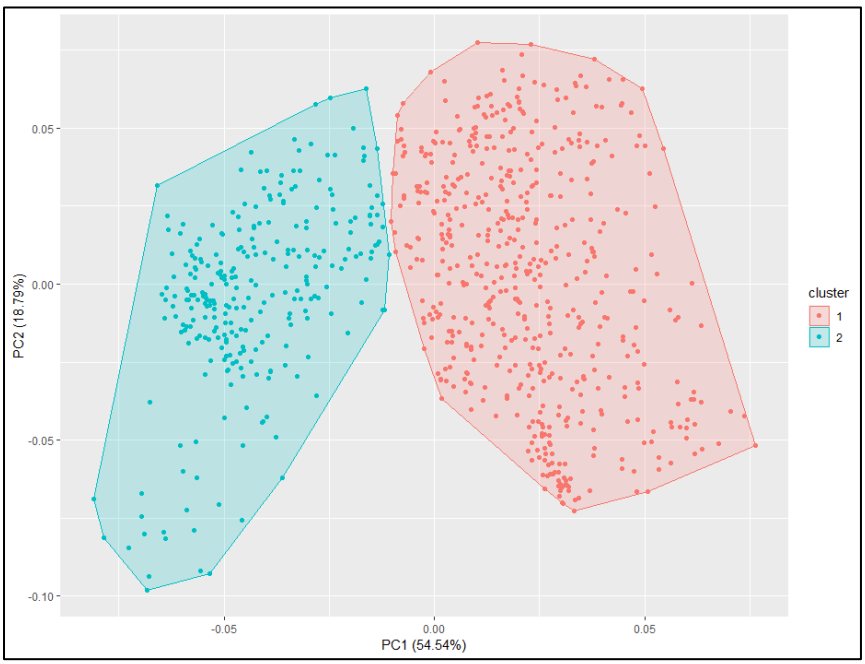


Figure 17:clutering_output(k=2)

Find the ratio of the number of squares to the number of square groups when k = 2.

```r
# When k=2, Extract relevant information.

# Cluster centers
cluster_centers <- k2$centers

# Clustered results
cluster_assignments <- k2$cluster

# Between-cluster sum of squares
BSSk2 <- k2$betweenss

# Within-cluster sum of squares
WSSk2 <- k2$tot.withinss

# Total sum of squares
TSSk2 <- BSSk2 + WSSk2

# BSS/TSS ratio
BSS_TSS_ratiok2 <- BSSk2 / TSSk2

# Percentage of variance explained
percent_vark2 <- round(BSS_TSS_ratiok2 * 100, 3)

# Output results
cat("Cluster centers:\n", cluster_centers, "\n\n")
cat("Cluster assignments:\n", cluster_assignments, "\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("Percentage of variance explained: ", percent_vark2, "%\n")
```

Figure 18:ratio_calculation(k=2)

Output

```
> percent_vark2 <- round(BSS_TSS_ratiok2 * 100, 3)
>
> # Output results
> cat("Cluster centers:\n", cluster_centers, "\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 -0.5309785 1.021847 -0.6175342 1.18842 -0.6400189 1.231691 -0.
5270382 1.014264 0.04804773 -0.09246595 -0.05799499 0.1116091 -0.1198347 0.2306172 -0.02909619 0.05599447 -0.1073133 0.2
065202

> cat("Cluster assignments:\n", cluster_assignments, "\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 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 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 1 1 2 2 2 1 1 2 1 1 1 1 1 2 2 1 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 1 1 1 1 2 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 1 2 2 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 2 1 1 1 2 1 2 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 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 1 2 1 1 1 1 2 1 2 1 1 1 2 1 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 1 2 2 1 2 2 1 2 1 1 1 1 2 2 1 1 1 2 1 1 1 2 1 1 2 2 2 1 1 1 2 1 1 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 1 2 1 1 1 1 2 1 1 1 1 2 1 2 1 2 2 1 1 2 1 1 1 2 2
2 1 1 2 2 1 2 2 2 1 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 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 1 2 1 1 1 1 2 2 2 1 1 2 2 1 2 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 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 2 1 2 1 1 1 2 1 2 1 1 1 2 1 2 1 2 1 1 1 2 1 1 1 1 1 2 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 1 1 1 2
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 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 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("Percentage of variance explained: ", percent_vark2, "%\n")
Percentage of variance explained:  42.692 %
>
```

Figure 19:ratio_calculation_output(k=2)

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

```
163
164 #3 clusters
165
166 k3 <-kmeans(scaled_Vehicles_dataset, 3)
167 k3
168 autoplot(k3,scaled_Vehicles_dataset,frame=TRUE)
169
170
171 # When k=3, Extract relevant information
172 # Cluster centers
173 cluster_centers <- k3$centers
```

Figure 20:clutering_code(k=3)

The "scaled_Vehicles_dataset" is being clustered using k-means with 3 clusters by the code. On the first line, the "kmeans()" function is used to perform k-means clustering and the result is stored in the "k3" variable. In the second line, we use the "autoplot()" function from the "ggplot2" package to output the result and generate a plot of the clustering outcomes. The generated scatter plot displays cluster midpoints and data points shaded according to their cluster membership.

Output

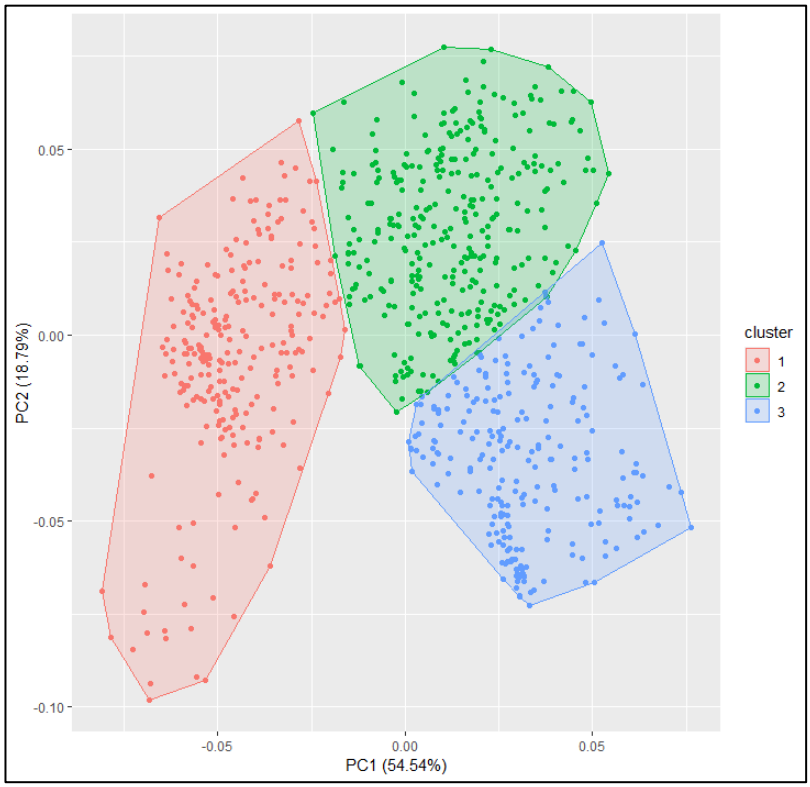


Figure 21:clutering_output(k=3)

Calculate the ratio between the total squares and the square clusters when k = 3

```
169
170
171 # When k=3, Extract relevant information
172 # Cluster centers
173 cluster_centers <- k3$centers
174
175 # Clustered results
176 cluster_assignments <- k3$cluster
177
178 # Between-cluster sum of squares
179 BSSk3 <- k3$betweenss
180
181 # Within-cluster sum of squares
182 WSSk3 <- k3$tot.withinss
183
184 # Total sum of squares
185 TSSk3 <- BSSk3 + WSSk3
186
187 # BSS/TSS ratio
188 BSS_TSS_ratiok3 <- BSSk3 / TSSk3
189
190 # Percentage of variance explained
191 percent_vark3 <- round(BSS_TSS_ratiok3 * 100, 3)
192
193 # Output results
194 cat("Cluster centers:\n", cluster_centers, "\n\n")
195 cat("Cluster assignments:\n", cluster_assignments, "\n\n")
196 cat("BSS/TSS ratio: ", round(BSS_TSS_ratiok3, 3), "\n\n")
197 cat("BSS: ", round(BSSk3, 3), "\n\n")
198 cat("WSS: ", round(WSSk3, 3), "\n\n")
199 cat("Percentage of variance explained: ", percent_vark3, "%\n")
200
201
202
```

Figure 22:ratio_calculation(k=3)

Output

```
> # Output results
> cat("Cluster centers:\n", cluster_centers, "\n\n")
Cluster centers:
 1.163264 -0.2285316 -0.9423859 1.189408 -0.5301921 -0.5474136 1.224256 -0.2929582 -0.9181953 1.053282 0.001398735 -1.14
5655 0.2198741 0.3660925 -0.7525338 0.7124052 -0.1699273 -0.5350742 1.312257 -0.4491392 -0.7945601 -1.224891 0.3136041
0.8899092 1.317075 -0.4769669 -0.7607372 1.110503 -0.4987139 -0.5059132 1.265981 -0.4002859 -0.8128739 1.324288 -0.45618
52 -0.7977349 1.096209 -0.5520307 -0.4155658 -0.0306539 -0.6831445 0.992033 0.1386373 -0.03494371 -0.1014964 0.2752784 -
0.02711354 -0.2608554 -0.02367658 0.7756077 -1.062805 0.1594321 0.6807868 -1.128556

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

> 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("Percentage of variance explained: ", percent_vark3, "%\n")
Percentage of variance explained:  55.217 %
>
```

Figure 23:ratio_calculation_output(k=3)

1.1.4. Code section for silhouette plot

When Fit k-means model with k=2

```
206
207 k <- 2
208
209 part01_kmeans_model_2 <- kmeans(scaled_Vehicles_dataset, centers = k, nstart = 25)
210
211 # Generate silhouette plot
212
213 silhouette_plot2 <- silhouette(part01_kmeans_model_2$cluster, dist(scaled_Vehicles_dataset))
214
215 # Determine the typical silhouette's width.
216
217 avg_silhouette_width <- mean(silhouette_plot2[, 3])
218
219 # Plot the silhouette plot
220
221 plot(silhouette_plot2, main = paste0("Silhouette Plot for k =", k),
222      xlab = "Silhouette Width", ylab = "Cluster", border = NA)
223
224 # As a vertical line, add the average silhouette width.
225
226 abline(v = avg_silhouette_width, lty = 2, lwd = 2, col = "red")
227
228
229
```

Figure 24:sihoutte_plot(k=2)

The code uses the "kmeans()" function to fit a k-means clustering algorithm to the scaled car dataset. The number of clusters, represented by the centres parameter, and the number of times the algorithm should be repeated with various initial centres, represented by the nstart parameter. The "part01_kmeans_model_2" object contains the final k-means model. The distances between the observations in the scaled vehicle dataset are utilised in conjunction with the cluster assignments in "part01_kmeans_model_2" to produce a silhouette plot via the "silhouette()" function. The average silhouette width is displayed as a red vertical line in the silhouette plot, generated with the" plot() "method. In addition to the main argument, which sets the plot's title, the "xlab" and "ylab" arguments, which set the plot's axis names, and the border argument, which makes the plot's borders transparent, are all required.

Output

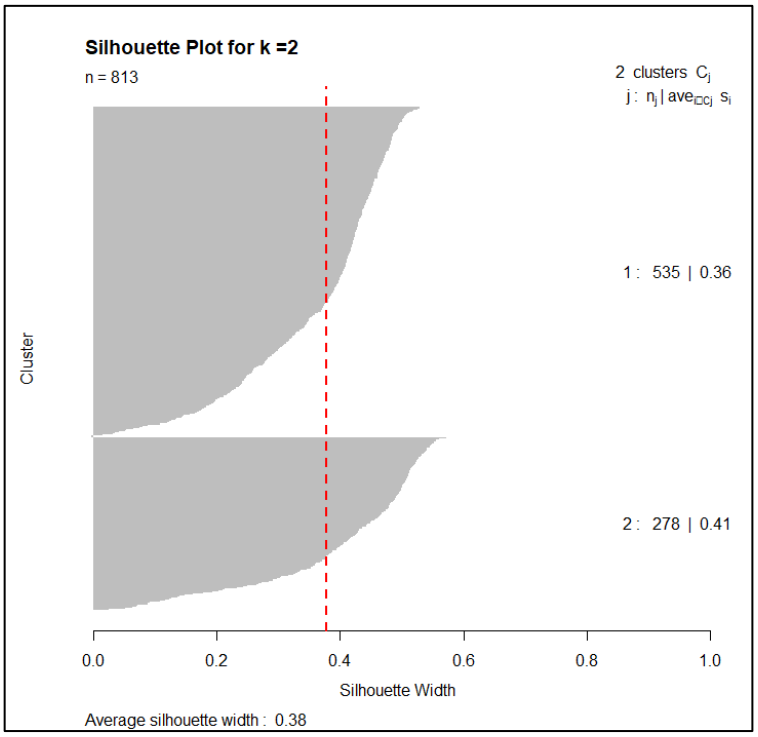


Figure 25:sihoutte_plot_output(k=2)

When Fit k-means model with k=3

```
231
232 # When Fit k-means model with k=3
233
234 k <- 3
235 part01_kmeans_model_3 <- kmeans(scaled_Vehicles_dataset, centers = k, nstart = 25)
236
237 # Generate silhouette plot
238
239 silhouette_plot3 <- silhouette(part01_kmeans_model_3$cluster, dist(scaled_Vehicles_dataset))
240
241 # Determine the typical silhouette's width.
242
243 avg_silhouette_width <- mean(silhouette_plot3[, 3])
244
245 # Plot the silhouette plot
246
247 plot(silhouette_plot3, main = paste0("Silhouette Plot for k =", k),
248      xlab = "Silhouette Width", ylab = "Cluster", border = NA)
249
250 # As a vertical line, add the average silhouette width.
251
252 abline(v = avg_silhouette_width, lty = 2, lwd = 2, col = "red")
253
254
255
```

Figure 26:Figure 23:sihoutte_plot(k=3)

For a k=3 k-means clustering model, the code to generate a Silhouette plot is shown. Using k = 3 as the number of clusters, a k-means model is fit to the scaled Vehicles dataset. The "silhouette()" function uses the model's cluster assignments and the data's distance matrix to produce the Silhouette plot. When applied to the third column of the "silhouette_plot3" object, the "mean()" function yields the average width of the Silhouette. A vertical line is added to the plot using the "abline()" function, and the plot is displayed using the "plot()" function with descriptive labels.The Silhouette plot provides a graphical representation for assessing the accuracy of the clustering results by contrasting the distance between a data point and its fellow members within the same cluster with that between the data point and its nearest neighbouring cluster members. Better clustering results can be expected from Silhouettes with wider average widths.

Output

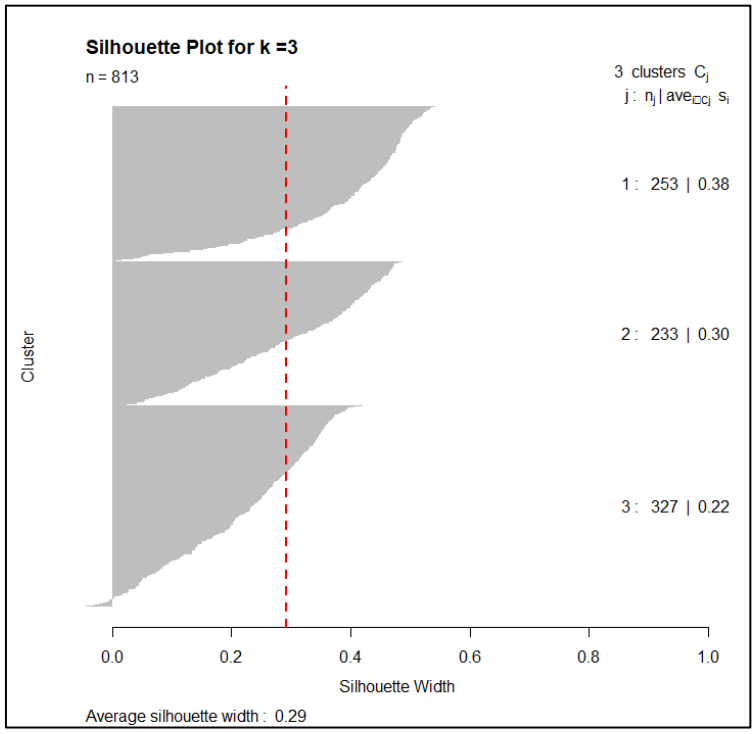


Figure 27:sihoutte_plot_output(k=3)

1.2. 2nd Subtask Objectives

1.2.1. What is pcs principle?

Principal Component Analysis is a statistical method for reducing the number of dimensions in a large dataset. Finding the linear combinations of the original variables that capture the most variation in the data—the principal components—is how it works. To minimize the dimensionality of a dataset while preserving as much information as possible, it is common practice to keep only the most salient main components. To determine the most influential aspects of a system, such as the elements that influence consumer behavior or the features of a dataset that best predict an outcome, PCA is frequently used in disciplines including data science, economics, and engineering.

1.2.2. PCs that provide at least cumulative score

Principal Component Analysis (PCA) is performed on the "scaled_Vehicles_dataset" using the belove code. The principal component analysis (PCA) is executed with the "prcomp()" function, and the eigenvectors and eigenvalues are displayed with the "summary()" function. The proportion of variance explained by each principle component is then calculated, and the results are displayed graphically in the form of a cumulative plot.Then, the "predict() "function is used to create a new dataset from the original one, this time with attributes representing the principal components. A new converted dataset is generated by selecting the PCs with a cumulative score of 92% or higher. Finally, the "boxplot()" function is used to create a graphical representation of the changed data's distribution, and the "View()" function is employed to examine the modified dataset in a new window.

```
261
262
263 # Conduct a PCA analysis.
264
265 pca <- prcomp(scaled_Vehicles_dataset)
266
267
268 # Print the eigenvectors and eigenvalues
269
270 print(summary(pca))
271
272 # Calculate the principal component (PC) cumulative score.
273
274 pca_var <- pca$sdev^2
275 pca_var_prop <- pca_var / sum(pca_var)
276 pca_var_cumprop <- cumsum(pca_var_prop)
277
278 # Cumulative plot score for each PC
279
280 plot(pca_var_cumprop, xlab = "Principal Component", ylab = "Explained Cumulative Proportion of Variance",
281      ylim = c(0, 1), type = "b")
282
283 # Make a new converted dataset with attributes representing the principal components.
284
285 pca_trans <- predict(pca, newdata = scaled_Vehicles_dataset)
286
287 # Select computers with a minimum cumulative score of over 92%.
288
289 selected_pcs <- which(pca_var_cumprop > 0.92)
290 transformed_data <- pca_trans[, selected_pcs]
291 boxplot(transformed_data)
292
293 View(transformed_data)
294
295
```

Figure 28:code for pca

Output

```
>
> 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 29:pca_out_1

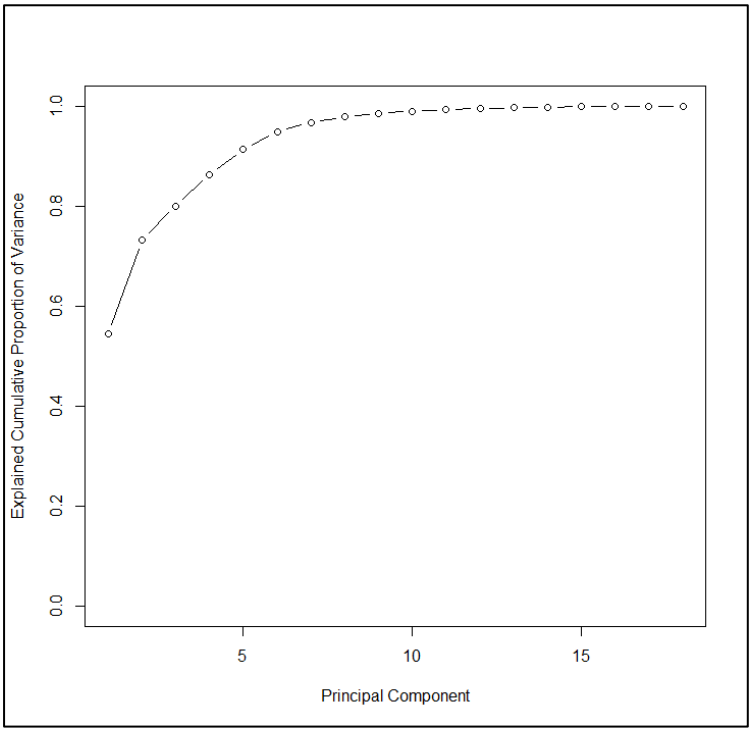


Figure 31:pca_output_2

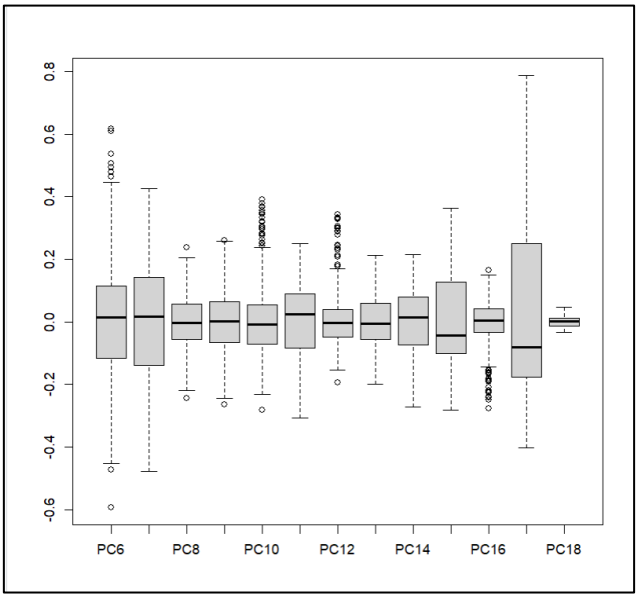


Figure 30:pca_output_3

1.2.3. Determine the number of cluster centers via four “automated tools.”

1.2.3.1. NBclust

The Euclidean distance and the gap is used in the following NBclust method

Code :

```
299
300 #Nbclust method
301
302 set.seed(1234)
303
304 NBcluster <- NbClust(transformed_data, min.nc = 2,max.nc = 10, method = "kmeans")
305
306 table(NBcluster$Best.n[1,])
307
```

Figure 32:Nbclust_code

Output :

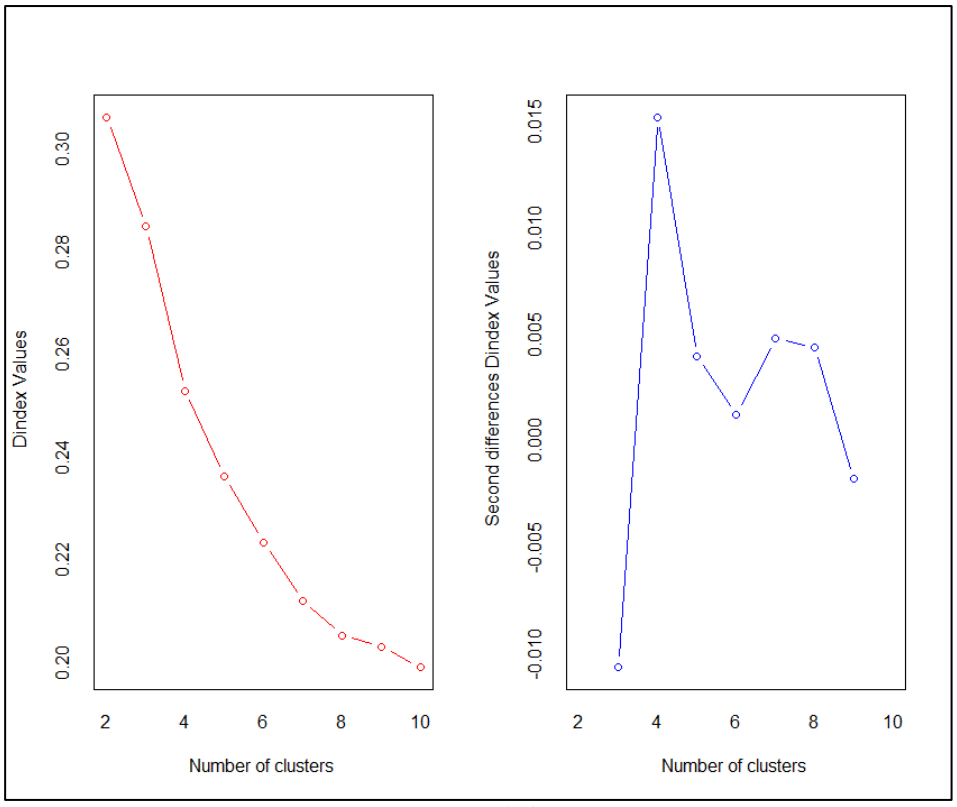


Figure 33:NbClust_output

1.2.3.2. Code section for elbow function

Code

```
307
308 #elbow method
309 fviz_nbclust(transformed_data,kmeans,method = "wss")
310
311 #silhouette method
```

Figure 34:elbow_code_pca

Output

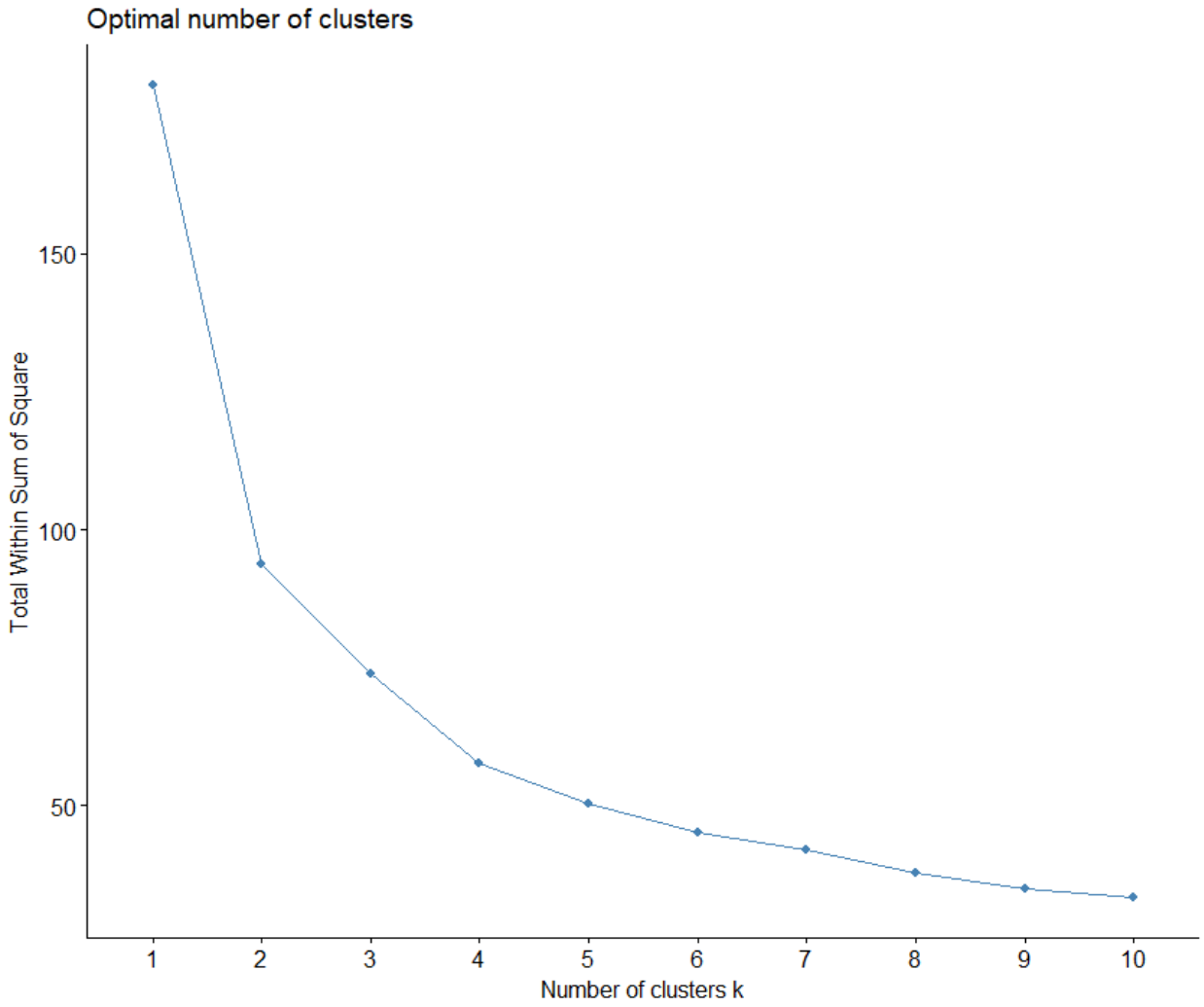


Figure 35:elbow_output_pca

1.2.3.4. Code section for Silhouette method

Code:

```
310
311 #silhouette method
312 fviz_nbclust(transformed_data,kmeans,method = "silhouette")
313
```

Figure 36:code_section_silhouette

Output:

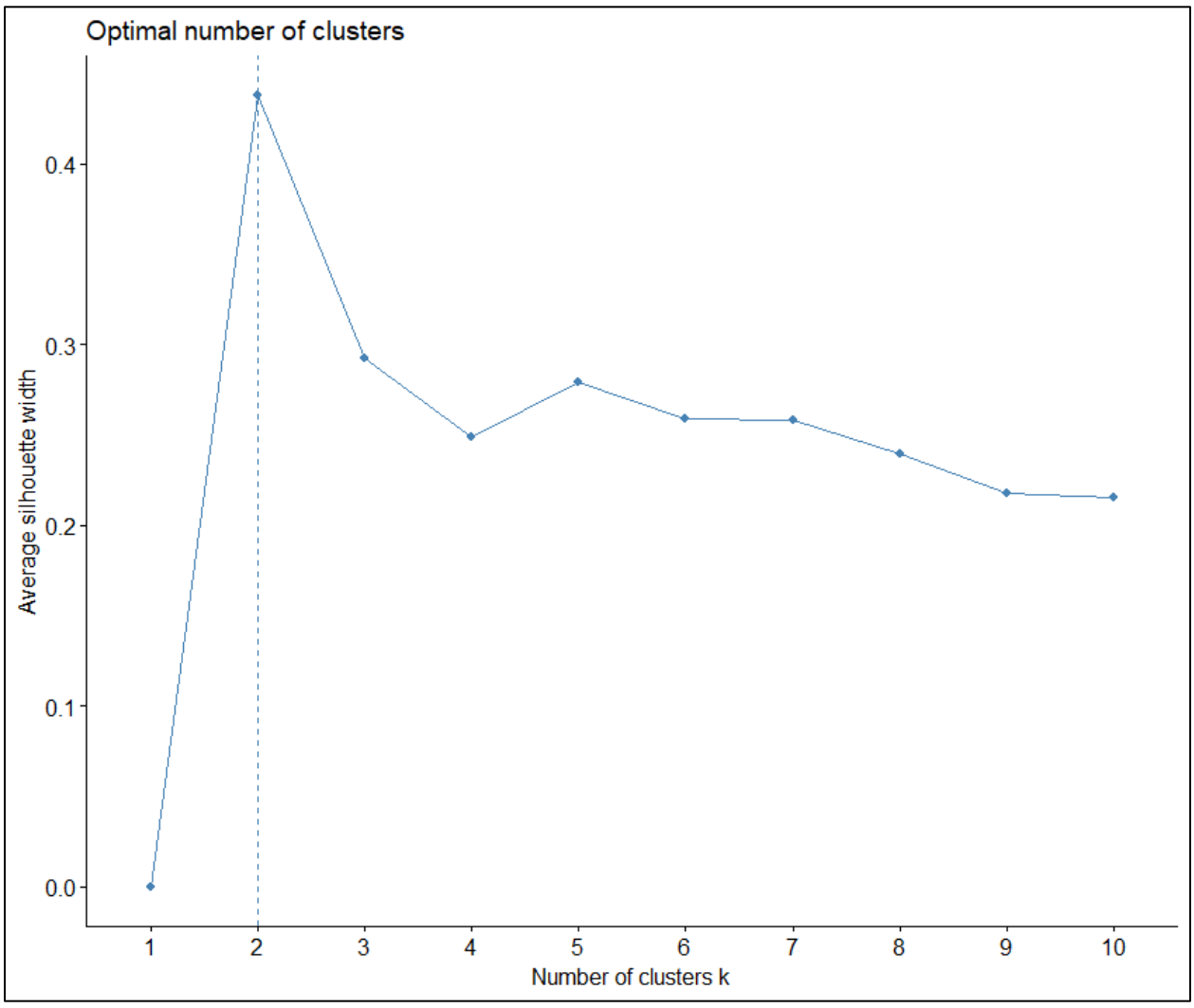


Figure 37:silhoutte_pca_output

1.2.3.4.5. Code section for Gap statistics method

Code:

```
312  fviz_nbclust(transformed_data,kmeans,method = "silhouette")
313
314  #gap static method
315  fviz_nbclust(transformed_data,kmeans,method = "gap_stat")
316
317
318
```

Figure 38:gap_code_pca

Output:

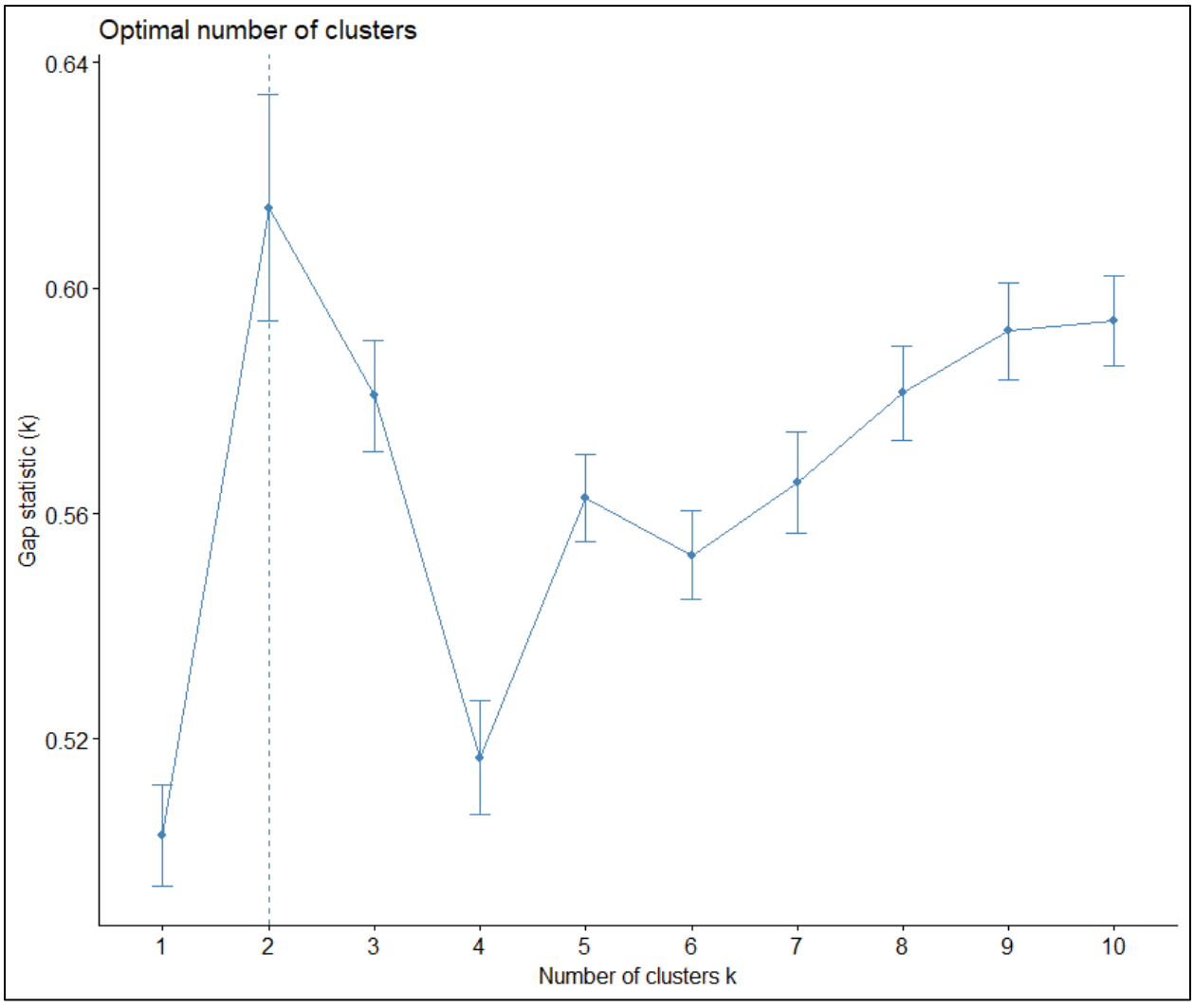


Figure 39:gap_output_pca

1.2.4. Evaluation metrics for k-means clustering.

K mean = 2

code

```
# Perform k-means clustering with k=2
set.seed(1234)

kmeans_model_2 <- kmeans(transformed_data, centers = 2, nstart = 25)

# Print the k-means output

print(kmeans_model_2)
autoplot(kmeans_model_2,transformed_data,frame=TRUE)


cat("For k=2:\n")
# Calculate the within-cluster sum of squares (WSS)
wss_2 <- sum(kmeans_model_2$withinss)
# Print the WSS
cat("Within-cluster sum of squares (WSS): ", wss_2, "\n")


# 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)
# Print the BSS
cat("Between-cluster sum of squares (BSS): ", bss_2, "\n")


# Calculate the total sum of squares (TSS)
tss_2 <- sum(dist(transformed_data)^2)
# Print the TSS
cat("Total sum of squares (TSS): ", tss_2, "\n")


# Calculate the ratio of BSS to TSS
bss_tss_ratio_2 <- bss_2 / tss_2
# Print the ratio of BSS to TSS
cat("Ratio of BSS to TSS: ", bss_tss_ratio_2, "\n\n")
```

Figure 40:code_ for_kmean_pca(k=2)

outputs

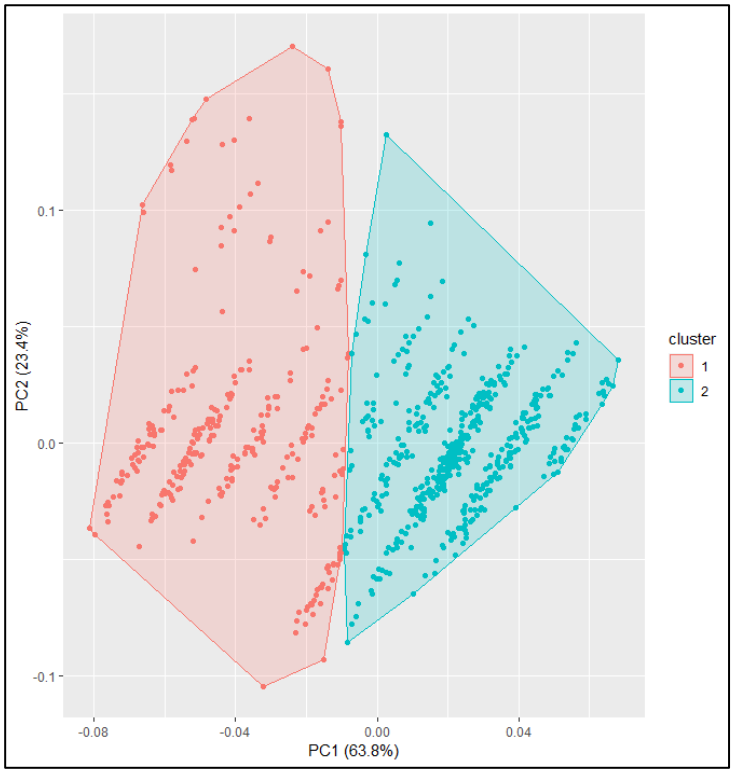


Figure 41:cluster_pca_output_1(k=2)

```
>
> print(kmeans_model_2)
K-means clustering with 2 clusters of sizes 298, 515

Cluster means:
          PC6        PC7         PC8         PC9         PC10        PC11        PC12        PC13
1 -0.08711086 -0.1557934  0.05933291 -0.09249059 -0.009458928 -0.11804152  0.02984440  0.06281173
2  0.05040589  0.0901484 -0.03433244  0.05351883  0.005473322  0.06830364 -0.01726919 -0.03634543
         PC14        PC15         PC16        PC17         PC18
1  0.08020637  0.15843351 -0.013179478  0.2996110 -0.016402714
2 -0.04641068 -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 2 1 2 2 1 1 2 2 2 2 1 2 2 2 1 2 1 2 1 2 1 2 2 2 1
 [50] 2 1 2 1 1 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
 [99] 1 1 2 2 2 2 2 2 2 2 1 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 2
[148] 2 2 1 2 2 1 1 2 1 2 2 1 1 2 1 1 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1 2 2 2 2 1 2 1 1 2 2
[197] 1 2 2 2 2 2 2 2 1 2 2 1 1 2 2 2 2 1 2 1 2 2 2 1 2 1 2 2 1 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 1 1
[246] 1 2 1 2 2 1 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 1 2 2 2 2 2 1
[295] 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 2 2 2 2 1 1 1 2 2 2 1 2 2 2 1 2
[344] 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2 1 2 2 1 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 1 2 1 2 2
[393] 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 2 2 1 2 2 2
[442] 1 2 2 2 2 2 2 1 1 2 2 1 1 1 2 1 1 1 2 2 1 1 2 2 2 1 2 2 1 1 2 2 1 1 2 2 1 1 2 1 2 1 1 1 2 2 1
[491] 1 2 2 2 1 2 2 2 2 2 2 1 2 1 1 2 2 2 1 1 2 2 1 1 2 1 1 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 1 1 1 2 2
[540] 2 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 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 2
[589] 2 2 2 2 2 2 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 1 2 1 2 2 2 1 1 2 1 2 1 2 2 2
[638] 2 1 1 1 1 2 1 1 1 2 2 1 2 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 2 2 2 1 2 1 1 1 2 2 2 2 2 1 1 2 2 1 1
[687] 2 1 2 2 1 1 2 1 1 1 2 2 1 2 2 1 1 1 2 1 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 1
[736] 1 2 2 2 1 2 2 1 1 2 2 2 1 1 1 2 1 2 1 1 2 2 1 2 1 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2
[785] 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 2 1 1 2 2

Within cluster sum of squares by cluster:
[1] 46.31438 47.33746
 (between_SS / total_SS =  48.1 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
> autoplot(kmeans_model_2,transformed_data,frame=TRUE)
>
>
> cat("For k=2:\n")
For k=2:
> # Calculate the within-cluster sum of squares (WSS)
> wss_2 <- sum(kmeans_model_2$withinss)
> # Print the WSS
> cat("Within-cluster sum of squares (WSS): ", wss_2, "\n")
Within-cluster sum of squares (WSS):  93.65185
>
>
> # Calculate the between-cluster sum of squares (BSS)
> bss_2 <- sum(kmeans_model_2$size * dist(rbind(kmeans_model_2$centers, colMeans(transformed_data)))^2)
Warning message:
In kmeans_model_2$size * dist(rbind(kmeans_model_2$centers, colMeans(transformed_data)))^2 :
  longer object length is not a multiple of shorter object length
> # Print the BSS
> cat("Between-cluster sum of squares (BSS): ", bss_2, "\n")
Between-cluster sum of squares (BSS):  250.4963
>
>
> # Calculate the total sum of squares (TSS)
> tss_2 <- sum(dist(transformed_data)^2)
> # Print the TSS
> cat("Total sum of squares (TSS): ", tss_2, "\n")
Total sum of squares (TSS):  146717.9
>
```

Figure 42:cluster_pca_output(k=2)

K mean =3

code

```r

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

# Print the k-means output

print(kmeans_model_3)
autoplot(kmeans_model_3,transformed_data,frame=TRUE)

cat("For k=3:\n")
# Calculate the within-cluster sum of squares (WSS)
wss_3 <- sum(kmeans_model_3$withinss)
# Print the WSS
cat("Within-cluster sum of squares (WSS): ", wss_3, "\n")


# Calculate the between-cluster sum of squares (BSS)
bss_3 <- sum(kmeans_model_3$size * dist(rbind(kmeans_model_3$centers, colMeans(transformed_data)))^2)
# Print the BSS
cat("Between-cluster sum of squares (BSS): ", bss_3, "\n")


# Calculate the total sum of squares (TSS)
tss_3 <- sum(dist(transformed_data)^2)
# Print the TSS
cat("Total sum of squares (TSS): ", tss_3, "\n")


# Calculate the ratio of BSS to TSS
bss_tss_ratio_3 <- bss_3 / tss_3
# Print the ratio of BSS to TSS
cat("Ratio of BSS to TSS: ", bss_tss_ratio_3, "\n")
```

Figure 43:code_ for_kmean_pca(k=3)

Outputs

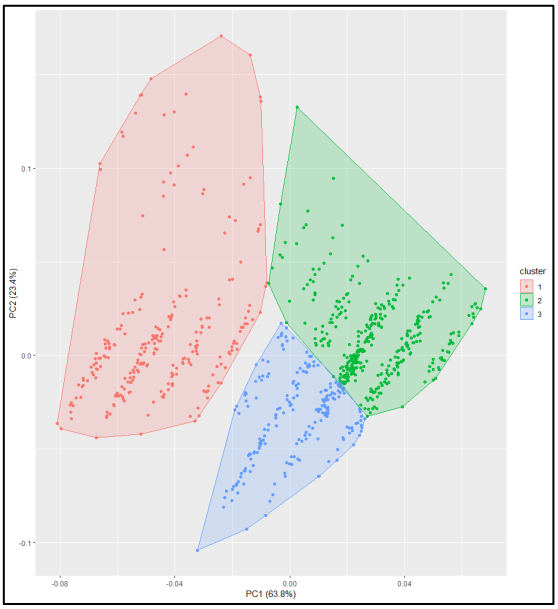


Figure 44:cluster_pca_output1(k=3)

```
> 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
1 -0.06429511 -0.14710694  0.066555191 -0.09409261  0.01462570 -0.12532617  0.0529470480  0.05975654  0.09426947
2  0.09886444  0.14905159 -0.041712044  0.07876478  0.03944040  0.09470938 -0.0007331176 -0.05867234 -0.05796432
3 -0.09694866 -0.08462067 -0.007459087 -0.02465531 -0.08807775 -0.01469887 -0.0636411798  0.03104107 -0.01255161
         PC15         PC16        PC17          PC18
1  0.18788103 -0.028382176  0.3639999 -0.0179636541
2 -0.10496631 -0.003847911 -0.1887212  0.0125562078
3 -0.04379083  0.041656443 -0.1108750 -0.0002947839

Clustering vector:
  [1] 3 3 1 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
 [60] 2 1 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
[119] 3 1 1 2 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
[178] 3 3 2 1 2 2 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 1 3 1 2 2 2 1 3 3 2 3 1 2 3 2 2 1 3 2 2 2 1 2
[237] 3 1 3 2 3 1 2 2 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
[296] 3 3 2 1 2 2 3 1 2 1 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
[355] 2 2 3 2 2 1 1 2 2 1 2 2 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
[414] 2 1 2 2 2 3 1 1 3 2 1 1 3 1 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
[473] 2 1 1 2 1 3 1 1 1 3 2 1 1 3 3 1 2 2 1 3 2 2 1 2 2 3 2 2 1 2 1 1 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
[532] 1 2 2 1 1 3 1 2 2 3 1 3 2 3 1 2 3 3 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
[591] 2 2 2 2 2 1 1 3 3 2 1 2 2 3 2 2 1 2 1 2 2 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
[650] 3 2 2 3 3 2 3 1 2 2 2 1 2 1 3 2 1 3 3 2 2 2 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
[709] 1 1 1 3 1 2 2 1 3 2 3 2 3 1 2 2 2 2 2 2 1 2 2 3 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
[768] 1 2 3 2 2 2 1 2 2 1 2 2 1 2 2 2 3 2 3 1 1 1 2 1 3 1 1 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 %)

Available components:

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

Figure 45:cluster_pca_output2(k=3)

1.2.5. Provide the silhouette plot which displays how close each point.

Code And Output:

```
k <- 2
kmeans_model_2 <- kmeans(transformed_data, centers = k, nstart = 25)

# Generate silhouette plot

silhouette_plot <- silhouette(kmeans_model_2$cluster, dist(transformed_data))

# Determine the typical silhouette's width.

avg_silhouette_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_silhouette_width, lty = 2, lwd = 2, col = "red")




# Construct a k-means model with k=3

k <- 3
kmeans_model_3 <- kmeans(transformed_data, centers = k, nstart = 25)

# Generate silhouette plot

silhouette_plot <- silhouette(kmeans_model_3$cluster, dist(transformed_data))

# Determine the typical silhouette's width.

avg_silhouette_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_silhouette_width, lty = 2, lwd = 2, col = "red")
```

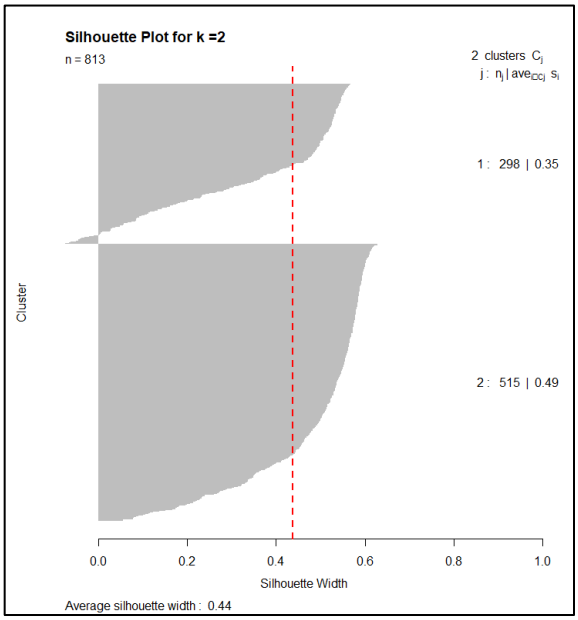


Figure 46:silhoutte_plot_pca(k=2)

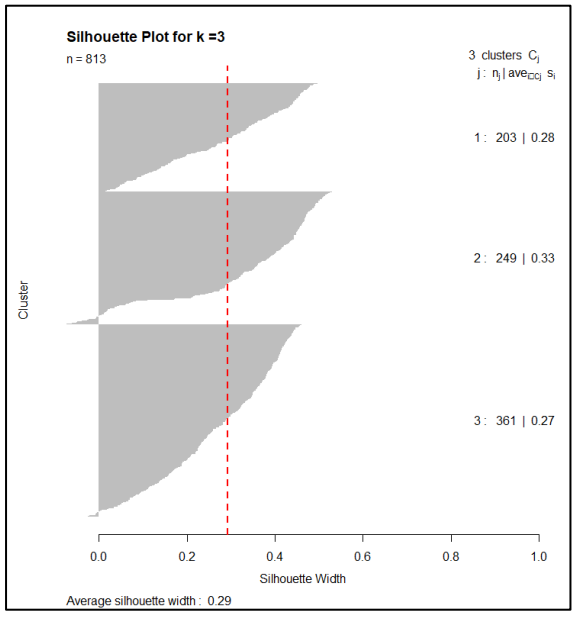


Figure 47:silhouette_code_pca(k=3)

Figure 48:silhoutte_plot_pca(k=3)

1.2.6. Implement and show the Calinski-Harabasz index

In cluster analysis, the validity of the clusters is evaluated using the Calinski-Harabasz index. It is determined by dividing the total squared distances between cluster centers by the total squared distances within each cluster times the number of clusters minus one to get the ratio of the between-cluster dispersion mean to the within-cluster dispersion mean. How well the data points are grouped and how different the clusters are from one another are also factors that the index considers. Better clustering are represented by higher values of the Calinski-Harabasz index.

$$CH = \frac{\frac{BGSS}{K-1}}{\frac{WGSS}{N-K}} = \frac{BGSS}{WGSS} \times \frac{N-K}{K-1}$$

Figure 49: the Calinski-Harabasz _formular

Code for k=2:

```
456
457
458 # Define a function to calculate the Calinski-Harabasz index for a given K-means clustering result and data
459
460 calinski_harabasz_pca2 <- function(cluster_result, data) {
461
462   # Extract the number of clusters from the clustering result
463
464   k2 <- length(unique(cluster_result$cluster))
465
466   # Extract the number of rows in the data
467   n2 <- nrow(data)
468
469   # Extract the between-cluster sum of squares (BSS) and within-cluster sum of squares (WSS) from the clustering result
470
471   BSS2 <- cluster_result$betweenss
472   WSS2 <- cluster_result$tot.withinss
473
474   # Calculate the Calinski-Harabasz index
475
476   ch_index2 <- ((n2 - k2) / (k2 - 1)) * (BSS2 / WSS2)
477
478   # Return the Calinski-Harabasz index
479
480   return(ch_index2)
481 }
482
483 # Call the calinski_harabasz_pca function to calculate the index for a specific K-means clustering result and data
484
485 ch_index_pca_2 <- calinski_harabasz_pca2(kmeans_model_2, transformed_data)
486
487 # Print the calculated index to the console for interpretation
488
489 cat("The Calinski-Harabasz index for the K-means (k=2) clustering result is:", ch_index_pca_2, "\n")
490
491
492
493
```

Figure 50:the Calinski-Harabasz code(K=2)

The "calinski_harabasz_pca2" function is defined in this code, and it accepts a K-means clustering result (cluster_result) and the data that was used to perform the clustering (data). Adjusted for the number of clusters and the total number of data points, the function computes the Calinski-Harabasz index, which is the ratio of the sum of squares between clusters to the sum of squares within clusters. First, it takes the clustering result and the data and extracts the number of clusters (k2) and the number of rows in the data (n2). Next, it takes the clustering result and returns the between- and within-cluster sums of squares (BSS2 and WSS2, respectively). The Calinski-Harabasz index for the 2-cluster K-means clustering result is then computed by passing the clustering result (kmeans_model_2) and the modified data (transformed_data) as inputs to the function. The console will display the computed index for analysis.

Output:

```
>
> # Print the calculated index to the console for interpretation
>
> cat("The Calinski-Harabasz index for the K-means (k=2) clustering result is:", ch_index_pca_2, "\n")
The Calinski-Harabasz index for the K-means (k=2) clustering result is: 751.7775
>
```

Figure 51:Calinski-Harabasz code_output(k=2)

Code for k=3:

code:

```
# Define a function to calculate the Calinski-Harabasz index for a given K-means clustering result and data
calinski_harabasz_pca3 <- function(cluster_result, data) {
  # Extract the number of clusters from the clustering result
  k3 <- length(unique(cluster_result$cluster))
  # Extract the number of rows in the data
  n3 <- nrow(data)
  # Extract the between-cluster sum of squares (BSS) and within-cluster sum of squares (WSS) from the clustering result
  BSS3 <- cluster_result$betweenss
  WSS3 <- cluster_result$tot.withinss
  # Calculate the Calinski-Harabasz index
  ch_index3 <- ((n3 - k3) / (k3 - 1)) * (BSS3 / WSS3)
  # Return the Calinski-Harabasz index
  return(ch_index3)
}
# Call the calinski_harabasz_pca function to calculate the index for kmeans_model_3 and transformed_data
ch_index_pca_3 <- calinski_harabasz_pca3(kmeans_model_3, transformed_data)
# Print the calculated index to the console for interpretation
cat("The Calinski-Harabasz index for the K-means (k=3) clustering result is:", ch_index_pca_3, "\n")
```

Figure 52:Calinski-Harabasz code(k=3)

Output:

```
> # Call the calinski_harabasz_pca function to calculate the index for kmeans_model_3 and transformed_data
>
> ch_index_pca_3 <- calinski_harabasz_pca3(kmeans_model_3, transformed_data)
>
> # Print the calculated index to the console for interpretation
>
> cat("The Calinski-Harabasz index for the K-means (k=3) clustering result is:", ch_index_pca_3, "\n")
The Calinski-Harabasz index for the K-means (k=3) clustering result is: 583.4431
>
```

Figure 53:Calinski-Harabasz code_output(k=3)

2. Multi-layer Neural Network

2.1. 1st Subtask Objectives

2.1.1. Part A

2.1.1.1. Electricity load forecasting

The input vector is a crucial feature of MLP models developed for energy load forecasting. It is made up of many factors that are utilized to foretell power usage in the next time period. Input vectors for MLP models can be defined in several different ways, each with its own implementation and design approach. The following are examples of several common methods now in use.

Autoregressive (AR) approach: Time series analysis, including power demand forecasting, frequently employs the autoregressive (AR) method as a modelling tool. Predicting future power use requires utilizing historical data on current use as inputs. The model predicts that a variable's future value will be a linear mixture of its past values at specified delays. The number of time-lags between the data employed in the AR model is defined by its order, indicated by p.

Moving Average (MA) approach: In the MA method of electrical load forecasting, prior measurements of electricity use are averaged and then used as inputs to make predictions about future use. The model assumes that the present value of the variable is in some way related to the weighted average of its error components at various time delays in the past. The number of lagged error terms that are utilized as inputs is designated by the MA model's order, which is indicated by q.

Fourier transform approach: Data on electrical consumption may be analyzed for periodic patterns by first converting it from the time domain to the frequency domain.

Weather data approach : Electricity usage is correlated with meteorological variables including temperature, humidity, and wind speed***.***

As part of our coursework, This MLP model was trained using a dataset that included columns for things like "data," "electricity consumption," "19th hour," and "20th hour." Next day (future) 20th hour energy usage must be predicted using historical data. The 20th hour's energy usage can be predicted in several different ways.

- ✓ According to power use during the 20th hour.
- ✓ According to power use during the 19th hour.
- ✓ According to power use during the 18th hour.

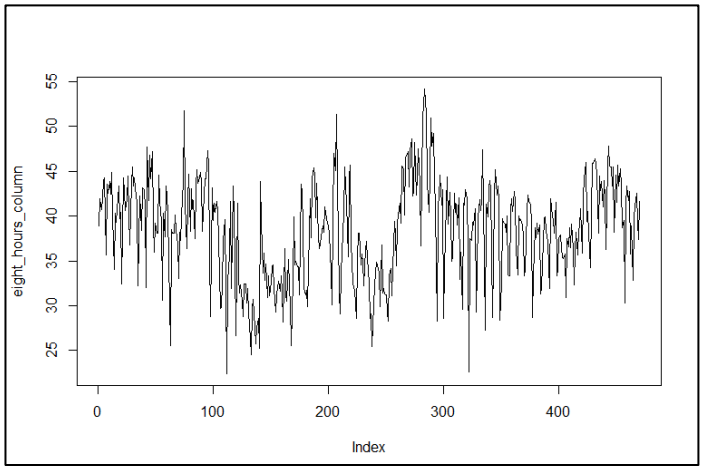
We may extrapolate previous 20-hour usages, such as tomorrow's energy use, from today's and yesterday's combined consumption. The utilization of two data sets has resulted in a more precise forecast. Similarly, we can extrapolate tomorrow's energy use from data as far back as a week ago. The consumption of the following hour, the 20, may be estimated from the consumption of the previous hour, the 19. In addition to being able to foretell the 20th hour, the 18th hour has a strong correlation with it. More accurate predictions of energy use in the 20th hour may be made by combining data from the 18th and 19th hours of use. These parameters can be used to estimate energy use in the next 20 hours.

2.1.1.2. Input vectors & I/O matrix

```
# create a  matrix
time_delayed_matrix <- bind_cols(t7 = lag(eight_hours_column,8),
                                 t4 = lag(eight_hours_column,5),
                                 t3 = lag(eight_hours_column,4),
                                 t2 = lag(eight_hours_column,3),
                                 t1 = lag(eight_hours_column,2),
                                 eight_hoursHour = eight_hours_column)

delayed_matrix <- na.omit(time_delayed_matrix)
```

Figure 54:crate_matrix

This function takes a data column with the name "eight_hours_column" and generates a matrix with a delay of eight hours. Create a matrix where each column is a time-delayed version of "eight_hours_column" using the "bind_cols()" function in the "dplyr" package. A new matrix called "time_delayed_matrix" is created as a consequence. However, rows with missing values are omitted using the "na.omit()" method, and the resultant matrix is stored in the delayed_matrix variable. The following code may be used to convert a single column of data into a time series matrix for use in machine learning model training. It contains data that may be used to make predictions about the future value of the target variable (eight_hoursHour) and other related variables.



2.1.2. Normalization

2.1.2.1. What is normalization and why is it important?

The process of normalization, a type of data preparation, involves the rescaling of numbers to fit a predetermined range. By minimizing the influence of feature size, outliers, and training convergence, it helps to boost machine learning model performance. It is widely employed in a wide variety of machine learning methods, including k-nearest neighbor's, support vector machines, and neural networks.

2.1.2.2. Using Normalization on a Model

```
48
49  # normalization function
50  normalization <- function(x){
51    return ((x - min(x)) / (max(x) - min(x)))
52  }
53
```

Figure 55:normalization_code

Using the min-max normalization technique, this code provides a function that accepts a vector of numbers as input and returns the normalized values. By removing the least value of the input vector from each element and dividing by the difference between the maximum and minimum values, this technique scales the input values to a range between 0 and 1. This ensures that smaller features have an equal impact on the output when using machine learning techniques like k-nearest neighbor's and support vector networks, which rely on distance measures.

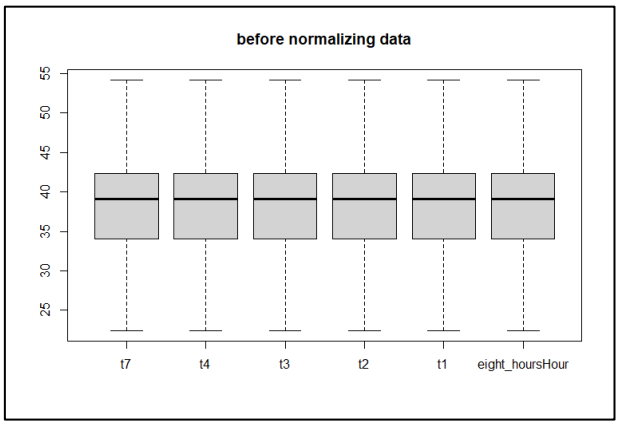


Figure 57:before_normalization

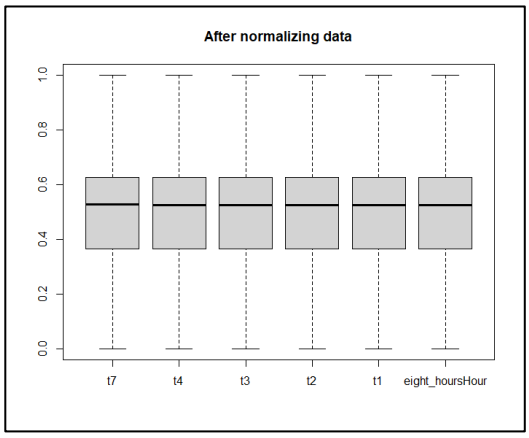


Figure 56:after_normalization

Values between 25 and 55 may be seen in both graphs before normalization, but only 0 and 1 remain in the final graph.

2.1.3. Training Neural Networks

2.1.3.1.1. Splitting data for training and testing

```
61
62  # Separating the normalised data into test and train sets
63  train_datasetNormaliz <- time_delayedNormaliz[1:380,]
64  test_datasetNormaliz <- time_delayedNormaliz[381:nrow(delayed_matrix),]
65
```

Figure 58:splitting_code

This code divides the standardized data into a training set and a test set. The first line copies the first 380 rows from the "time_delayedNormaliz" data frame" into a new data frame called "train_datasetNormaliz." The second line extracts rows 381–final from the "time_delayedNormaliz" data frame and stores them in a new data frame called "test_datasetNormaliz". By separating the data into training and testing sets, we can evaluate the accuracy and generalizability of the models by training them on a part of the data and then testing their performance with the remaining data.

2.1.3.1.2. testing data for each time delay

```
70  # Producing Test Information for Each and EveryDelay
71  t1_testDataSet <- as.data.frame(test_datasetNormaliz[, c("t1")])
72  t2_testDataSet <- test_datasetNormaliz[, c("t1", "t2")]
73  t3_testDataSet <- test_datasetNormaliz[, c("t1", "t2", "t3")]
74  t4_testDataSet <- test_datasetNormaliz[, c("t1", "t2", "t3", "t4")]
75  t7_testDataSet <- test_datasetNormaliz[, c("t1", "t2", "t3", "t4", "t7")]
76
```

Figure 59:testingdata

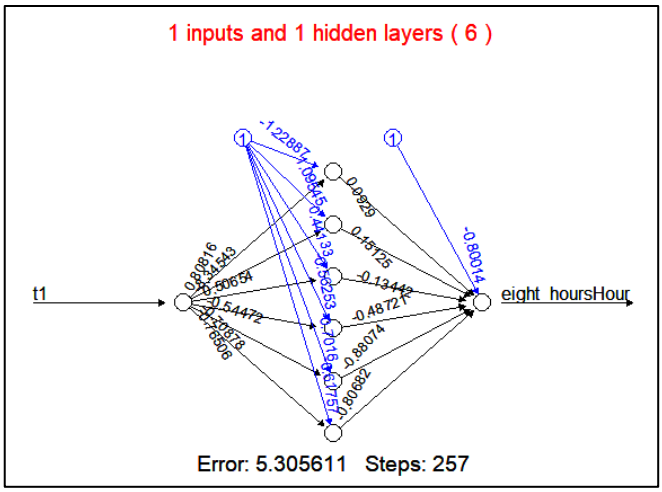
Only the required columns from the test data section will be chosen for model testing.

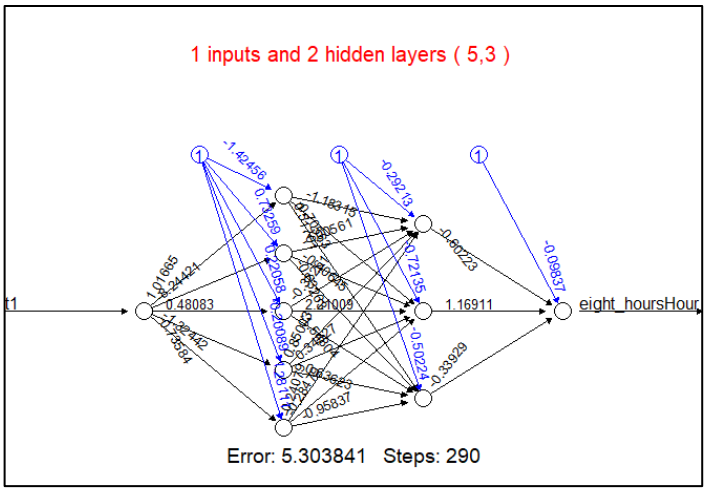
2.1.3.2. Training the model

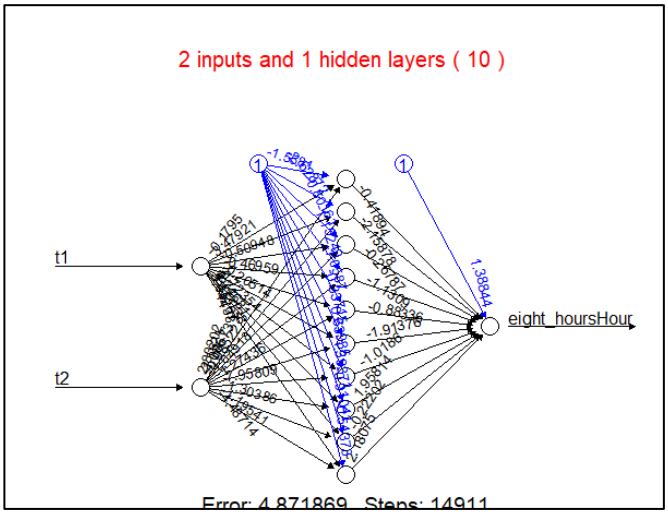
```r
#Function to train an AR model

ModelTrain <- function(formula, hiddenval, isLinear, actFunc,inputs,hidden){

  # Defining a title for the plot
  my_title <- paste(inputs,"inputs and",length(hidden),"hidden layers","(",paste(hidden, collapse=","),") \n")

  # Establishing a reproducible seed
  set.seed(1234)

  # The neuralnet method is used to train a model of a neural network.
  nural <- neuralnet(formula, data = train_datasetNormaliz, hidden = hiddenval, act.fct = actFunc, linear.output = isLinear)

  # Using a neural network to generate a plot
  plot(nural)

  # Keeping the plot in a variable
  plot_panel <- grid.grab(wrap = TRUE)

  # Creating a title grob
  plot_title <- textGrob(my_title,
                         x = .5, y = .20,
                         gp = gpar(lineheight_hours = 2,
                                   fontsize = 15, col = "red",
                                   adj = c(1, 0)
                         )
  )

  # Stacking the title and main panel, and plotting
  grid.arrange(
    grobs = list(plot_title,
                 plot_panel),
    height_hourss = unit(c(.15, .85), units = "npc"),
    width = unit(1, "npc")
  )
  dev.new()
  dev.off()
  return(nural)
}
```

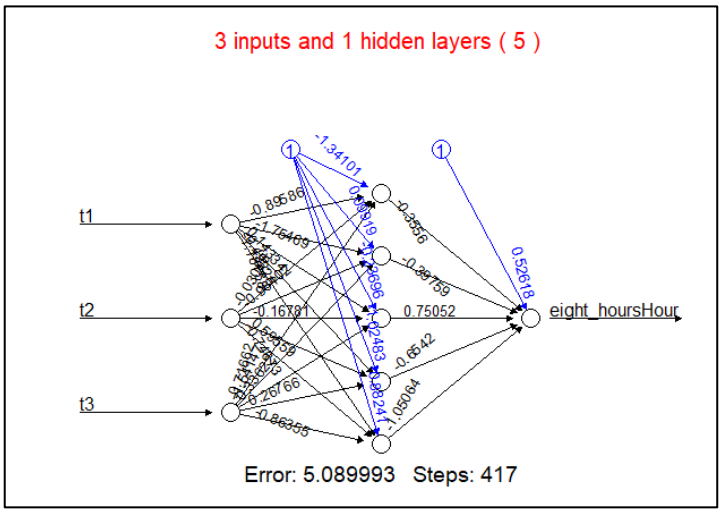
Figure 60:model_code

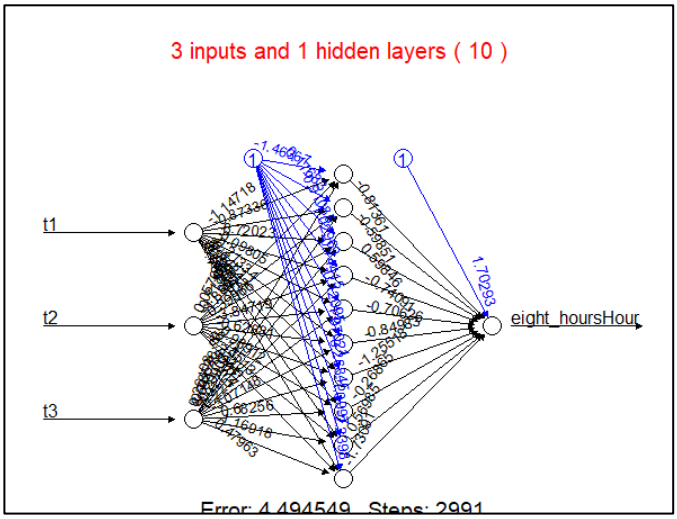
This code specifies a function named "ModelTrain" that accepts numerous parameters, including the inputs and hidden layers of the neural network, as well as the activation function, the number of hidden layers, and whether the output is linear. The "neuralnet" technique is used to train the network, a plot is generated, a title grob is made, and the plot of the title and major panel grobs is stacked and plotted using the "grid.arrange" function inside the function. At last, the function gives back the object that represents the trained neural network.

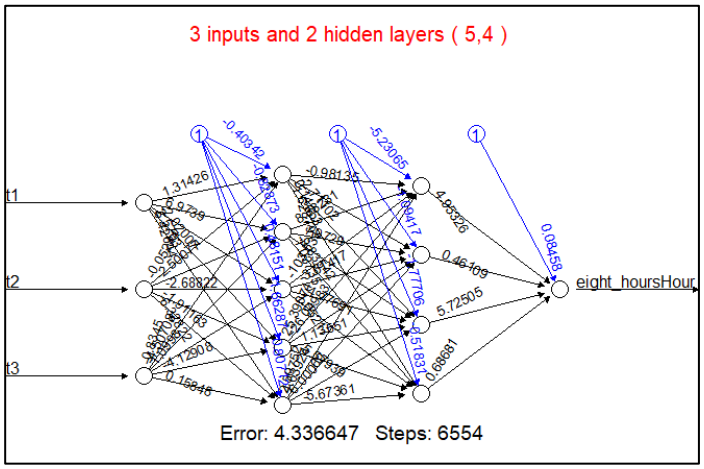


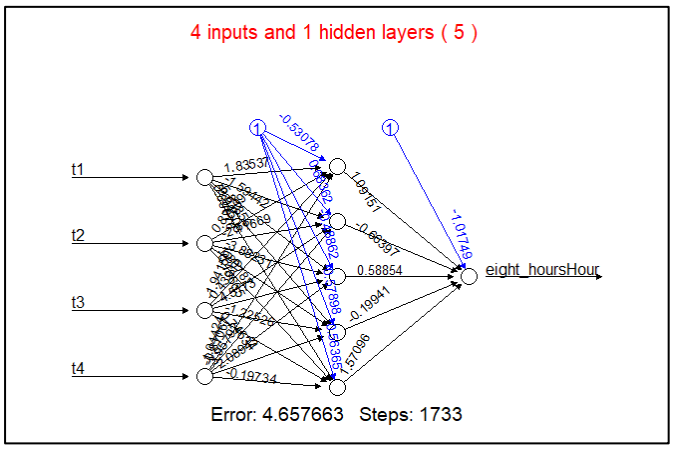


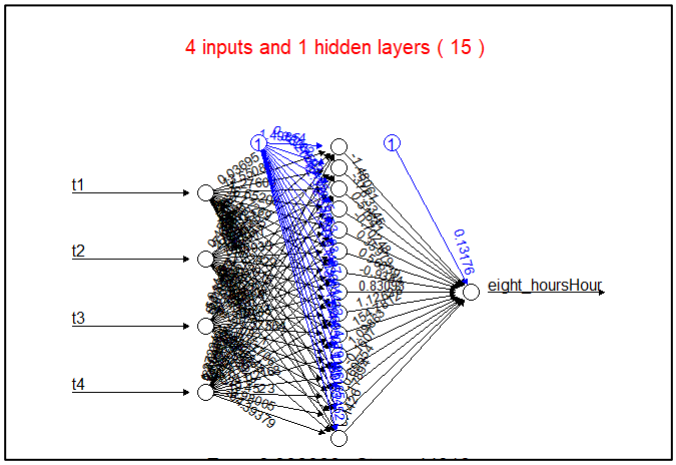

2 inputs and 1 hidden layers (10)
t1
t2
eight_hoursHour
Error: 4.871869 Steps: 14911

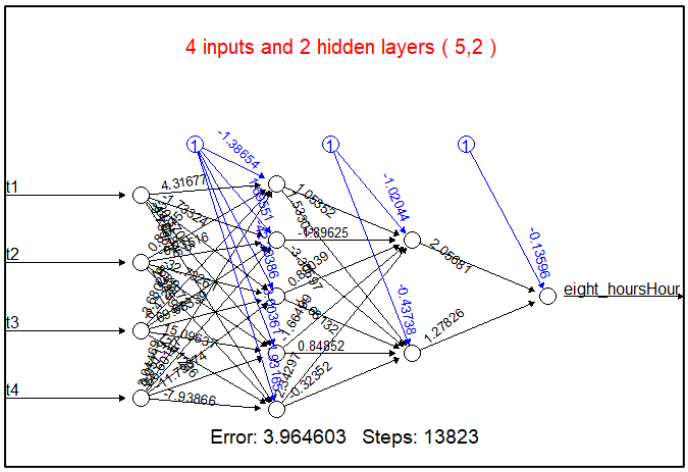

3 inputs and 1 hidden layers (5)
t1
t2
t3
eight_hoursHour
Error: 5.089993 Steps: 417

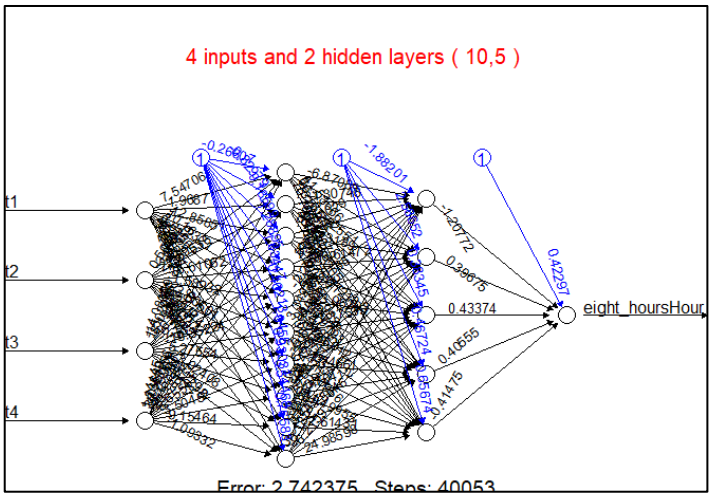

3 inputs and 1 hidden layers (10)
t1
t2
t3
eight_hoursHour
Error: 4.494549 Steps: 2991

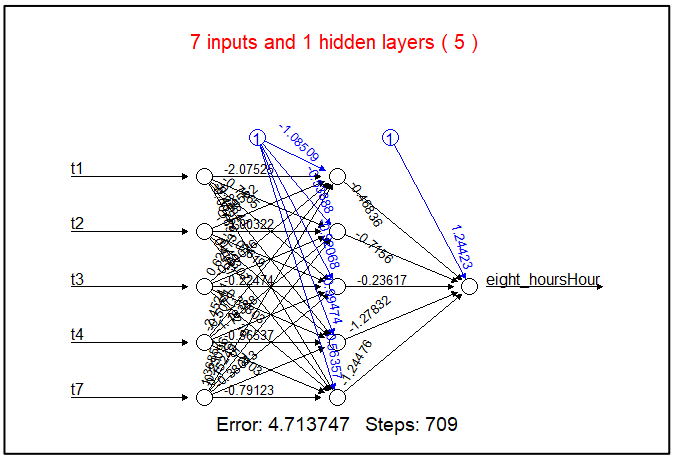

3 inputs and 2 hidden layers (5,4)
t1
t2
t3
eight_hoursHour
Error: 4.336647 Steps: 6554

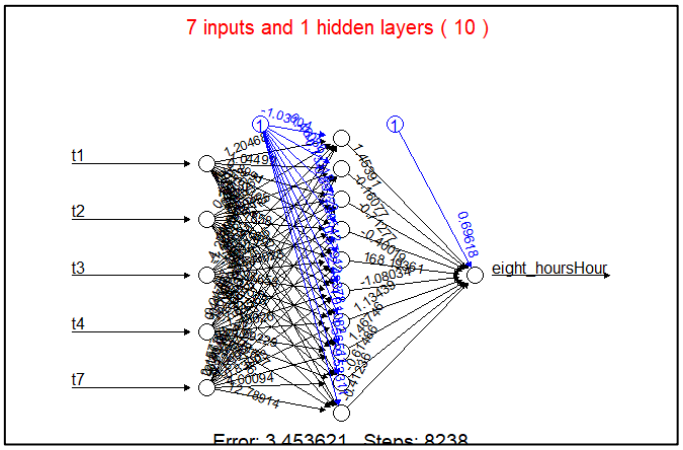

4 inputs and 1 hidden layers (5)
t1
t2
t3
t4
eight_hoursHour
Error: 4.657663 Steps: 1733

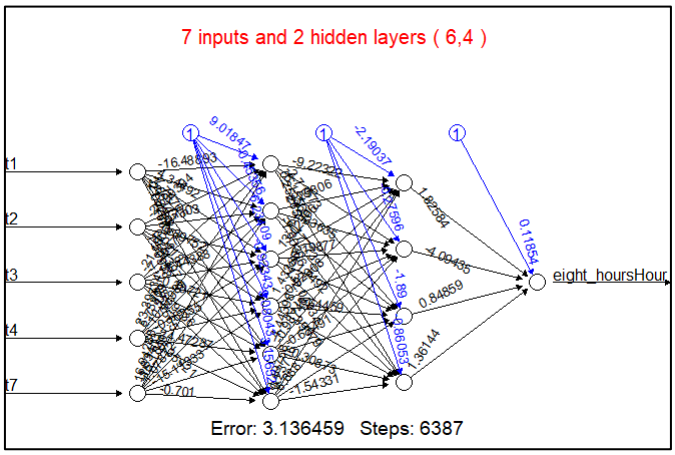

4 inputs and 1 hidden layers (15)
t1
t2
t3
t4
eight_hoursHour

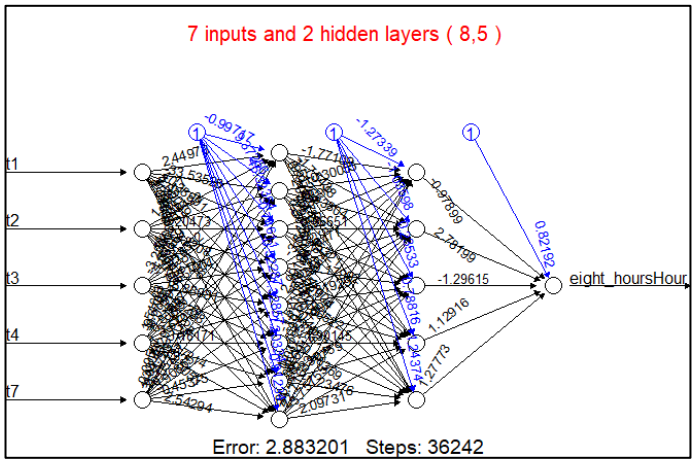

4 inputs and 2 hidden layers (5,2)
t1
t2
t3
t4
eight_hoursHour
Error: 3.964603 Steps: 13823

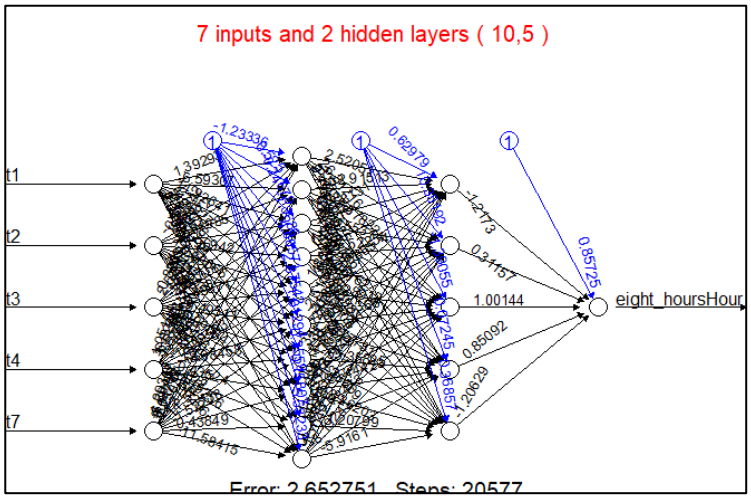

4 inputs and 2 hidden layers (10,5)
t1
t2
t3
t4
eight_hoursHour


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


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


7 inputs and 2 hidden layers (6,4)
t1
t2
t3
t4
t7
eight_hoursHour
Error: 3.136459 Steps: 6387


7 inputs and 2 hidden layers (8,5)
t1
t2
t3
t4
t7
eight_hoursHour
Error: 2.883201 Steps: 36242


7 inputs and 2 hidden layers (10,5)
t1
t2
t3
t4
t7
eight_hoursHour

2.1.4. RMSE, MAE, MAPE, and sMAPE

In regression and forecasting tasks, the RMSE, MAE, MAPE, and sMAPE metrics are frequently employed to evaluate model accuracy and performance. Absolute differences, relative mistakes, and symmetric adjustments are all taken into account by the various metrics, providing a comprehensive picture of the accuracy of the forecasts.

- RMSE (Root Mean Square Error)

In regression analysis, the statistic known as the Root Mean Squared Error (RMSE) is used to measure the degree to which observed values deviate from those predicted. It is calculated by taking the mean squared deviation from the predicted and observed values. The root-mean-squared error (RMSE) is a common metric used to compare the precision of various machine learning models. A smaller RMSE number indicates that the model is more accurate.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

- MAE (Mean Absolute Error)

Mean Absolute Error (MAE) is a statistic used in regression analysis to quantify the degree to which estimates deviate from reality. It is easier to read since it uses the same units as the target variable, and it is produced by averaging the absolute differences between the anticipated and actual values.

$$\mathbf{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\boldsymbol{x}_i - \boldsymbol{x}|$$

- MAPE (Mean Absolute Percentage Error)

Mean Absolute Percentage Error (MAPE) is a statistic used in regression analysis to quantify the percentage error between the predicted and actual values. It is used to assess the accuracy of different machine learning models but has several drawbacks, such as being undefined when the real value is 0.

$$MAPE = \frac{1}{n} . \sum_{i=1}^{n} \frac{|(A_t - Y_t)|}{A_t} . 100$$

- sMAPE (Symmetric Mean Absolute Percentage Error)

For the purpose of regression analysis, the standardised mean absolute percentage error (sMAPE) is a useful indicator. To determine this, we use the mean of the percentages by which predictions deviate from observations, with the total of predictions and observations serving as the denominator. If the sMAPE value is low, the accuracy is high.

$$\text{SMAPE} = \frac{1}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(A_t + F_t)/2}$$

2.1.5. Comparison table

Input count	*Hidden layers*	*neurons*	*Activation function*	*Linear*	*Accuracy*	*RMSE*	*MAE*	*MAPE*	*sMAPE*
1	1	6	tanh	FALSE	98.08 %	3.96811	3.281691	0.08219015	0.08335449
	2	5, 3	tanh	FALSE	98.03 %	3.951561	3.265439	0.08172144	0.08290732
2	1	10	logistic	TRUE	97.91 %	3.681026	3.015663	0.07545289	0.07640198
3	1	5	logistic	TRUE	98.93 %	3.857852	3.159695	0.07992375	0.07993308
	1	10	logistic	TRUE	97.61 %	3.750374	2.930764	0.0732769	0.07432807
	2	5,4	logistic	TRUE	97.71 %	3.878686	3.059654	0.07659256	0.07758889
4	1	5	logistic	TRUE	97.99 %	3.853526	3.160198	0.07934702	0.08003193
	1	15	logistic	TRUE	98.25 %	4.409359	3.43928	0.08703834	0.08785926
	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	6,4	logistic	TRUE	97.68 %	4.831348	3.670521	0.09346433	0.09523758
	2	8,5	logistic	TRUE	97.89 %	4.813866	3.748068	0.09419091	0.09571004
	2	10,5	logistic	TRUE	97.99 %	5.005324	4.085764	0.1030105	0.1048161

Table 1:comparison table 01

2.1.6. Best one-hidden layer & two-hidden layer

2.1.6.1. Best one hidden layer

The following comparison table suggests that a neural network with seven inputs and five hidden neurons is the optimal model for a single hidden layer. This model was chosen because, according to the comparison table, it achieved the highest accuracy (99.35%).

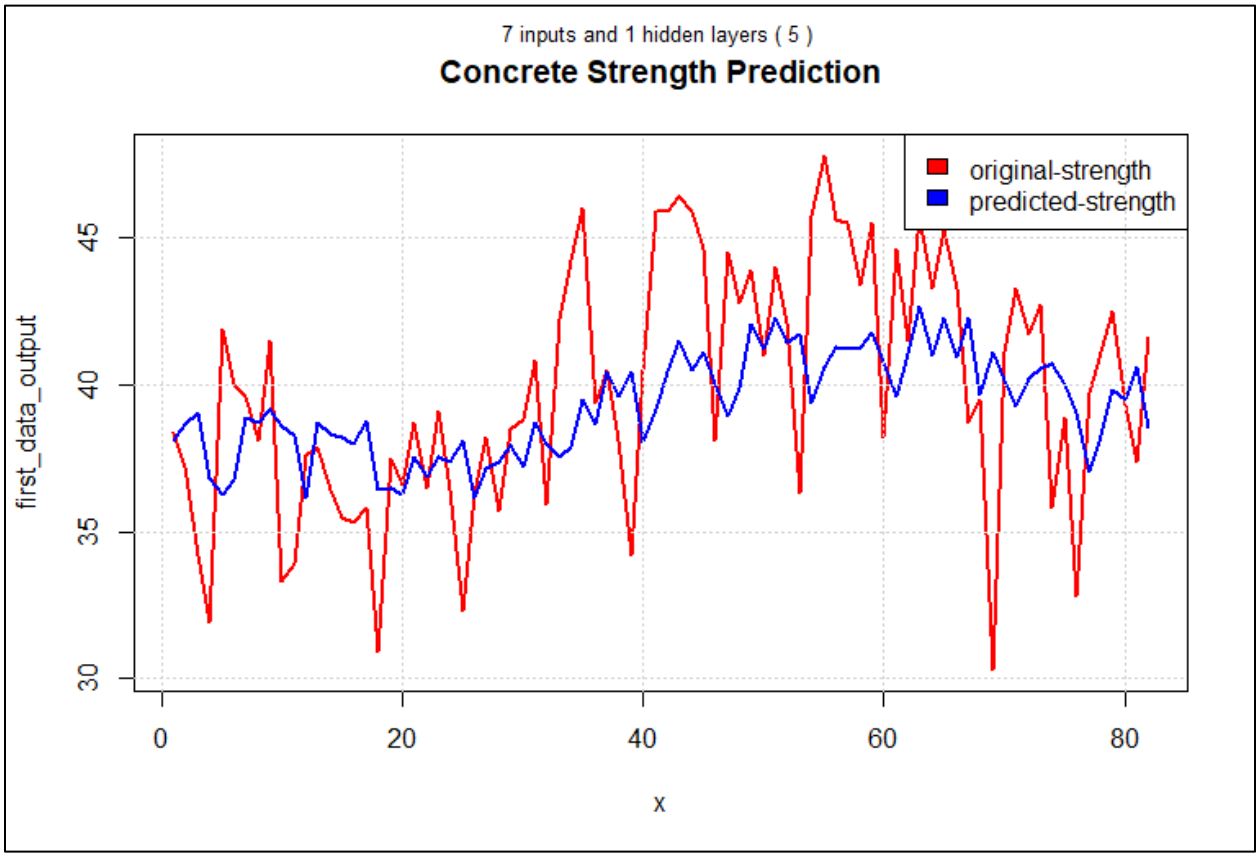


Figure 61:best one hidden layer

2.1.6.2. *Best two hidden layer*

The neural network with 4 inputs and 10 and 5 neurons on two hidden layers performs the best in the above comparison table. Since its accuracy (98.94%) exceeds that of competing two-hidden-layer neural network models, it has been chosen as the finest of its kind.

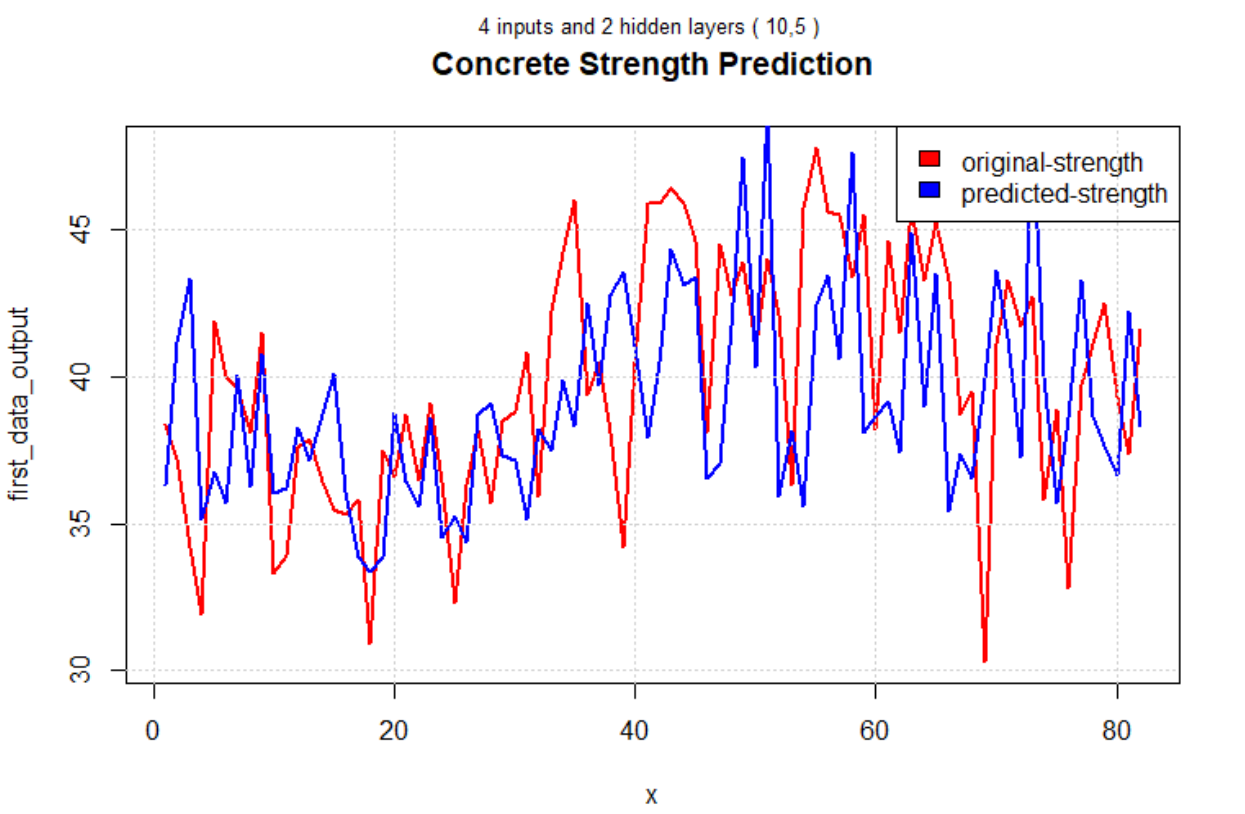


Figure 62:best two hidden layer

2.2. 2nd Subtask Objectives

2.2.1. make a matrix of inputs

```
228
229 #-------------------------------------------------------------------------------------------------#
230 # combine six_hours and seven_hours columns
231 delayed_matrix <- cbind(uow_dataFrame[,2:3], time_delayed_matrix)
232
```

Figure 63:make_matrix

The above code uses the "time_delayed_matrix" to link together columns 2 and 3 of the current data frame "uow_dataFrame" with the "delayed_matrix." These columns are concatenated using the "cbind()" function, and the resulting "delayed_matrix" data frame is returned.

2.2.2. Normalization

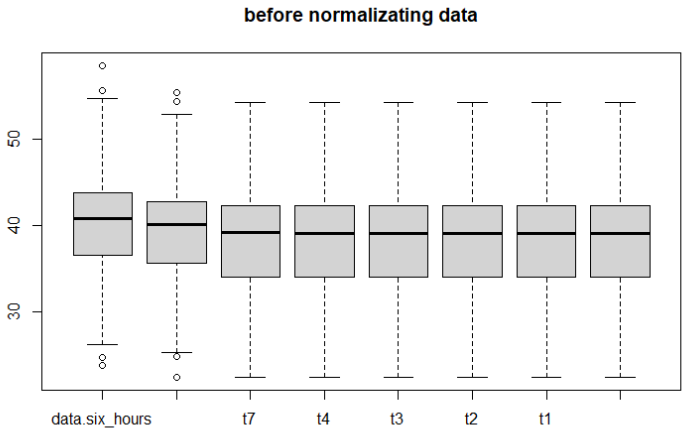


Figure 64:before_normalization_2nd

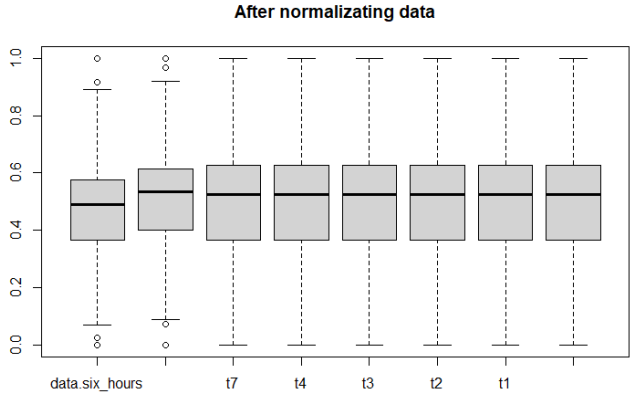
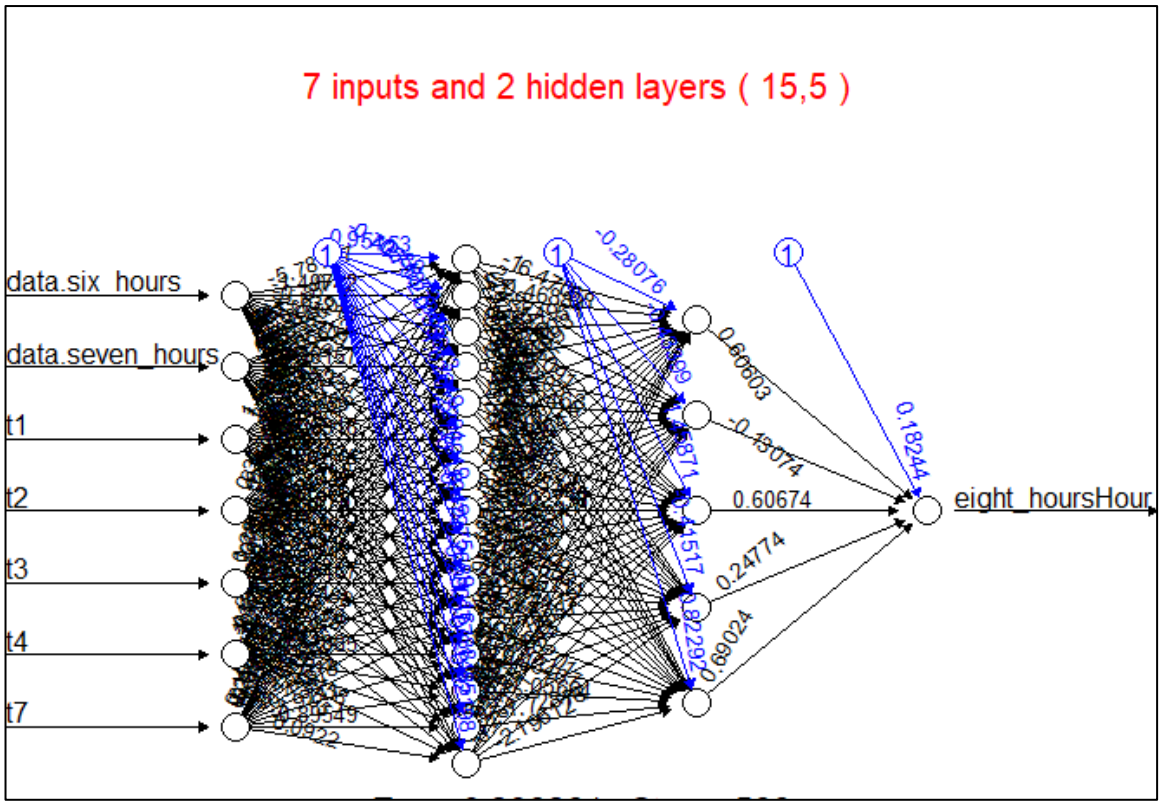
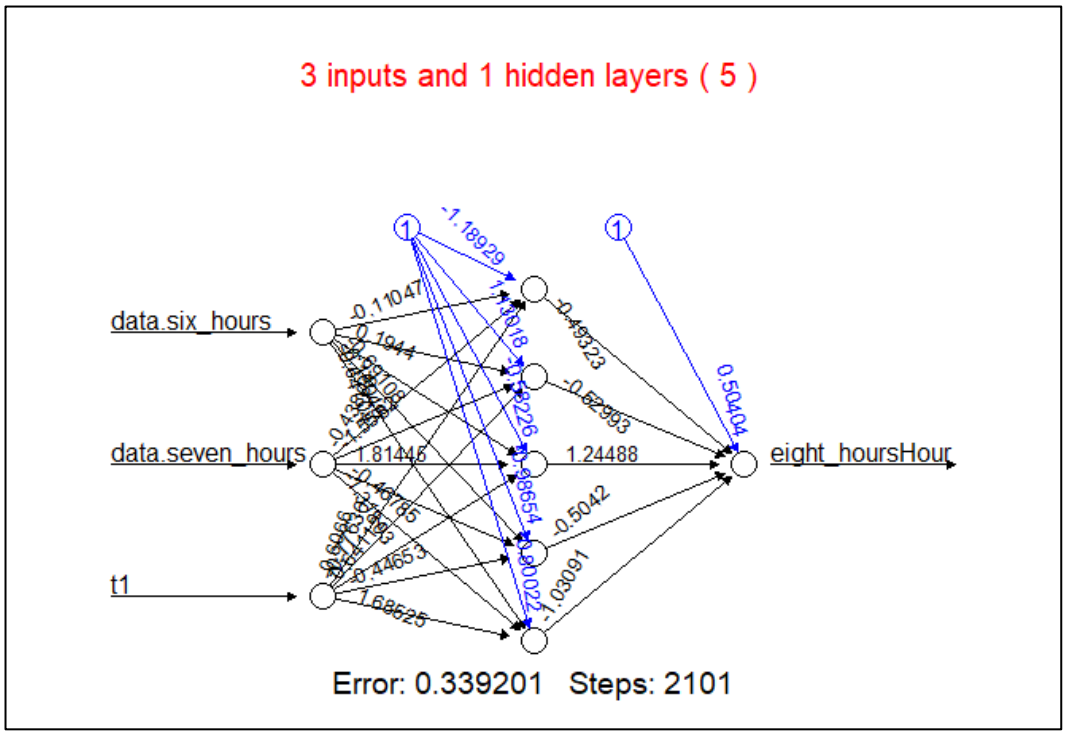
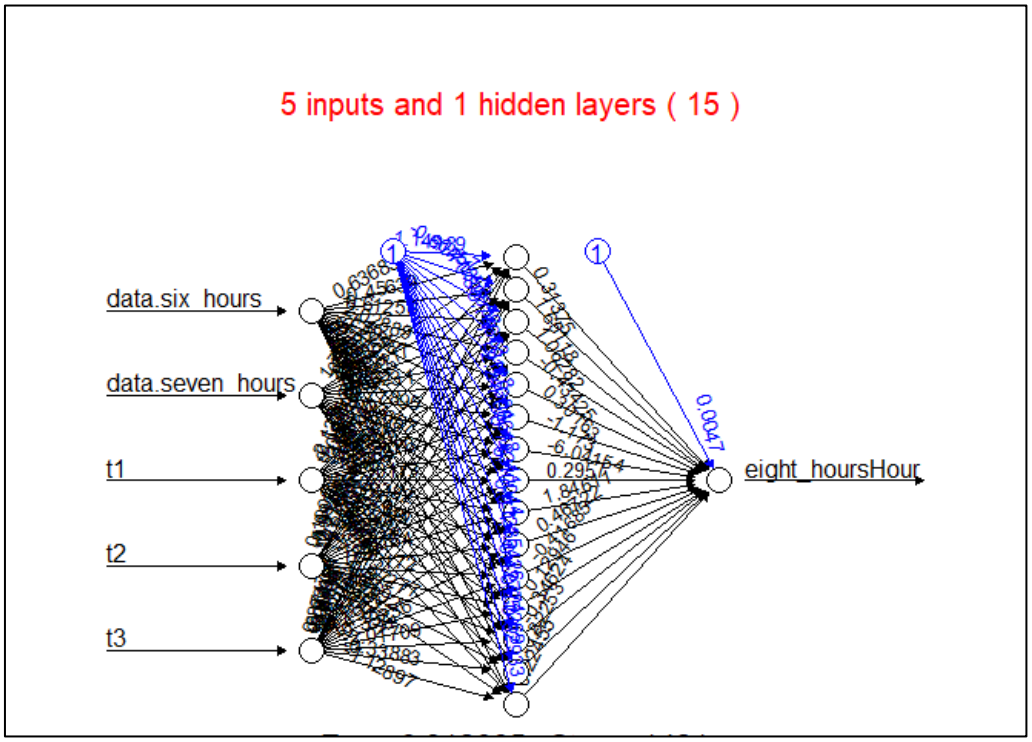


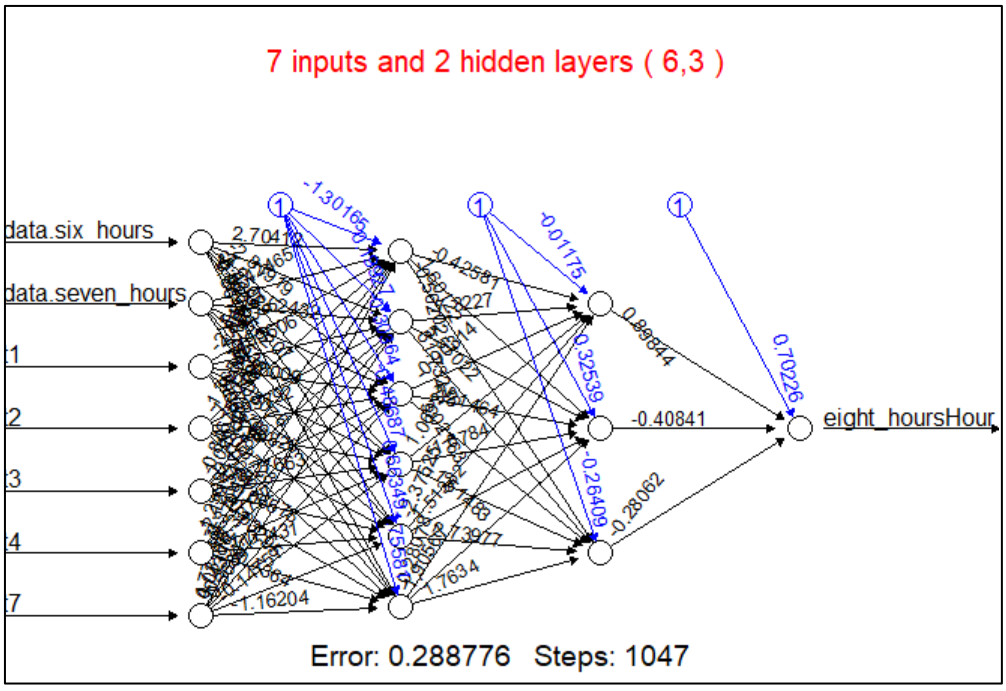
Figure 65:after_normalization_2nd

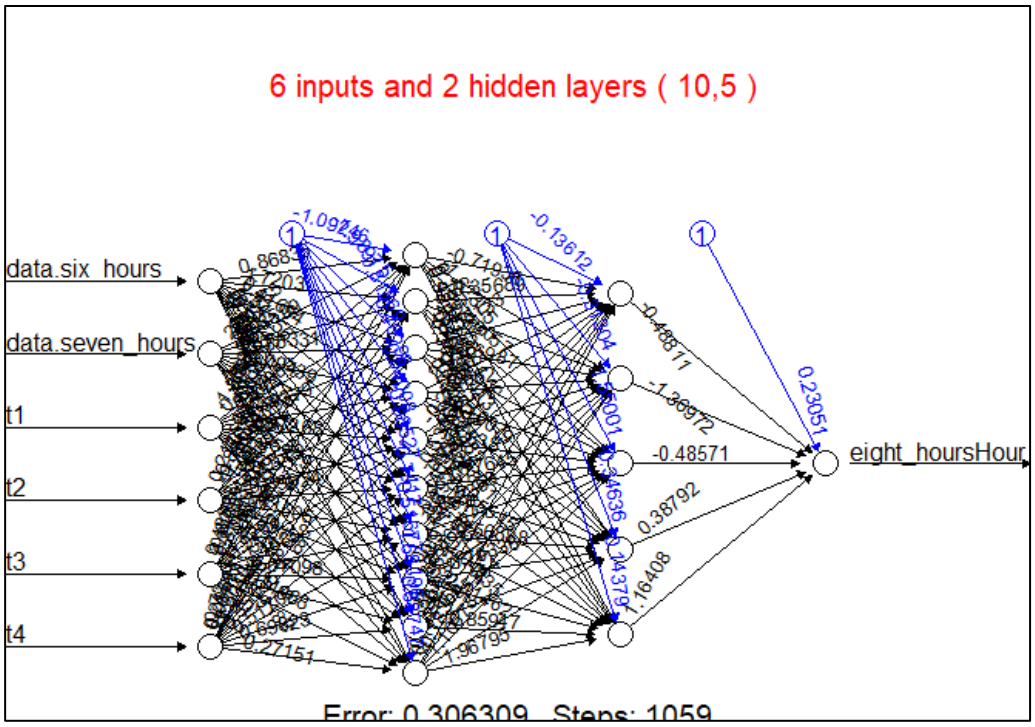
2.2.3. Creation of a NN model






5 inputs and 1 hidden layers (15)
data.six_hours
data.seven_hours
t1
t2
t3
eight_hoursHour


7 inputs and 2 hidden layers (6,3)
data.six_hours
data.seven_hours
t1
t2
t3
t4
t7
eight_hoursHour
Error: 0.288776 Steps: 1047



2.3.4. Comparison table

Input count	*Hidden layers*	*neurons*	*Activation function*	*Linear*	*Accuracy*	*RMSE*	*MAE*	*MAPE*	*sMAPE*
3	1	5	logistic	TRUE	94.1 %	2.688929	2.310642	0.05896598	0.05660477
4	1	10	logistic	TRUE	94.19 %	2.681459	2.288516	0.05826366	0.05593023
5	1	15	logistic	TRUE	94.3 %	2.64334	2.257533	0.05713579	0.05490405
6	2	10,5	logistic	TRUE	94.18 %	2.660069	2.299145	0.05816606	0.05591089
7	2	15,5	logistic	TRUE	94.25 %	2.663941	2.290847	0.05779902	0.0555692
	2	6,3	logistic	TRUE	94.04 %	2.750781	2.372198	0.05980683	0.0574475

Table 2:comparison_table_2nd

2.3.5. Best MLP network

2.3.5.1. Top MLP model for augmented reality

The neural network with seven inputs and five neurons in the hidden layer is the optimal MLP network for the AR approach. This network should be chosen because it has a higher degree of accuracy than others. The graph depicting actual versus predicted values is shown below.

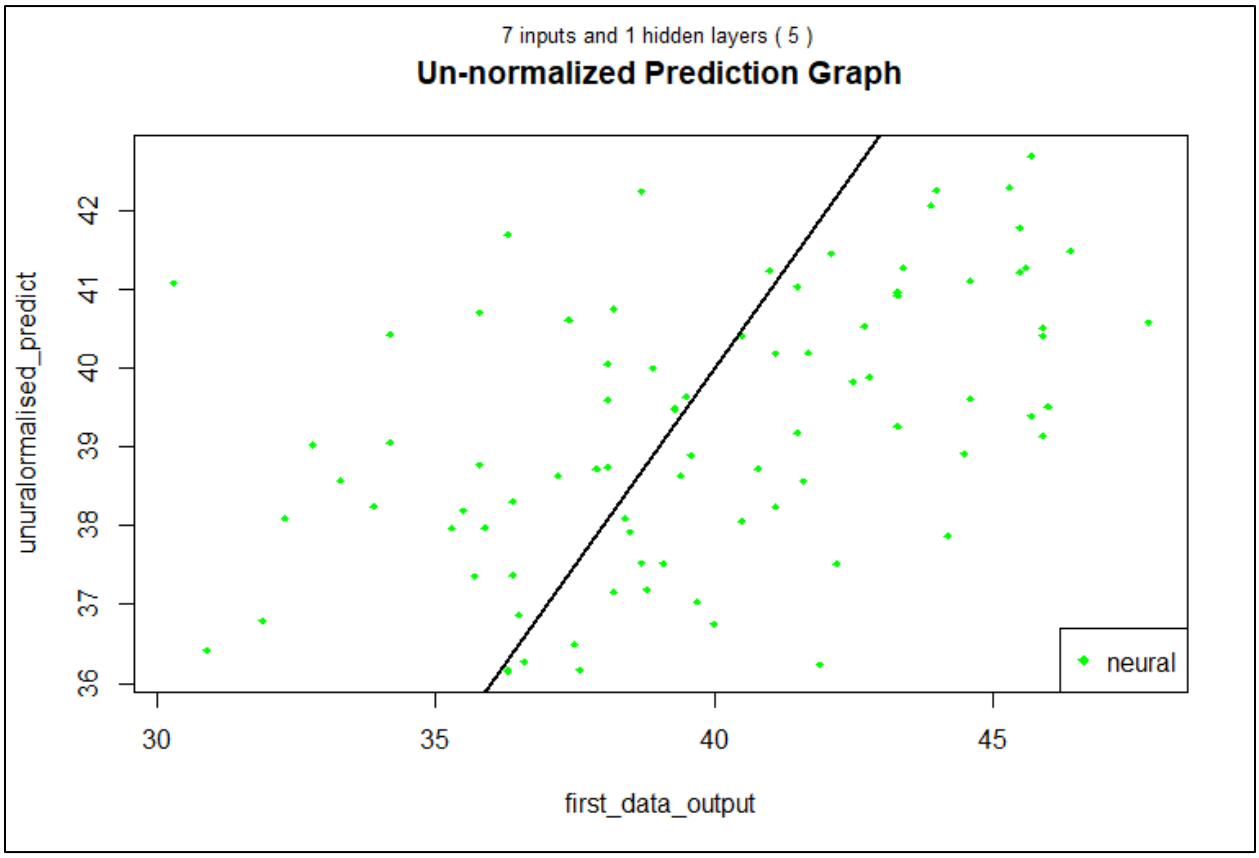


Figure 66:best_MLP_AR_Approch

2.3.5.2. Top MLP network for NARX design

The neural network with seven inputs and fifteen and five neurons in the two hidden layers is regarded as the optimal MLP network for the NARX approach. This network should be chosen because it has a higher degree of accuracy than others. The actual versus predicted graph is depicted in the figure below.

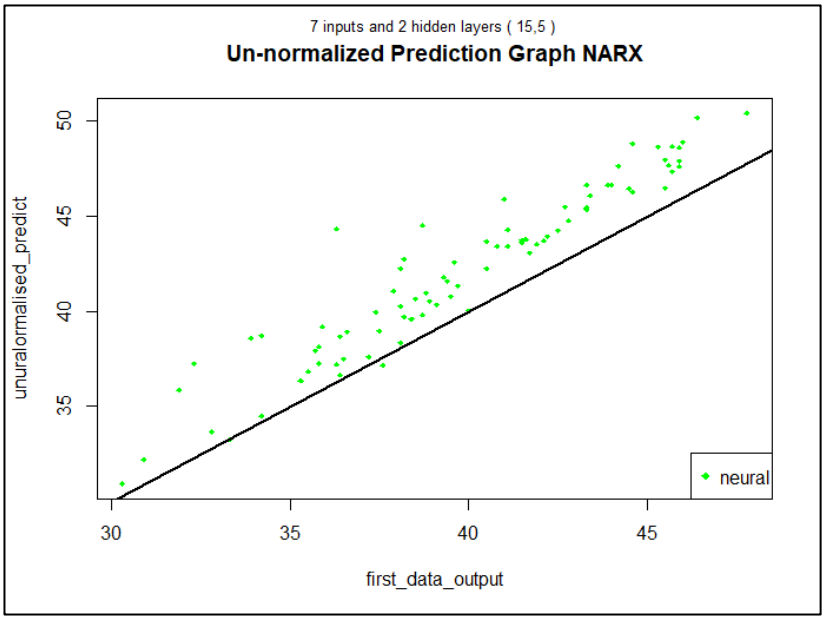
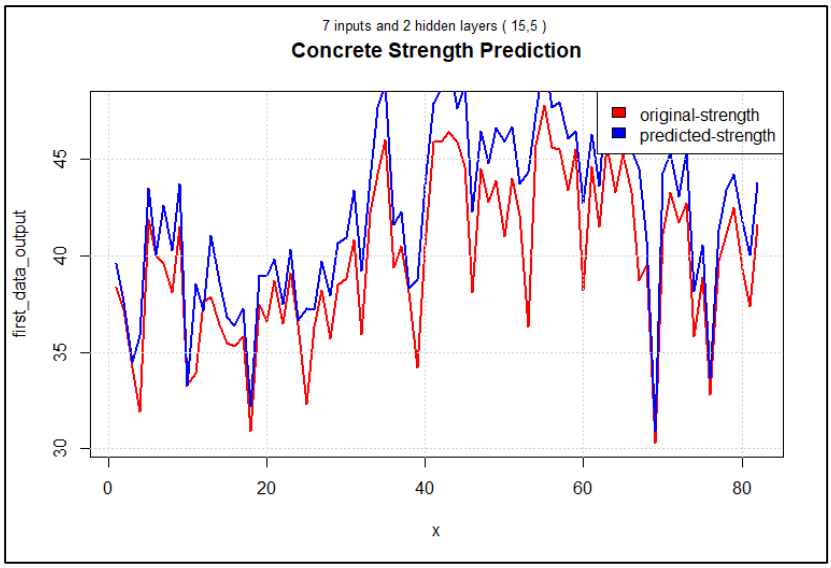


Figure 67:Best_MLP_NARX_network



3. References

Cortez, P., 2022. UCI Machine Learning Repository: Wine Quality Data Set. [online] Archive.ics.uci.edu.Available at: <https://archive.ics.uci.edu/ml/datasets/wine+quality> [Accessed 13 April 2022].

Grob, A., Lenders, A., Schwenker, F., Braun, D. and Fischer, D., 2021. Comparison of short-term electrical load forecasting methods for different building types.

Hammad, M., Jereb, B., Rosi, B. and Dragan, D., 2020. Methods and Models for Electric Load Forecasting: A Comprehensive Review. [online] Sciendo.com. Available at: <https://sciendo.com/pdf/10.2478/jlst-2020-0004> [Accessed 1 May 2022].

Khan, M., 2018. White Wine Exploratory Data Analysis. [online] Rstudio-pubs static.s3.amazonaws.com. Available at: <https://rstudio-pubs static.s3.amazonaws.com/377211_0f3055951ba34266aa8a5ba28cf45cfe.html> [Accessed 4 April 2022].

Rpubs.com. 2021. RPubs - K-means clustering analysis of the white wine dataset using RStudio. [online] Available at: <https://rpubs.com/HassanOUKHOUYA/K-means> [Accessed 4 April 2022].

Singh, A., Ibraheem, Khatoon, S., Muazzam, M. and Chaturvedi, D., 2022. Load forecasting techniques and methodologies: A review. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6508132> [Accessed 1 May 2022].

Verilog, C., 2019. Load Forecasting using Artificial Neural Networks. [online] Available at: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=video&cd=&cad=rja&uact=8&ve

d=2ahUKEwiU1sjrz9D3AhVYeXAKHTYYCF0QtwJ6BAgPEAI&url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3D_eeqqQbKUF8&usg=AOvVaw2m8OQR42lRtHzEtR4Zfset> [Accessed 1 April 2022].

Views, R., 2020. Building A Neural Net from Scratch Using R - Part 1. [online] Rviews.rstudio.com. Available at: <https://rviews.rstudio.com/2020/07/20/shallow-neural-net-from-scratch-using-r part-1/> [Accessed 2 April 2022]

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning (Adaptive Computation and Machine Learning series). MIT Press.

Bishop, C. M. (1995). Neural Networks for Pattern Recognition. Oxford University Press.

Haykin, S. (2009). Neural Networks and Learning Machines (3rd ed.). Prentice Hall.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In Parallel Distributed Processing: Explorations in the Microstructure of Cognition (Vol. 1). MIT Press.

Nielsen, M. A. (2015). Neural Networks and Deep Learning: A Textbook. Determination Press.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786), 504-507.

4. Appendix

4.1. Part 01

```
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(cluster)



# Read the XLSX file by opening it.



Vehicles_dataset <- read_excel("F://SEMESTER 02//5DATA001C.2 Machine Learning and Data Mining//CW//MyCW//vehicles.xlsx")
```

```
# Delete the first column.

Vehicles_dataset <- Vehicles_dataset[-1]


# Examine the data type to determine which columns are strings.


string_cols <- sapply(Vehicles_dataset, class) == "character"

#Use the subset function to eliminate the string column(s).

data <- Vehicles_dataset[, !string_cols]

View(data)

boxplot(data, main = "Before Outlier Removal", outcol="red")


# Before removing the outliers, count the rows and columns and print the results.

rows_of_dataset <- nrow(data)
columns_of_dataset <- ncol(data)

# print the results
cat("Rows to delete before outliers   :", rows_of_dataset, "\n")
```

```
cat("columns to delete before outliers:", columns_of_dataset, "\n")



# Create a function to remove outliers from a single column using the boxplot method.

outliers_remove <- 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.

whithout_outliers_dataset <- apply(data, 2, outliers_remove)

# Remove any rows with missing values

whithout_outliers_dataset <- na.omit(whithout_outliers_dataset)



# After removing the outliers, count the rows and columns and print the results.

rows_of_dataset <- nrow(whithout_outliers_dataset)
columns_of_dataset <- ncol(whithout_outliers_dataset)

# print the results
```

```
cat("Rows to delete before outliers   :", rows_of_dataset, "\n")
cat("Columns to delete before outliers:", columns_of_dataset, "\n")


boxplot(whithout_outliers_dataset,main = "After Outlier Removal")


# Scale up the data set.

scaled_Vehicles_dataset <- scale(whithout_outliers_dataset)
head(scaled_Vehicles_dataset)

boxplot(scaled_Vehicles_dataset)




# Nbclust method

# Set the random seed to 1234 for reproducibility

set.seed(1234)

NBcluster <- NbClust(scaled_Vehicles_dataset, min.nc = 2,max.nc = 10, method = "kmeans")
```

```
# table(NBcluster$Best.n[1,])
barplot(table(NBcluster$Best.n[1,]),
        xlab="Numer of Clusters",
        ylab="Number of Criteria",
        main="Number of Clusters Chosen")



# elbow method

fviz_nbclust(scaled_Vehicles_dataset,kmeans,method = "wss")

# silhouette method

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

# gap static method

fviz_nbclust(scaled_Vehicles_dataset,kmeans,method = "gap_stat")




# 2 clusters

k2 <-kmeans(scaled_Vehicles_dataset, 2)
```

```
k2
autoplot(k2,scaled_Vehicles_dataset,frame=TRUE)

# When k=2, Extract relevant information.

# Cluster centers
cluster_centers <- k2$centers

# Clustered results
cluster_assignments <- k2$cluster

# Between-cluster sum of squares
BSSk2 <- k2$betweenss

# Within-cluster sum of squares
WSSk2 <- k2$tot.withinss

# Total sum of squares
TSSk2 <- BSSk2 + WSSk2

# BSS/TSS ratio
BSS_TSS_ratiok2 <- BSSk2 / TSSk2

# Percentage of variance explained
percent_vark2 <- round(BSS_TSS_ratiok2 * 100, 3)

# Output results
```

```
cat("Cluster centers:\n", cluster_centers, "\n\n")
cat("Cluster assignments:\n", cluster_assignments, "\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("Percentage of variance explained: ", percent_vark2, "%\n")
```

```
#3 clusters

k3 <-kmeans(scaled_Vehicles_dataset, 3)
k3
autoplot(k3,scaled_Vehicles_dataset,frame=TRUE)
```

```
# When k=3, Extract relevant information
# Cluster centers
cluster_centers <- k3$centers

# Clustered results
cluster_assignments <- k3$cluster

# Between-cluster sum of squares
```

```
BSSk3 <- k3$betweenss

# Within-cluster sum of squares
WSSk3 <- k3$tot.withinss

# Total sum of squares
TSSk3 <- BSSk3 + WSSk3

# BSS/TSS ratio
BSS_TSS_ratiok3 <- BSSk3 / TSSk3

# Percentage of variance explained
percent_vark3 <- round(BSS_TSS_ratiok3 * 100, 3)

# Output results
cat("Cluster centers:\n", cluster_centers, "\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("Percentage of variance explained: ", percent_vark3, "%\n")
```

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

```
k <- 2

part01_kmeans_model_2 <- kmeans(scaled_Vehicles_dataset, centers = k, nstart = 25)

# Generate silhouette plot

silhouette_plot2 <- silhouette(part01_kmeans_model_2$cluster, dist(scaled_Vehicles_dataset))

# Determine the typical silhouette's width.

avg_silhouette_width <- mean(silhouette_plot2[, 3])

# Plot the silhouette plot

plot(silhouette_plot2, 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_silhouette_width, lty = 2, lwd = 2, col = "red")




# When Fit k-means model with k=3
```

```
k <- 3
part01_kmeans_model_3 <- kmeans(scaled_Vehicles_dataset, centers = k, nstart = 25)

# Generate silhouette plot

silhouette_plot3 <- silhouette(part01_kmeans_model_3$cluster, dist(scaled_Vehicles_dataset))

# Determine the typical silhouette's width.

avg_silhouette_width <- mean(silhouette_plot3[, 3])

# Plot the silhouette plot

plot(silhouette_plot3, 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_silhouette_width, lty = 2, lwd = 2, col = "red")
```

```
################################################################################
################################################################################
############################################################
```

```
# Conduct a PCA analysis.

pca <- prcomp(scaled_Vehicles_dataset)



# Print the eigenvectors and eigenvalues

print(summary(pca))

# Calculate the principal component (PC) cumulative score.

pca_var <- pca$sdev^2

pca_var_prop <- pca_var / sum(pca_var)

pca_var_cumprop <- cumsum(pca_var_prop)

# Cumulative plot score for each PC

plot(pca_var_cumprop, xlab = "Principal Component", ylab = "Explained Cumulative Proportion
of Variance",

    ylim = c(0, 1), type = "b")

# Make a new converted dataset with attributes representing the principal components.
```

```
pca_trans <- predict(pca, newdata = scaled_Vehicles_dataset)

# Select computers with a minimum cumulative score of over 92%.

selected_pcs <- which(pca_var_cumprop > 0.92)
transformed_data <- pca_trans[, selected_pcs]
boxplot(transformed_data)

View(transformed_data)




#Nbclust method

set.seed(1234)

NBcluster <- NbClust(transformed_data, min.nc = 2,max.nc = 10, method = "kmeans")

barplot(table(NBcluster$Best.n[1,]),
      xlab="Numer of Clusters",
      ylab="Number of Criteria",
      main="Number of Clusters Chosen after PCA2")

#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)

# Print the k-means output

print(kmeans_model_2)
autoplot(kmeans_model_2,transformed_data,frame=TRUE)



cat("For k=2:\n")
# Calculate the within-cluster sum of squares (WSS)
wss_2 <- sum(kmeans_model_2$withinss)
# Print the WSS
```

```
cat("Within-cluster sum of squares (WSS): ", wss_2, "\n")



# 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)

# Print the BSS

cat("Between-cluster sum of squares (BSS): ", bss_2, "\n")



# Calculate the total sum of squares (TSS)

tss_2 <- sum(dist(transformed_data)^2)

# Print the TSS

cat("Total sum of squares (TSS): ", tss_2, "\n")



# Calculate the ratio of BSS to TSS

bss_tss_ratio_2 <- bss_2 / tss_2

# Print the ratio of BSS to TSS

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)

# Print the k-means output

print(kmeans_model_3)
autoplot(kmeans_model_3,transformed_data,frame=TRUE)

cat("For k=3:\n")
# Calculate the within-cluster sum of squares (WSS)
wss_3 <- sum(kmeans_model_3$withinss)
# Print the WSS
cat("Within-cluster sum of squares (WSS): ", wss_3, "\n")


# Calculate the between-cluster sum of squares (BSS)
bss_3 <- sum(kmeans_model_3$size * dist(rbind(kmeans_model_3$centers, colMeans(transformed_data)))^2)
# Print the BSS
cat("Between-cluster sum of squares (BSS): ", bss_3, "\n")


# Calculate the total sum of squares (TSS)
tss_3 <- sum(dist(transformed_data)^2)
# Print the TSS
cat("Total sum of squares (TSS): ", tss_3, "\n")
```

```
# Calculate the ratio of BSS to TSS
bss_tss_ratio_3 <- bss_3 / tss_3
# Print the ratio of BSS to TSS
cat("Ratio of BSS to TSS: ", bss_tss_ratio_3, "\n")
```

```
# Construct a k-means model with k=2

k <- 2
kmeans_model_2 <- kmeans(transformed_data, centers = k, nstart = 25)

# Generate silhouette plot

silhouette_plot <- silhouette(kmeans_model_2$cluster, dist(transformed_data))

# Determine the typical silhouette's width.

avg_silhouette_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_silhouette_width, lty = 2, lwd = 2, col = "red")









# Construct a k-means model with k=3

k <- 3
kmeans_model_3 <- kmeans(transformed_data, centers = k, nstart = 25)

# Generate silhouette plot

silhouette_plot <- silhouette(kmeans_model_3$cluster, dist(transformed_data))

# Determine the typical silhouette's width.

avg_silhouette_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_silhouette_width, lty = 2, lwd = 2, col = "red")
```

```
# Define a function to calculate the Calinski-Harabasz index for a given K-means clustering result and data

calinski_harabasz_pca2 <- function(cluster_result, data) {

  # Extract the number of clusters from the clustering result

  k2 <- length(unique(cluster_result$cluster))

  # Extract the number of rows in the data

  n2 <- nrow(data)
```

```
  # Extract the between-cluster sum of squares (BSS) and within-cluster sum of squares (WSS) from the clustering result

  BSS2 <- cluster_result$betweenss
  WSS2 <- cluster_result$tot.withinss

  # Calculate the Calinski-Harabasz index

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

  # Return the Calinski-Harabasz index

  return(ch_index2)
}

# Call the calinski_harabasz_pca function to calculate the index for a specific K-means clustering result and data

ch_index_pca_2 <- calinski_harabasz_pca2(kmeans_model_2, transformed_data)

# Print the calculated index to the console for interpretation

cat("The Calinski-Harabasz index for the K-means (k=2) clustering result is:", ch_index_pca_2, "\n")
```

```
# Define a function to calculate the Calinski-Harabasz index for a given K-means clustering result and data

calinski_harabasz_pca3 <- function(cluster_result, data) {

  # Extract the number of clusters from the clustering result

  k3 <- length(unique(cluster_result$cluster))

  # Extract the number of rows in the data

  n3 <- nrow(data)

  # Extract the between-cluster sum of squares (BSS) and within-cluster sum of squares (WSS) from the clustering result

  BSS3 <- cluster_result$betweenss
  WSS3 <- cluster_result$tot.withinss

  # Calculate the Calinski-Harabasz index

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

```
  # Return the Calinski-Harabasz index

  return(ch_index3)
}

# Call the calinski_harabasz_pca function to calculate the index for kmeans_model_3 and transformed_data

ch_index_pca_3 <- calinski_harabasz_pca3(kmeans_model_3, transformed_data)

# Print the calculated index to the console for interpretation

cat("The Calinski-Harabasz index for the K-means (k=3) clustering result is:", ch_index_pca_3, "\n")
```

4.2. Part 02

```
# import libraries

library(readxl)

library(dplyr)

library(neuralnet)

library(Metrics)

library(grid)

library(gridExtra)


# import dataset

data <- read_excel("F://SEMESTER 02//5DATA001C.2 Machine Learning and Data Mining//CW//MyCW//uow_consumption.xlsx", sheet = 1)


# change column names

names(data)[2] <- 'six_hours'

names(data)[3] <- 'seven_hours'

names(data)[4] <- 'eight_hours'


# change date to numeric

date <-factor(data$date)

date <-as.numeric(date)


# create data frame

uow_dataFrame <- data.frame(date,data$'six_hours',data$'seven_hours',data$'eight_hours')
```

```
# plot the eight_hours_column
eight_hours_column <- c(uow_dataFrame$data.eight_hours)
plot(eight_hours_column, type = "l")

# create a I/O  matrix
time_delayed_matrix <- bind_cols(t7 = lag(eight_hours_column,8),
                        t4 = lag(eight_hours_column,5),
                        t3 = lag(eight_hours_column,4),
                        t2 = lag(eight_hours_column,3),
                        t1 = lag(eight_hours_column,2),
                        eight_hoursHour = eight_hours_column)

delayed_matrix <- na.omit(time_delayed_matrix)

# Separating dataset into test and training sets.
train_dataset <- delayed_matrix[1:380,]
test_dataset <- delayed_matrix[381:nrow(delayed_matrix),]

# Determining the minimum and maximum values in the training set
first_min_value <- min(train_dataset)
first_max_value <- max(train_dataset)

# Extracting the output data from the testing dataset
first_data_output <- test_dataset$eight_hoursHour

# normalization function
```

```
normalization <- function(x){
  return ((x - min(x)) / (max(x) - min(x)))
}

# Un-normalization function
un_normalization <- function(x, min, max) {
  return( (max - min)*x + min )
}

# The normalization function is used to normalise the data.
time_delayedNormaliz <- as.data.frame(lapply(delayed_matrix[1:ncol(delayed_matrix)], normalization))

# Separating the normalised data into test and train sets
train_datasetNormaliz <- time_delayedNormaliz[1:380,]
test_datasetNormaliz <- time_delayedNormaliz[381:nrow(delayed_matrix),]

# See how the data looks like before and after normalisation with a Boxplot
boxplot(delayed_matrix, main="before normalizing data")
boxplot(time_delayedNormaliz, main="After normalizing data")

# Producing Test Information for Each and EveryDelay
t1_testDataSet <- as.data.frame(test_datasetNormaliz[, c("t1")])
t2_testDataSet <- test_datasetNormaliz[, c("t1", "t2")]
t3_testDataSet <- test_datasetNormaliz[, c("t1", "t2", "t3")]
t4_testDataSet <- test_datasetNormaliz[, c("t1", "t2", "t3", "t4")]
t7_testDataSet <- test_datasetNormaliz[, c("t1", "t2", "t3", "t4", "t7")]
```

```
#Function to train an AR model

ModelTrain <- function(formula, hiddenVal, isLinear, actFunc,inputs,hidden){

  # Defining a title for the plot

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

  # Establishing a reproducible seed

  set.seed(1234)

  # The neuralnet method is used to train a model of a neural network.

  nural <- neuralnet(formula, data = train_datasetNormaliz, hidden = hiddenVal, act.fct = actFunc, linear.output = isLinear)

  # Using a neural network to generate a plot

  plot(nural)

  # Keeping the plot in a variable

  plot_panel <- grid.grab(wrap = TRUE)

  # Creating a title grob

  plot_title <- textGrob(my_title,
                         x = .5, y = .20,
                         gp = gpar(lineheight_hours = 2,
                                   fontsize = 15, col = 'red',
                                   adj = c(1, 0)
```

```
                    )
  )

  # Stacking the title and main panel, and plotting
  grid.arrange(
    grobs = list(plot_title,
                 plot_panel),
    height_hourss = unit(c(.15, .85), units = "npc"),
    width = unit(1, "npc")
  )
  dev.new()
  dev.off()
  return(nural)
}

# Define function ModelTest with input arguments nuralModel, testing_df, inputs, hidden
ModelTest <- function(nuralModel, testing_df, inputs, hidden){

  # Print the number of inputs and hidden layers specified
  cat("There are", inputs, "inputs and", length(hidden), "hidden layers", "(", paste(hidden, collapse=","), ") \n")

  # Create a title for the plots based on the inputs and hidden layers specified
  my_title <- paste(inputs, "inputs and", length(hidden), "hidden layers", "(", paste(hidden, collapse=","), ") \n")

  # Use the test data to calculate the output of the neural network.
  nnResults <- compute(nuralModel, testing_df)
```

```
# Extract the predicted values and un-normalize them using the min and max values of the original data

predict <- nnResults$net.result

unuralormalised_predict <- un_normalization(predict, first_min_value, first_max_value)


# Find the difference between your expected results and the actual ones.

devia = ((first_data_output - unuralormalised_predict) / first_data_output)


# Calculate the model accuracy as 1 minus the absolute mean deviation

modelAccu = 1 - abs(mean(devia))

accuracy = round(modelAccu * 100, digits = 2)


# Plot the predicted vs. actual output values with the un-normalized data

plot(first_data_output, unuralormalised_predict, col = 'green', main = "Un-normalized Prediction Graph", pch = 18, cex = 0.7)

mtext(my_title,  side = 3, line = 2, cex = 0.8)

abline(0,1,lwd=2)

legend("bottomright", legend = 'neural', pch = 18, col = 'green')


# Create a new plot showing the original output values vs. the predicted values with un-normalized data

x = 1:length(first_data_output)

plot(x, first_data_output, col = "red", type = "l", lwd=2, main = "Concrete Strength Prediction")

mtext(my_title,  side = 3, line = 2, cex = 0.8)

lines(x, unuralormalised_predict, 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()

  # Calculate the RMSE, MAE, MAPE, and sMAPE metrics for the model
  rmse = rmse(first_data_output, unuralormalised_predict)
  mae = mae(first_data_output, unuralormalised_predict)
  mape = mape(first_data_output, unuralormalised_predict)
  smape = smape(first_data_output, unuralormalised_predict)

  # Get the model's RMSE, MAE, MAPE, and sMAPE values.
  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 the un-normalized predicted output values
  return(unuralormalised_predict)
}

# t1 with different hidden layer sizes
hidden_layers <- list( c(6),c(5, 3))

# loop through different hidden layer sizes
for (i in seq_along(hidden_layers)) {
  # train model using t1 as input and current hidden layer size
```

```
 model <- ModelTrain(eight_hoursHour ~ t1, hidden_layers[[i]], isLinear = FALSE, "tanh",1,hidden_layers[[i]])
 # test model on t1_testDataSet
 pred <- ModelTest(model, t1_testDataSet,1,hidden_layers[[i]])
}



# t2 with different hidden layer sizes
# train model using t1 and t2 as input and hidden layer size of 10
t2_train <- ModelTrain(eight_hoursHour ~ t1 + t2, c(10),isLinear = TRUE, "logistic",2,c(10))
# test model on t2_testDataSet
test_t2_predict <- ModelTest(t2_train, t2_testDataSet,2,c(10))



# t3 with different hidden layer sizes
hidden_layers <- list( c(5),c(10),c(5,4))

# loop through different hidden layer sizes
for (i in seq_along(hidden_layers)) {
 # train model using t1, t2, and t3 as input and current hidden layer size
 model <- ModelTrain(eight_hoursHour ~ t1 + t2 + t3 ,hidden_layers[[i]],isLinear = TRUE, "logistic",3,hidden_layers[[i]])
 # test model on t3_testDataSet
 pred <- ModelTest(model, t3_testDataSet,3,hidden_layers[[i]])
}
```

```
# t4 with different hidden layer sizes
hidden_layers <- list( c(5),c(15),c(5,2),c(10,5))

# loop through different hidden layer sizes
for (i in seq_along(hidden_layers)) {
  # train model using t1, t2, t3, and t4 as input and current hidden layer size
  model <- ModelTrain(eight_hoursHour ~ t1 + t2 + t3 + t4,hidden_layers[[i]],isLinear = TRUE, "logistic",4,hidden_layers[[i]])
  # test model on t4_testDataSet
  pred <- ModelTest(model, t4_testDataSet,4,hidden_layers[[i]])
}


# t7 with different hidden layer sizes
hidden_layers <- list( c(5),c(10),c(6,4),c(8,5),c(10,5))

# loop through different hidden layer sizes
for (i in seq_along(hidden_layers)) {
  # train model using t1, t2, t3, t4, and t7 as input and current hidden layer size
  model <- ModelTrain(eight_hoursHour ~ t1 + t2 + t3 + t4 + t7,hidden_layers[[i]],isLinear = TRUE, "logistic",7,hidden_layers[[i]])
  # test model on t7_testDataSet
  pred <- ModelTest(model, t7_testDataSet,7,hidden_layers[[i]])
}


#-------------------------------------------------------------------------------------------#
```

```
# combine six_hours and seven_hours columns
delayed_matrix <- cbind(uow_dataFrame[,2:3], time_delayed_matrix)

# Remove rows with missing values
delayed_matrix <- na.omit(delayed_matrix)

# Separate data into a training set and a testing set.
train_dataset <- delayed_matrix[1:380,]
test_dataset <- delayed_matrix[381:nrow(delayed_matrix),]

# Determine the training dataset's minimum and maximum values.
first_min_value <- min(train_dataset)
first_max_value <- max(train_dataset)

# Get the output data for the test dataset
first_data_output <- test_dataset$eight_hoursHour

# Apply normalization to the dataset
time_delayedNormaliz <- as.data.frame(lapply(delayed_matrix[1:ncol(delayed_matrix)], normalization))

# Separate the normalised data set into a training set and a test set.
train_datasetNormaliz <- time_delayedNormaliz[1:380,]
test_datasetNormaliz <- time_delayedNormaliz[381:nrow(delayed_matrix),]

# Plot boxplots to visualize the data before and after normalization
boxplot(delayed_matrix, main="before normalizating data")
```

```
boxplot(time_delayedNormaliz, main="After normalizating data")


# Create testing data for each timeDelay

t1_testDataSet <- test_datasetNormaliz[, c("data.six_hours","data.seven_hours", "t1")]

t2_testDataSet <- test_datasetNormaliz[, c("data.six_hours","data.seven_hours", "t1", "t2")]

t3_testDataSet <- test_datasetNormaliz[, c("data.six_hours","data.seven_hours", "t1", "t2", "t3")]

t4_testDataSet <- test_datasetNormaliz[, c("data.six_hours","data.seven_hours", "t1", "t2", "t3", "t4")]

t7_testDataSet <- test_datasetNormaliz[, c("data.six_hours","data.seven_hours", "t1", "t2", "t3", "t4", "t7")]


# Define the input features and their corresponding test data

inputs <- c("t1", "t2", "t3","t4","t7")

test_dataset <- list(t1_testDataSet, t2_testDataSet, t3_testDataSet, t4_testDataSet, t7_testDataSet)


# Define the different hidden layer configurations to test

h1 <- list(c(5))

h2 <- list(c(10))

h3 <- list(c(15))

h4 <- list(c(10,5))

h5 <- list(c(15,5),c(6,3))

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


# Loop through the different numbers of input variables

for (i in seq_along(inputs)) {

 # Iterate over the possible arrangements of hidden layers given the current number of input variables.
```

```
  for(j in seq_along(hidden_layers[[i]])){
    # Get the current input variables
    current_inputs <- inputs[1:i]

    # Get the current hidden layer configuration
    currentHidden <- hidden_layers[[i]][[j]]

    # Train the model using the current input variables and hidden layer configuration
    formula <- as.formula(paste("eight_hoursHour ~ data.six_hours + data.seven_hours +", paste(current_inputs, collapse="+")))
    model <- ModelTrain(formula, currentHidden, isLinear = TRUE, "logistic", (length(current_inputs)+2), currentHidden)

    # Test the model using the test dataset and print the predictions
    test_predict <- ModelTest(model, test_dataset[[i]], (length(current_inputs)+2), currentHidden)
  }
}
```