



INFORMATICS  
INSTITUTE OF  
TECHNOLOGY

UNIVERSITY OF  
WESTMINSTER 

**University Of Westminster**  
**Information Institute of Technology**

**5DATA001C.2 Machine Learning and Data Mining**

**Name - Ahmed Ammar Ismeth Mohideen (20210536/w1869538)**

## 1<sup>st</sup> Subtask Objectives

a)

### Clustering

#### Data and Library Selection

The provided dataset consists of 846 observations of vehicle samples with 18 attributes each defined with different correlations to the output attribute “Type of cars”.

Specific libraries were selected to perform various tasks like “readxl” library to read excel files. “Nbclust” libraries for clustering and determining the number of clusters. The clusters used are as below :

```
1 library(NbClust)
2 library(ggplot2)
3 library(gridExtra)
4 library(factoextra)
5 library(cluster)
6 library(MASS)|
```

#### Data Preprocessing

Outlier removal and data scaling are the two categories under which data preparation can be categorized.

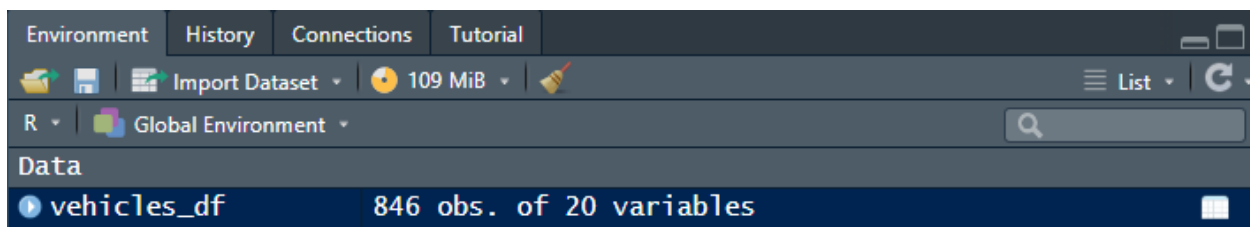
#### Importing of Data

The dataset was imported into the workspace using the readxl library and a was assigned a variable called “vehicles\_df”.

```
1 library(readxl)
2 vehicles_df <- read_excel("vehicles.xlsx")
```

Figure 1 : Importing vehicles data

As seen by the above we can verify the data was imported correctly through the environment data tab which gives us a prompt of 846 observations.



We can run the following commands to get a better understanding of the dataset. Firstly, we run the head() function to view the first n number of rows in each variable or attribute. By default, the head() function would return the first 6 rows of data if n is not specified by the user.

```

4 library(dplyr)
5 vehicles_df %>%
6 head()

> library(dplyr)
> vehicles_df %>%
+ head()
# A tibble: 6 x 20
  Samples Comp Circ D.Circ Rad.Ra Pr.Axis.Ra Max.L.Ra Scat.Ra Elong Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis Sc.Var.maxis
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 1 95 48 83 178 72 10 162 42 20 159 176 379
2 2 91 41 84 141 57 9 149 45 19 143 170 330
3 3 104 50 106 209 66 10 207 32 23 158 223 635
4 4 93 41 82 159 63 9 144 46 19 143 160 309
5 5 85 44 70 205 103 52 149 45 19 144 241 325
6 6 107 57 106 172 50 6 255 26 28 169 280 957
# i 7 more variables: Ra.Gyr <dbl>, Skew.Maxis <dbl>, Skew.maxis <dbl>, Kurt.maxis <dbl>, Kurt.Maxis <dbl>, Holl.Ra <dbl>,
# Class <chr>
>

```

Figure 2 : Code snippet and result of head()

We use the summary() function to display all the basic information for each attribute, such as the minimum, maximum, median, mean, first quartile, and third quartile, after receiving the first six rows of the dataset for each attribute to better comprehend the statistical data of each attribute. This gives us an understanding of the range of observations for each attribute and will also assist us to get an understanding of the IQR, which facilitates the removal of outliers.

```

8 vehicles_df %>% summary ()
9:1 (Top Level)
R Script

Console Terminal Background Jobs
R 4.2.2 - D:/CW - Machine/CW ML/

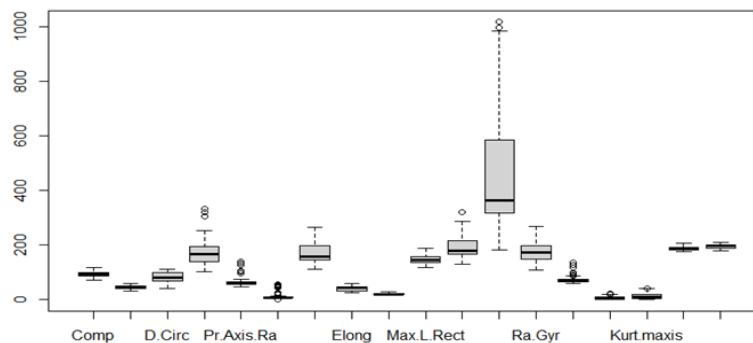
> vehicles_df %>% summary ()
  Samples  Comp      Circ      D.Circ      Rad.Ra      Pr.Axis.Ra      Max.L.Ra
Min.   : 1.0   Min.   :73.00   Min.   :33.00   Min.   :40.00   Min.   :104.0   Min.   :47.00   Min.   :2.000
1st Qu.:212.2  1st Qu.:87.00   1st Qu.:40.00   1st Qu.:70.00   1st Qu.:141.0   1st Qu.:57.00   1st Qu.:7.000
Median :423.5  Median :93.00   Median :44.00   Median :80.00   Median :167.0   Median :61.00   Median :8.000
Mean   :423.5  Mean   :93.68   Mean   :44.86   Mean   :82.09   Mean   :168.9   Mean   :61.69   Mean   :8.567
3rd Qu.:634.8  3rd Qu.:100.00  3rd Qu.:49.00   3rd Qu.:98.00   3rd Qu.:195.0   3rd Qu.:65.00   3rd Qu.:10.000
Max.   :846.0  Max.   :119.00  Max.   :59.00   Max.   :112.00  Max.   :333.0   Max.   :138.00  Max.   :55.000

  Scat.Ra      Elong      Pr.Axis.Rect      Max.L.Rect      Sc.Var.Maxis      Sc.Var.maxis      Ra.Gyr
Min.   :112.0   Min.   :26.00   Min.   :17.00   Min.   :118   Min.   :130.0   Min.   :184.0   Min.   :109.0
1st Qu.:146.2  1st Qu.:33.00  1st Qu.:19.00  1st Qu.:137   1st Qu.:167.0   1st Qu.:318.2   1st Qu.:149.0
Median :157.0  Median :43.00  Median :20.00  Median :146   Median :178.5   Median :364.0   Median :173.0
Mean   :168.8  Mean   :40.93  Mean   :20.58  Mean   :148   Mean   :188.6   Mean   :439.9   Mean   :174.7
3rd Qu.:198.0  3rd Qu.:46.00  3rd Qu.:23.00  3rd Qu.:159   3rd Qu.:217.0   3rd Qu.:587.0   3rd Qu.:198.0
Max.   :265.0  Max.   :61.00  Max.   :29.00  Max.   :188   Max.   :320.0   Max.   :1018.0   Max.   :268.0

  Skew.Maxis      Skew.maxis      Kurt.maxis      Kurt.Maxis      Holl.Ra      Class
Min.   :59.00   Min.   :0.000   Min.   :0.0   Min.   :176.0   Min.   :181.0   Length:846
1st Qu.:67.00   1st Qu.:2.000   1st Qu.:5.0   1st Qu.:184.0   1st Qu.:190.2   Class :character
Median :71.50   Median :6.000   Median :11.0   Median :188.0   Median :197.0   Mode  :character
Mean   :72.46   Mean   :6.377   Mean   :12.6   Mean   :188.9   Mean   :195.6
3rd Qu.:75.00   3rd Qu.:9.000   3rd Qu.:19.0   3rd Qu.:193.0   3rd Qu.:201.0
Max.   :135.00  Max.   :22.000   Max.   :41.0   Max.   :206.0   Max.   :211.0

```

Once this is done it is also vital that we remove the extreme values in the dataset, which are also known as outliers to help reduce the bias in statistical analysis and to further improve the accuracy of the model. The box plot below shows the outliers as circles and thus these could deteriorate the final results.



The detect outlier function takes vector X as an input which is the data frame of vehicles and it takes the interquartile range (IQR) method to detect the outliers in the dataset. The function begins with calculating Q1 the first and third quartiles(Q1 and Q3) and find the difference between the two values. The data outside this required range is considered as outliers. Thereby, returning a vector of True and if not it returns False as shown in the code below

```
21. detect_outlier <- function(x) {
22.   #Uses the interquartile range to locate the outliers
23.   Q1 <- quantile(x, probs = 0.25)
24.   Q3 <- quantile(x, probs = 0.75)
25.   IQR <- Q3 - Q1
26.   outliers <- x > (Q3 + 1.5 * IQR) | x < (Q1 - 1.5 * IQR)
27.   return(outliers)
28. }
```

Thereafter the above function is called inside the remove the outliers functions, as shown in the figure below which iterates through every column in the dataset, while ignoring the first column as it is not relevant and is an input, this removes outliers through the detection of outlier function this causing the removal of rows that are outliers. Following which the function then returns a new data frame without outliers, This is then stored in *clean\_data*. After the removal of outliers, the final training dataset displays 814 records which also mean that 32 records have been removed.

```
29. # define remove_outlier() function
30. remove_outlier <- function(dataframe) {
31.   for (i in 2:ncol(dataframe)) { # ignores the sample column
32.     if (is.numeric(dataframe[[i]])) { # skip non-numeric columns
33.       dataframe <- dataframe[!detect_outlier(dataframe[[i]]), ]
34.     }
35.   }
36.   message("Outliers have been removed")
37.   return(dataframe)
38. }
39. clean_data <- remove_outlier(vehicles_df)
```

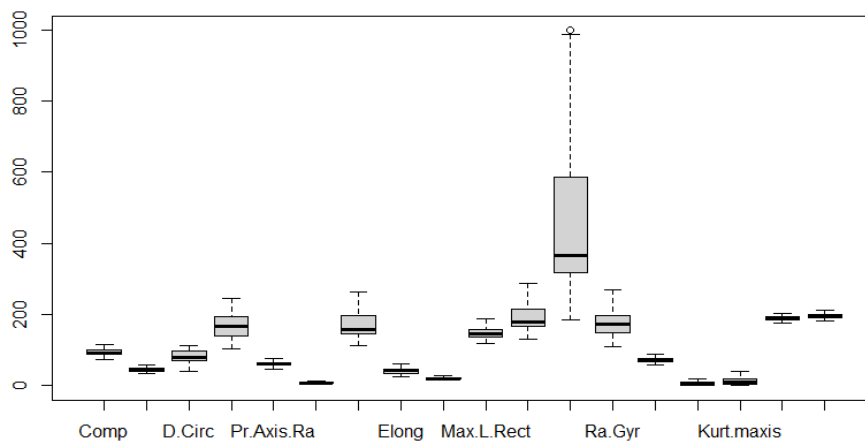
Next, the *clean\_data* [,2:19) columns are then stored in the *normalized\_data* set. To scale or standardize the data we use the *scale()* function. The data is changed by scaling it by the standard deviation and centering it around the mean. A new set of data with a mean of 0 and a standard deviation of 1 are produced by this change. The standardized values are also known as Z-scores. We used the '*scale()*' function to calculate the Z-Scores of the dataset. Firstly, the function subtracts the mean of the data

from each value, and then divides by the standard deviation. The result is a data frame of standardized values with a mean of 0 and a standard deviation of 1. By performing the scaling, we can compare variables that have different units and scales, thus making it easier to visualize and interpret data.

The columns with numeric values of the `clean_data` are then stored in `numeric_col` variable and there after the scaling is done for `numeric_col` and is stroed in `normalized_data`.

```
40 #Calculate the z-score of clean_data
41 numeric_col <- clean_data[,2:19]
42 normalized_data <- scale(numeric_col)
43
44 #View the standardized data
45 normalized_data
```

The figure below shows the boxplot after normalization and scaling.



## b) Determining the Number of Clusters

### NbClust

The library “NbClust” and the function `NbClust()` were used in order to obtain the optimal number of clusters. NbClust uses a majority rule where the maximum proposed cluster values are taken as the final concluded value for the optimum number of clusters for the given dataset. The below code was used to obtain the optimal number of clusters.

```
49 #NbClust Method
50 set.seed(1445)
51 num_clusters <- NbClust(normalized_data,distance="euclidean",min.nc = 2,max.nc =8,method = "kmeans",index="all")
52 table(num_clusters$Best.nc) |
53
54
```

```

> set.seed(1445)
> num_clusters <- NbClust(normalized_data,distance="euclidean",min.nc = 2,max.nc =8,method = "kmeans",index="all")
*** : 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:
* 6 proposed 2 as the best number of clusters
* 8 proposed 3 as the best number of clusters
* 7 proposed 4 as the best number of clusters
* 2 proposed 7 as the best number of clusters
* 1 proposed 8 as the best number of clusters

***** Conclusion *****

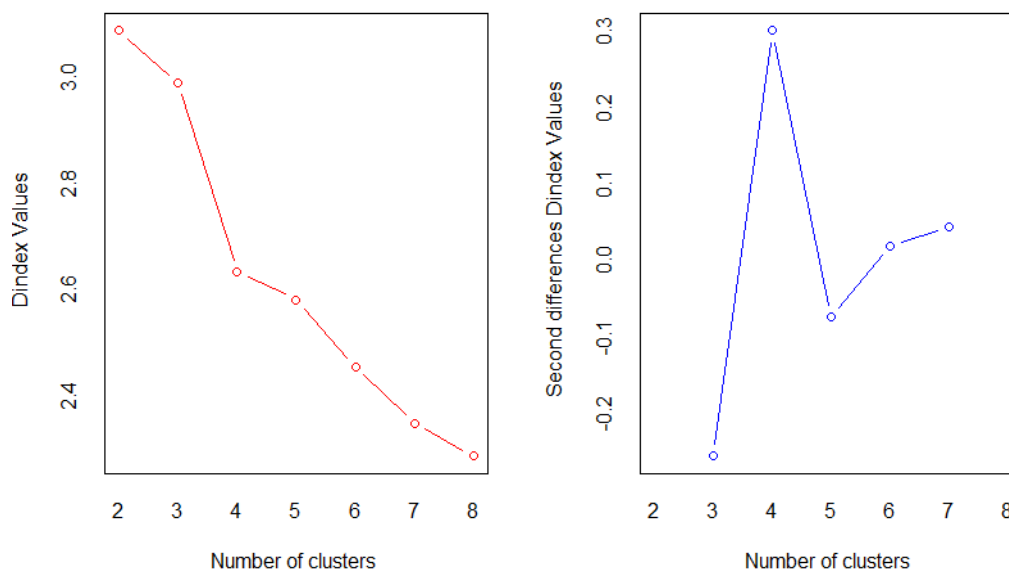
* According to the majority rule, the best number of clusters is 3
*****

```

The graphs below can also be depicted as follows. NbClust() uses two distinct graphical representations to determine the optimum number of clusters.

The Hubert index is a graphical method of determining the number of clusters. In the plot of Hubert index, it seeks a significant knee that corresponds to a significant increase of the value of the measure. For example the significant peak in Hubert index second differences plot. The significant peak in Hubert index of second differences plot is an indication of this.

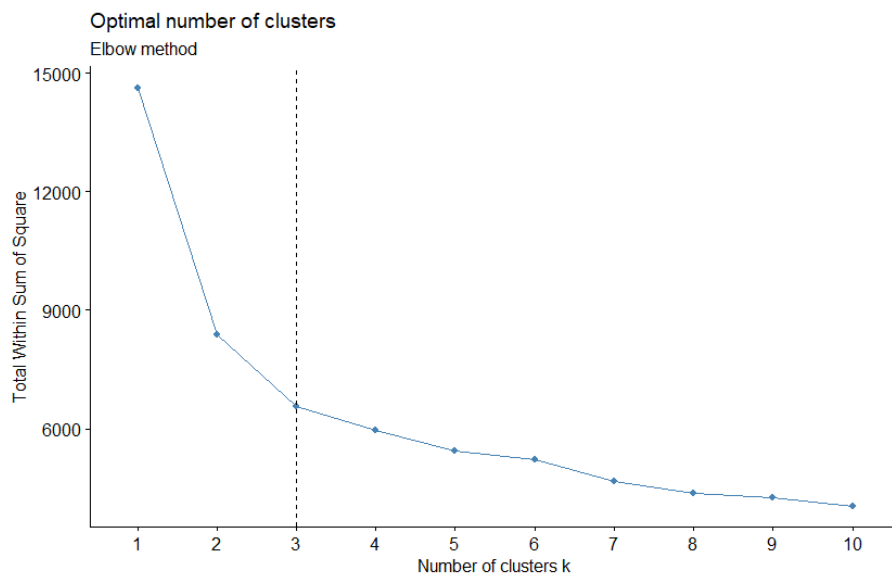
The D index is a graphical method of determining the number of clusters. In the plot of D index, we seek a significant knee (the significant peak in Dindex second differences plot) that corresponds to a significant increase of the value of the measure.



## Elbow Method

Using the Elbow approach, we adjust the number of clusters (K) from 1 to 10. The Elbow method involves calculating the WCSS for different values of K (the number of clusters) and plotting the WCSS against K. The WCSS typically decreases as the number of clusters increases, but there is often a point at which the rate of decrease slows down, resulting in a bend or "elbow" in the plot. This elbow point represents the ideal K value or the optimal number of clusters. The point at which the graph starts to flatten out and become almost parallel to the X-axis indicates that additional clusters are not adding much value in terms of reducing the WCSS, and so the optimal K value can be selected as the number of clusters at this elbow point. It is observed that when  $k=2$  is when the elbow occurs.

```
54 #Elbow Method
55 set.seed(63)
56 elbowmethod_plot<-fviz_nbclust(normalized_data, kmeans,method ="wss") +
57   geom_vline(xintercept = 3, linetype = 2)+
58   labs(subtitle = "Elbow method")|
59   print(elbowmethod_plot)
60
```

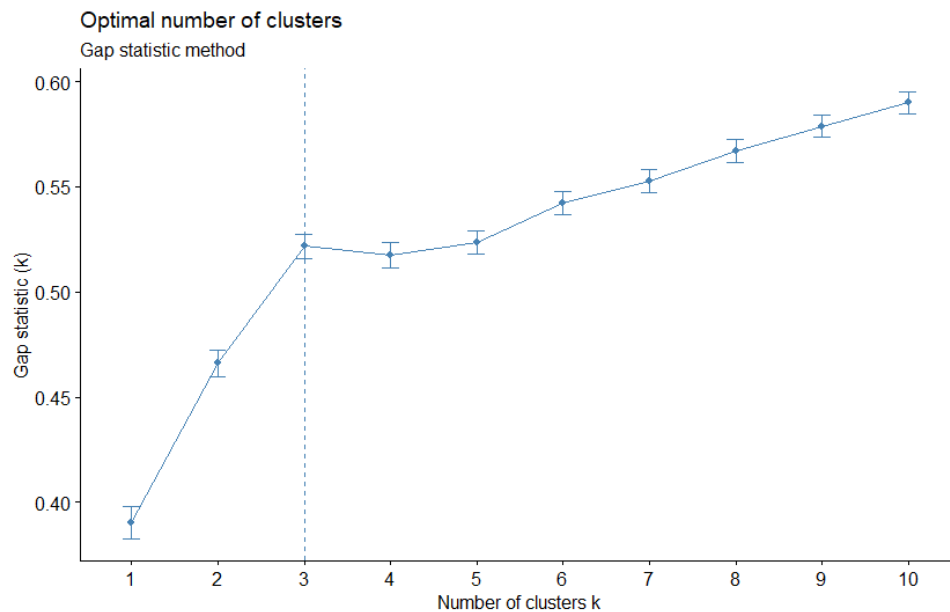


## Gap Statistic

The gap statistic is a clustering evaluation method that can be applied to any clustering approach, including K-means clustering and hierarchical clustering. It compares the total intra-cluster variation for different values of k to their expected values under a null reference distribution of the data. By calculating the difference between the observed and predicted values under the null hypothesis, the gap statistic determines the optimal number of clusters that best fit the data. The value that maximizes this difference indicates that the clustering structure deviates significantly from a uniform distribution of

points. This evaluation is done using R code that compares the observed and predicted values and selects the optimal number of clusters based on the maximum difference.

```
60
61 #Gap Statistics Method
62 set.seed(123)
63 gap_stat<-fviz_nbclust(normalized_data,kmeans, nstart = 25, method = "gap_stat", nboot = 100, iter.max=50)+
64   labs(subtitle = "Gap statistic method")
65 print(gap_stat)
66
```



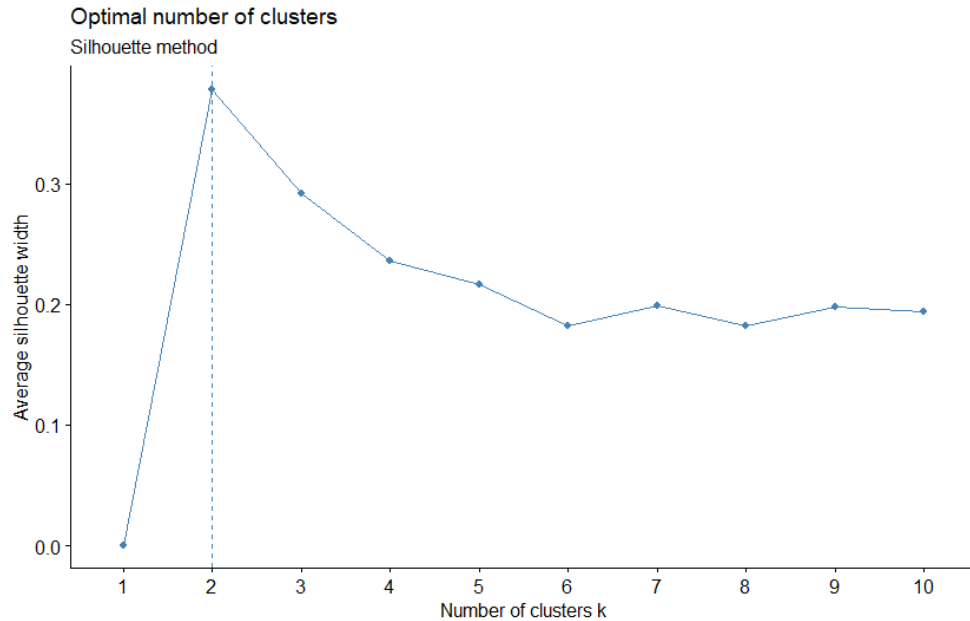
In the graph above we see that the gap stats method takes k=3 as the optimal cluster.

### Silhouette Method

We calculate the silhouette coefficient in the silhouette method. The silhouette coefficient measures how distinct (separation) from other clusters (cohesion) a data point is within a cluster. Then, using the code below, we measure against an average silhouette.

```
67 #Silhouette Method
68 set.seed(467)
69 silhouette_plot<-fviz_nbclust(normalized_data, kmeans, method = "silhouette")+
70   labs(subtitle = "Silhouette method")
71 print(silhouette_plot)
```





The graphs above demonstrate that when there are two clusters, the average silhouettes and their maximum width appear. Therefore, we conclude that the silhouette method chooses 2 as the optimal number of clusters.

Therefore, as a majority rule the optimal number of clusters is when the value of K is equal to 3.

### **c) K-Means Clustering**

To obtain the desired output and perform K-Means Clustering we first define a user defined function which will give the following outputs,

- Number of clusters
- Number of Centers
- Number of points for each cluster
- Within Sum of Squares
- Between Sum of Squares
- BSS / TSS ratio
- Number of iterations
- Visual plot of the clustering
- Internal Metrics for Silhouette method

All the above is defined within a function called “ApplyKMeans” which is displayed below,

```

73 #k- Means Clustering
74 ApplyKMeans <- function(normalized_data,k){
75     Km <- kmeans(normalized_data[, -length(normalized_data)],k)
76     C <- Km$centers
77     S <- Km$size
78     WSS<- Km$withinss
79     BSS <- Km$betweenss
80     TSS<- Km$betweenss/Km$totss
81     I <- Km$iter
82     cluster_plot<-fviz_cluster(Km, data = normalized_data,
83                               palette = c("#AE3FEA", "#00AFBB", "#E7B800", "#aff0e7"),
84                               geom = "point", ellipse.type = "convex", ggtheme = theme_bw(),
85     silhouette_plot <- silhouette(Km$cluster, dist(normalized_data)),
86     print(fviz_silhouette(silhouette_plot))
87 )
88 }
89 return(list(Number_of_clusters = k,
90             Number_of_points_for_each_cluster = S,
91             Number_of_iterations = I,
92             Centers = C,
93             withinss = WSS,
94             Betweenss = BSS,
95             Between_to_total_ratio = TSS,
96             Visual_cluster_plot = cluster_plot)) #internal metrics
97 }
98 ApplyKMeans(normalized_data, 3)

```

K -Means Clustering for k=3

The user defined functions “ApplyKmeans” is applied on the outlier removed and scaled data with k defined as 3 in the code below,

```
ApplyKMeans(normalized_data, 3)
```

The code above will give the visual plot and other outputs as below,

```

> ApplyKMeans(normalized_data, 3)
$Number_of_clusters
[1] 3

$Number_of_points_for_each_cluster
[1] 327 254 233

$Number_of_iterations
[1] 3

$Centers
  Comp      Circ    D.Circ   Rad.Ra Pr.Axis.Ra  Max.L.Ra  Scat.Ra
1 -0.2305862 -0.5316537 -0.2945969  0.000931716  0.3678022 -0.1686474 -0.4506875
2  1.1619572  1.1879128  1.2225756  1.050718567  0.2138098  0.7067637  1.3091738
3 -0.9430706 -0.5488373 -0.9193176 -1.146726125 -0.7492660 -0.5337781 -0.7946580
1  0.3154466 -0.4783570 -0.5001986 -0.4020087 -0.4575242 -0.5532959 -0.6844246
2 -1.2232515  1.3137764  1.1093979  1.2629923  1.3202566  1.0951042 -0.0257191
3  0.8907934 -0.7608432 -0.5073911 -0.8126318 -0.7971448 -0.4172906  0.9885816
  Skew.maxis Kurt.maxis Kurt.Maxis Holl.Ra
1 -0.03569905 -0.02718324  0.77687606  0.6820409
2  0.13978075  0.27448001 -0.02710884  0.1535188
3 -0.10227776 -0.26106869 -1.06074174 -1.1245542

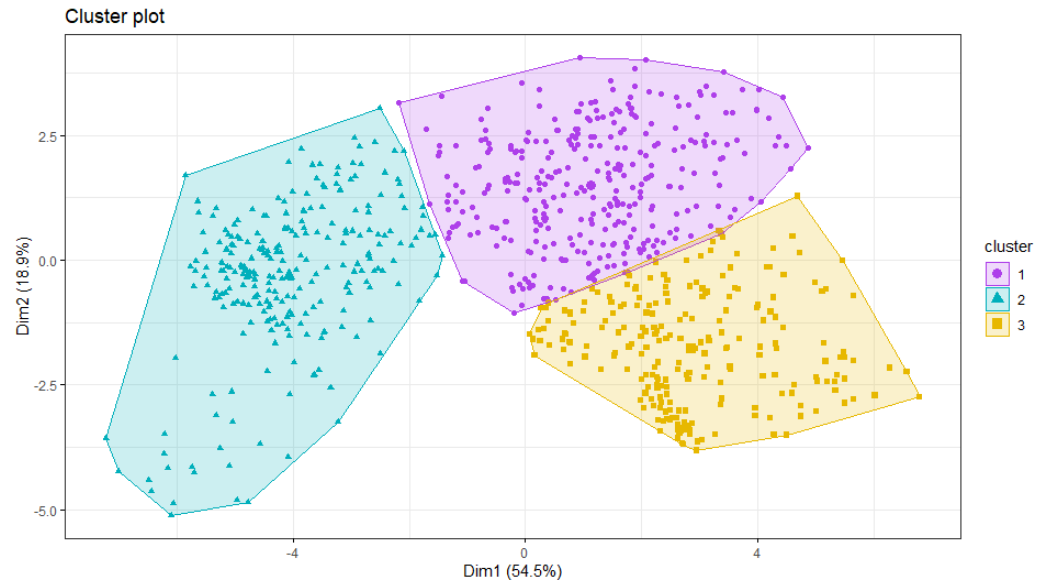
$withinss
[1] 2660.204 2284.200 1618.092

$Betweenss
[1] 8071.503

$Between_to_total_ratio
[1] 0.5515583

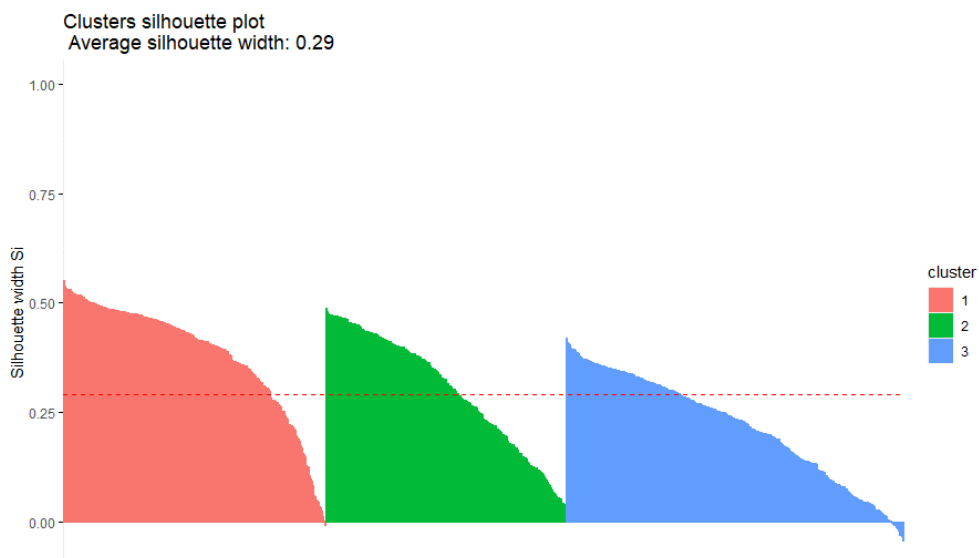
$Visual_cluster_plot

```



It is evident from the above statistics the ratio between the total sum of squares being 0.551 which is a percentile of 55.1%.

#### **d)Average Silhouette Width**



Each data point's silhouette width represents how well it fits into the cluster to which it has been assigned. The average distance between the data point and other data points in its own cluster as well as the average distance between the data point and data points in the cluster that is the closest neighbor are both taken into consideration. When data points are evenly spaced out inside the clusters they are allocated to and relatively far apart from points in other clusters, a bigger silhouette width denotes better clustering quality. The code offers a way to evaluate the separation and compactness of clusters produced by the k-means algorithm through the computation and visualization of the silhouette widths. It aids in assessing the caliber and suitability of the clustering solution found for the provided `normalized_data` and the predetermined number of clusters.

#### Evaluation of the above statistics

For each cluster, the WSS (Within-Cluster Sum of Squares) values are 2660.204, 2284.200, and 1618.092. The WSS calculates how variable each cluster's observations are. The clustering algorithm has done an excellent job of putting comparable data together because lower WSS values show that the observations in each cluster are related to one another.

The value of the BSS (Between-Cluster Sum of Squares) is 8071.503. The BSS calculates the variance among cluster centers. The clustering algorithm did a decent job of separating the clusters because higher BSS values show that the cluster centers are well-separated.

Based on these values, we can see that the clustering algorithm has done a good job of grouping similar observations together and creating distinct clusters.

## 2<sup>nd</sup> Subtask Objectives

### e) PCA

```
104 #PCA
105 library(fpc)
106 get_pca_attr <- function(pca_data){
107   eigenvalue <- pca_data$sdev^2
108   eigenvector <- pca_data$rotation
109   cumulative_score <- cumsum(eigenvalue / sum(eigenvalue))
110
111   print("Eigenvalues:")
112   print(eigenvalue[cumulative_score < 0.92])
113
114   print("Eigenvectors:")
115   print(eigenvector[, cumulative_score < 0.92])
116
117   print("Cumulative scores:")
118   print(cumulative_score[cumulative_score < 0.92])
119
120   return ( cumulative_score)
121 }
122
123
124
125
126
127 findcalinskiIndex <- function(data_frame, k){
128   kmeans <- kmeans(data_frame, k)
129   ch_index <- round(calinhara(normalized_data, kmeans$cluster), digits = 2)
130   cat("Calinski-Harabasz Index:", ch_index, "\n")
131 }
132
133
134 pca_great_data <- prcomp(normalized_data, center = TRUE, scale = FALSE)
135 summary(pca_great_data)
136
137 # Identifies the data sets that are within the cum score threshold of <92%
138 cumulative_score <- get_pca_attr(pca_great_data)
139 pcadataframe <- pca_great_data$x[, cumulative_score < 0.92]
140
141 summary(pcadataframe)
```

Overall, the code runs PCA without scaling the variables on the `normalized_data`. The final PCA object, `pca_great_data`, has details about the main components, including their scores, loadings (rotation matrices), and standard deviations. You can examine the variance explained by each component and determine their importance for additional analysis or dimensionality reduction using the `summary()` function, which is used to display a summary of the PCA results.

```

+ print("Eigenvectors:")
+ print(eigenvector[, cumulative_score < 0.92])
+
+ print("Cumulative scores:")
+ print(cumulative_score[cumulative_score < 0.92])
+
+ # Identifies the data sets that are within the cum score threshold of <92%
+ pcadataframe <- pca_great_data$x[, cumulative_score < 0.92]
+ get_pca_attr(pca_great_data)
+ summary(pcadataframe)
+
+ }
+
+ > findcalinskiIndex <- function(data_frame, k){
+   kmeans <- kmeans(data_frame, k)
+   ch_index <- round(calinhara(normalized_data, kmeans$cluster), digits = 2)
+   cat("Calinski-Harabasz Index:", ch_index, "\n")
+ }
+
+ > pca_great_data <- prcomp(normalized_data, center = TRUE, scale = FALSE)
+ summary(pca_great_data)
Importance of components:
      PC1      PC2      PC3      PC4      PC5      PC6      PC7      PC8      PC9     PC10
Standard deviation  3.1311 1.8429 1.09940 1.06428 0.94502 0.8106 0.5645 0.47447 0.33210 0.27280
Proportion of Variance 0.5447 0.1887 0.06715 0.06293 0.04961 0.0365 0.0177 0.01251 0.00613 0.00413
Cumulative Proportion 0.5447 0.7333 0.80048 0.86341 0.91302 0.9495 0.9672 0.97973 0.98586 0.99000
      PC11      PC12      PC13      PC14      PC15      PC16      PC17      PC18
Standard deviation  0.24112 0.20027 0.16527 0.14536 0.12373 0.10845 0.07728 0.01861
Proportion of Variance 0.00323 0.00223 0.00152 0.00117 0.00085 0.00065 0.00033 0.00002
Cumulative Proportion 0.99323 0.99545 0.99697 0.99815 0.99900 0.99965 0.99998 1.00000
>

```

Therefore, the optimal number of PCs would be 5.

The discussion for choosing a specific number of PCs depends on the specific analysis and dataset. Here are a few considerations:

**Variance Explained:** The cumulative score shows how much of the variance can be accounted for by the PCs. By choosing a cumulative score criterion of 92%, you may be certain that the PCs you've picked are able to account for a sizable amount of the dataset's variability.

**Reducing Dimensionality:** PCA can be used to reduce dimensions. You strike balance between capturing enough variance and minimizing the dimensionality of the dataset by choosing a sufficient number of PCs that offer a cumulative score of 92%.

**Interpretability:** Depending on the application, a lower number of PCs that are simpler to interpret and analyze may be preferable. You can prioritize interpretability while still capturing a sizable amount of variance by selecting the number of PCs based on a cumulative score threshold.

#### f) Determining the Number of Clusters

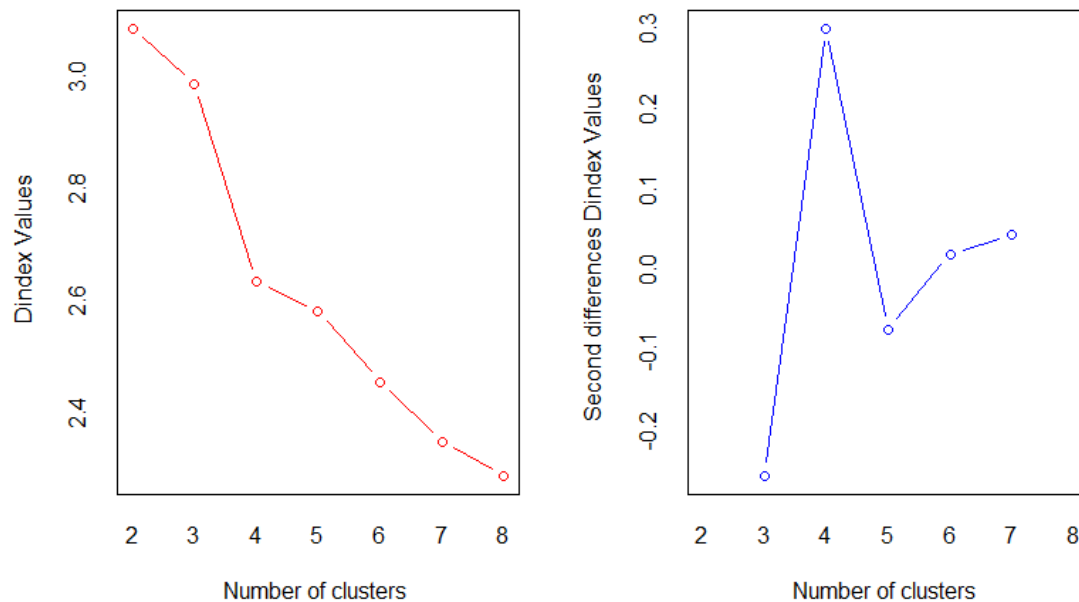
The function `spotClusters()` provides a convenient way to apply different clustering methods (NbClust, Elbow Method, Gap Statistics, and Silhouette Method) and visualize their results to determine the optimal number of clusters for the given `normalized_data`. The function below interprets this,

```

144 #To determine the optimal number of clusters
145 spotclusters <- function(data_frame){
146   #NbClust Method
147   set.seed(1445)
148   num_clusters <- NbClust(normalized_data,distance="euclidean",min.nc = 2,max.nc =8,method = "kmeans",index="all")
149   table(num_clusters$Best.nc)
150
151   #Elbow Method
152   set.seed(63)
153   elbowmethod_plot<-fviz_nbclust(normalized_data, kmeans,method ="wss") +
154     geom_vline(xintercept = 3, linetype = 2)+
155     labs(subtitle = "Elbow method")
156   print(elbowmethod_plot)
157
158   #Gap Statistics Method
159   set.seed(123)
160   gap_stat<-fviz_nbclust(normalized_data,kmeans, nstart = 25, method = "gap_stat", nboot = 100, iter.max=50)+
161     labs(subtitle = "Gap statistic method")
162   print(gap_stat)
163   #Silhouette Method
164   set.seed(467)
165   silhoutte_plot<-fviz_nbclust(normalized_data, kmeans, method ="silhouette")+
166     labs(subtitle = "Silhouette method")
167   print(silhoutte_plot)
168
169 } #Function defined by the user
170 spotclusters()
171

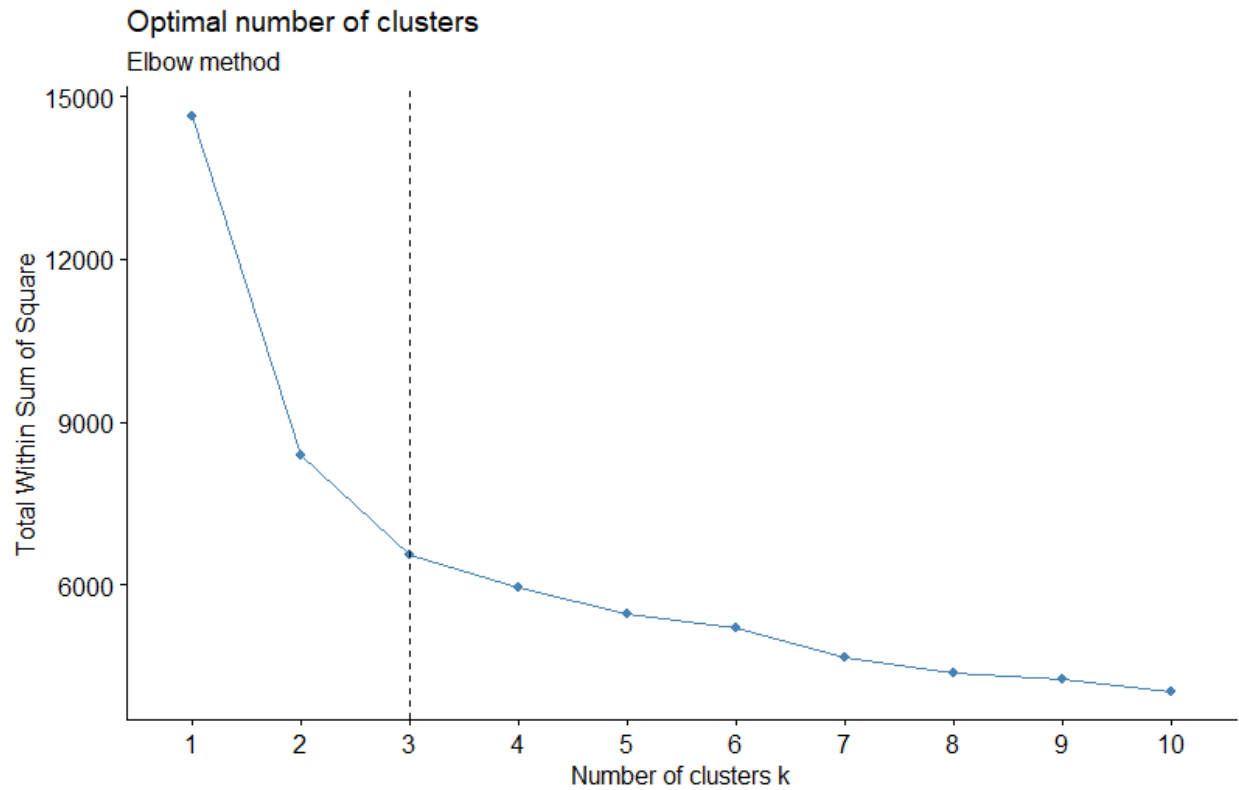
```

### Nbclust Method



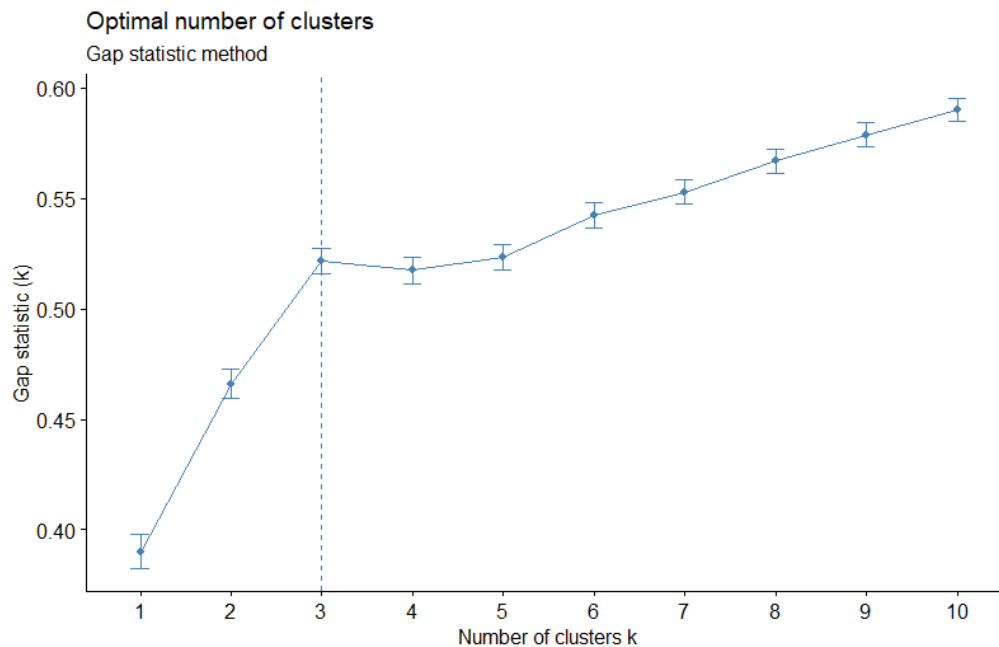
This has been done in a similar manner to the clustering on subtask 1 . The optimal number of  $k=4$ .

### Elbow Method



This has been done in a similar manner to the clustering on subtask 1 . The optimal number of  $k=3$ .

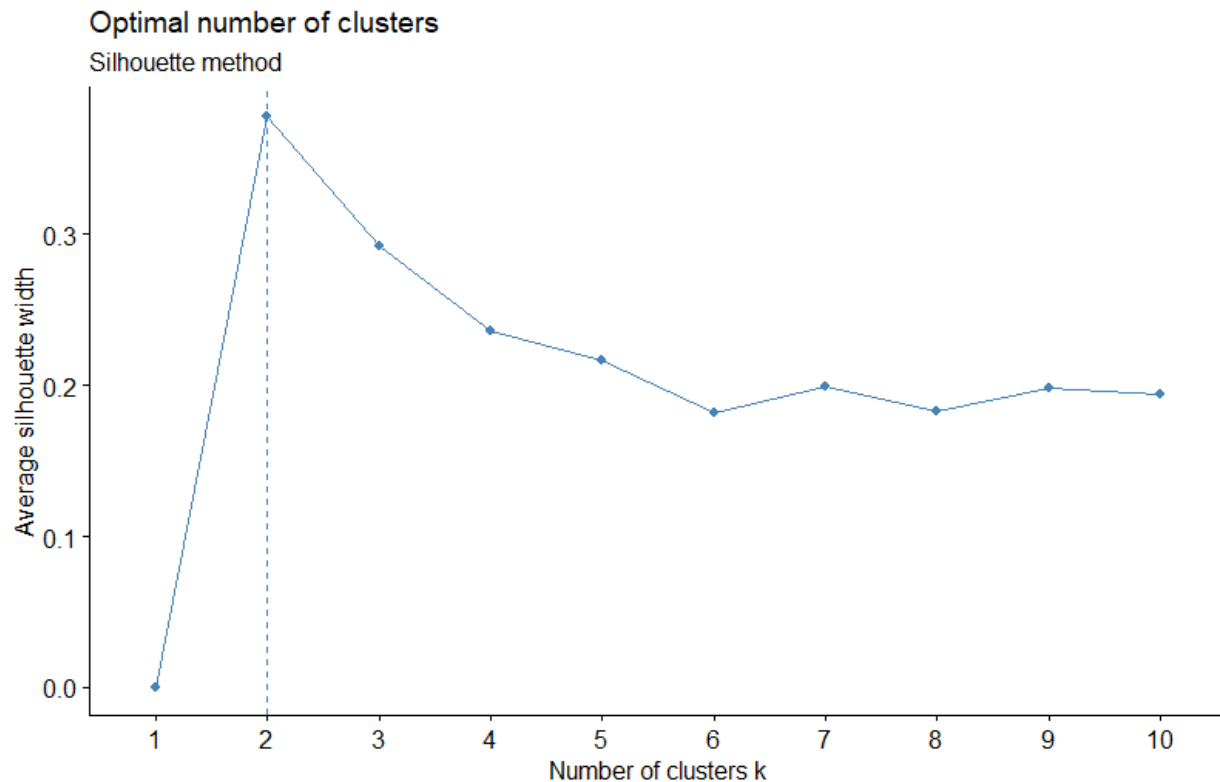
### Gap Statistics Method





This above has been done in a similar manner to the clustering on subtask 1 . The optimal number of clusters is  $k= 3$ .

#### Silhouette Method



This has been done in a similar manner to the clustering on subtask 1 . The optimal number of clusters is  $k= 2$ .

Therefore, based on majority rule the optimal number of clusters is when  $k=3$ .

#### **f) K-Means Clustering**

To obtain the desired output and perform K-Means Clustering we use the user defined function which will give the following outputs,

- Number of clusters
- Number of Centers
- Number of points for each cluster
- Within Sum of Squares
- Between Sum of Squares

- BSS / TSS ratio
- Number of iterations
- Visual plot of the clustering
- Internal Metrics for Silhouette method

All the above is defined within a function called “ApplyKMeans” which is displayed below,

```

171 #kmeansclustering
172 ApplyKMeans <- function(pcadataframe,k){
173   Km <- kmeans(pcadataframe[, -length(pcadataframe)],k)
174   C <- Km$centers
175   S <- Km$size
176   WSS<- Km$withinss
177   BSS <- Km$betweenss
178   TSS<- Km$betweenss/Km$totss
179   I <- Km$iter
180   cluster_plot<-fviz_cluster(Km, data = pcadataframe,
181     palette = c("#AE3FEA", "#00AFBB", "#E7B800", "#aff0e7"),
182     geom = "point", ellipse.type = "convex", ggtheme = theme_bw(),
183     silhouette_plot <- silhouette(Km$cluster, dist(pcadataframe)),
184     print(fviz_silhouette(silhouette_plot))
185   )
186 }
187 return(list(Number_of_clusters = k,
188   Number_of_points_for_each_cluster = S,
189   Number_of_iterations = I,
190   Centers = C,
191   withinss = WSS,
192   Betweenss = BSS,
193   Between_to_total_ratio = TSS,
194   Visual_cluster_plot = cluster_plot)) #internal metrics
195 }
196 ApplyKMeans(pcadataframe, 3)
197

```

### K -Means Clustering for k=3

The user defined functions “ApplyKmeans” is applied on the outlier removed and scaled data with k defined as 3 in the code below,

```

196 ApplyKMeans(pcadataframe, 3)
197

```

g)

```
> #Gap Statistics as most favored clusters
> #k=3
> km3 <- kmeans(pcadataframe,3)
> km3
K-means clustering with 3 clusters of sizes 254, 322, 238

cluster means:
      PC1      PC2      PC3      PC4      PC5
1 -4.073853 -0.316366  0.10134110 -0.07732116 -0.06358301
2  1.113302  1.530437 -0.06219545  0.15891630 -0.08234986
3  2.841494 -1.732956 -0.02400715 -0.13248519  0.17927202

clustering vector:
 [1] 2 2 1 2 1 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
[32] 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 2 3
[63] 3 1 2 1 1 1 2 3 2 1 2 3 1 3 1 2 3 2 1 2 3 2 3 1 2 1 2 3 1 3 3
[94] 1 3 2 2 3 1 1 1 3 3 2 2 2 3 3 2 1 1 3 2 3 3 2 3 2 3 2 1 1 2 3
[125] 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 3 1 1
[156] 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
[187] 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
[218] 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
[249] 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
[280] 2 1 2 1 3 2 2 1 2 2 2 3 2 1 1 1 1 1 3 2 1 3 3 2 3 1 1 1 3 1
[311] 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
[342] 2 1 2 1 1 1 2 3 1 3 3 3 2 2 2 2 2 3 1 1 3 3 1 3 3 1 2 2 2 3
[373] 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
[404] 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
[435] 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
[466] 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
[497] 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
[528] 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
[559] 1 1 1 2 1 3 2 1 3 3 3 2 3 2 2 2 2 2 2 1 2 2 1 2 2 3 3 1 3 3
[590] 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
[621] 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
[652] 1 3 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
[683] 2 1 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 1 2
[714] 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
[745] 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
[776] 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
[807] 2 2 2 2 2 1 2 3

within cluster sum of squares by cluster:
[1] 1834.001 2111.041 1356.928
(between_SS / total_SS = 60.3 %)

Available components:
[1] "cluster"      "centers"      "totss"        "withinss"
[5] "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

```
> WSS_G <- km3$withinss
> WSS_G
[1] 1834.001 2111.041 1356.928
>
> BSS_G <- km3$betweenss
> BSS_G
[1] 8059.236
>
> TSS_G <- km3$totss
> TSS_G
[1] 13361.21
>
> BSS_G_TSS_G_ratio <- BSS_G/TSS_G
> BSS_G_TSS_G_ratio
[1] 0.6031817
```

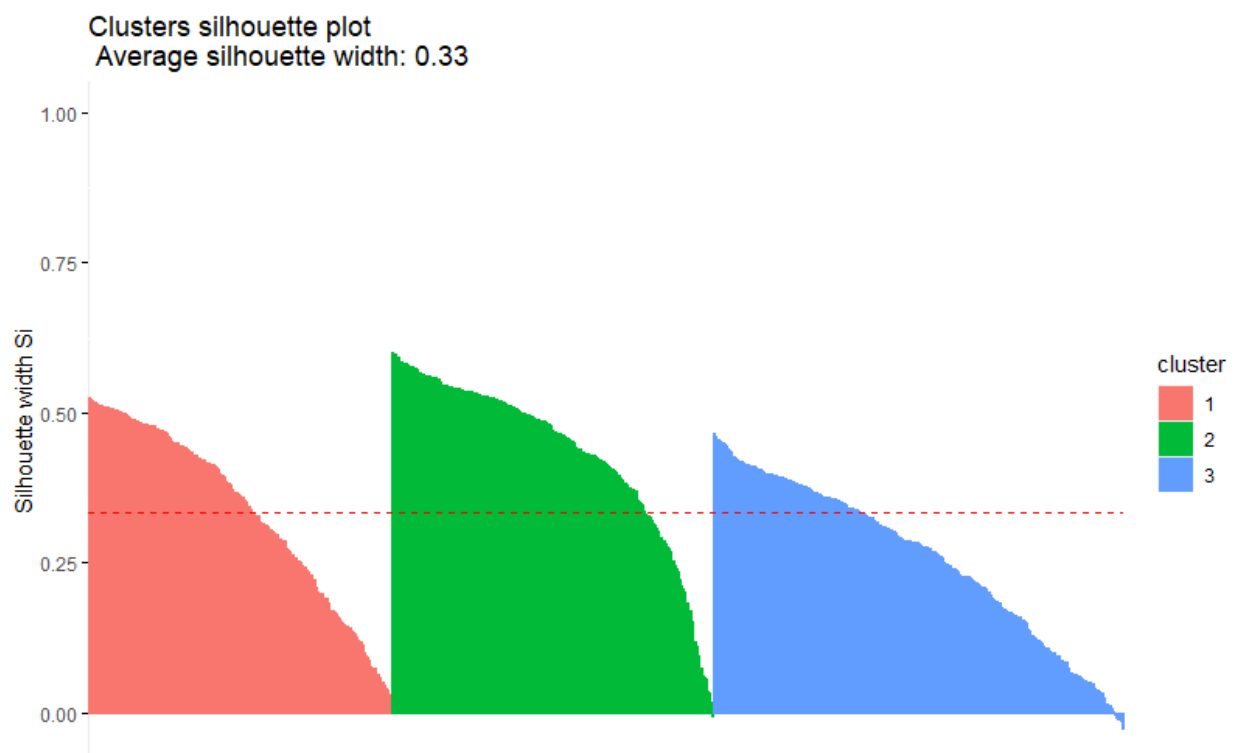
The three clusters' respective WSS values are 1834.001, 2111.041, and 1356.928. WSS is a measure of how far off each data point is from the centroid of the cluster to which it belongs. A lower WSS value suggests better clustering compactness since the data points within each cluster are located closer to their centroid.

There is an 8059.236 BSS value. BSS stands for the sum of the squared distances between the centroid of each cluster and the centroid of the entire dataset. BSS calculates the distance or dispersion between several clusters. The cluster centroids are well separated from one another when the BSS value is higher.

TSS is a metric for the dataset's overall variance. TSS represents the sum of squared distances between each data point and the overall centroid of the dataset. TSS is a measure of the total variance in the dataset.

TSS represents the sum of squared distances between each data point and the overall centroid of the dataset. TSS is a measure of the total variance in the dataset. The formula yields a BSS/TSS ratio of 0.6031817. This ratio shows how much of the overall variance can be accounted for by between-cluster sums of squares (BSS) in comparison to total sums of squares (TSS). Better separation and higher clustering quality are indicated by a higher BSS/TSS ratio, which means that variations across clusters rather than within clusters account for a larger share of the overall variance.

#### **h) Average Silhouette Width**



Similar to subtask 1 when the average silhouette width is higher it indicates better defined and well separated clusters, while a lower values suggest overlapping or poorly separated clusters.

### **i)Calinski-Harabasz Index**

Code for internal evaluation metrics using Calinski Harabasz Index is show below along with the result,

```
215 #Clinski Index - Internal Evaluation Metrics|  
216 findCalinskiIndex(pcadataframe,3)  
217
```

```
> findCalinskiIndex(pcadataframe,3)  
Calinski-Harabasz Index: 498.73
```

Better-defined and well-separated clusters are indicated by a higher Calinski-Harabasz index. It implies that the between-cluster dispersion—the distance between various clusters—is greater than the intra-cluster dispersion, which refers to the degree of compactness inside each cluster. This means that the clusters in the feature space are distinct and clearly separated from one another. The optimal clustering solution with the most distinct and compact clusters is indicated by the greatest index value, which is favored.

## **Neural Networks**

### **1<sup>st</sup> Subtask Objectives**

#### **a)**

##### **Input Variable Analysis for electricity load forecasting**

Multilayer Perceptron (MLP) models are frequently used in power load forecasting to produce precise predictions. In order to get the necessary data for forecasting, the definition of the input vector is essential.

1. Autoregressive (AR) Approach  
Incorporating lagged values of the load variable as input features is the autoregressive (AR) technique. This strategy has been applied in numerous research, including hybrid models that combine ARIMA with MLP (Hong et al., 2004) and MLP models based on ARIMA (Kim et al., 2002). To improve forecasting accuracy, lagged load values capture temporal interdependence and historical patterns.
2. Weather Variables  
Demand for electricity is greatly influenced by the weather. So, adding weather-related inputs can help with load predictions. Common weather factors include temperature, humidity, wind speed, and sun radiation (Daz et al., 2002; AlRashidi and El-Naggar, 2011). These factors are gathered from nearby weather stations or forecasts made by meteorologists.
3. Calendar Variables  
Calendar variables take into consideration temporal elements like the day of the week, the month, the holidays, and special occasions. The use of calendar variables makes it easier to identify recurring load patterns influenced by particular temporal elements. Studies (Hong et al., 2004; Hong and Fan, 2006) have taken into account elements including day type (weekday vs. weekend), day of the week, and month of the year.
4. Time – Related Features  
Time-related characteristics record daily load cycles and diurnal fluctuations. These patterns are recorded using time intervals, minute of the hour, and hour of the day (Taylor, 2003; Khosravi et al., 2011). These characteristics aid the model in determining the load behavior throughout the day.
5. Exogenous factors that influence power usage can be added as lagged features in addition to load and weather variables. Energy prices, population data, economic indicators, and other significant factors can all be considered (Hong and Fan, 2006; Hong and Fan, 2007). These factors account for outside influences on the demand for power.

## Preparation of Data

### Usage of Libraries

For task 2 the following libraries were used,

- Tidyverse – For better workflow and smoother productivity
- Neuralnet – To train the neural networks using back propagation.
- Readxl – Read data from xlsx files to the workspace.
- Dplyr - Provide a concise and expressive syntax that allows for easy chaining of operations, making it convenient for data manipulation tasks.

The code for the importing of libraries is below,

```
1 library(tidyverse)
2 library(neuralnet)
3 library(dplyr)
4 library(readxl)
```

The dataset was imported into the workspace using the readxl library and a was assigned a variable called "uow\_load".

```
6 #loading the dataset
7 uow_load <- read_excel("uow_consumption.xlsx")
8 uow_load
```

As seen by the above we can verify the data was imported correctly through the environment data tab which gives us a prompt of 470 observations

uow_load	470 obs. of 4 variables
----------	-------------------------

```
date          0.75 0.7916666666666663
<dtm>         <dbl>
1 2018-01-01 00:00:00 38.9      38.9
2 2018-01-02 00:00:00 42.3      41.9
3 2018-01-03 00:00:00 40.8      40.5
4 2018-01-04 00:00:00 42.3      41.9
5 2018-01-05 00:00:00 44        44.1
6 2018-01-06 00:00:00 45.6      44.5
7 2018-01-07 00:00:00 40.9      41.2
8 2018-01-08 00:00:00 44.6      44.3
9 2018-01-09 00:00:00 41.9      42.9
10 2018-01-10 00:00:00 44.9      44.4
# i 460 more rows
```

The column names were replaced using the following code,

```
10 # Replace the column names with new relevant column names
11 colnames(uow_load) <- c("Date", "18", "19", "20")
12 view(head(uow_load))
```

The result of changing column names,



	Date	18	19	20
1	2018-01-01	38.9	38.9	38.9
2	2018-01-02	42.3	41.9	41.9
3	2018-01-03	40.8	40.5	40.7
4	2018-01-04	42.3	41.9	41.9
5	2018-01-05	44.0	44.1	44.0
6	2018-01-06	45.6	44.5	44.3

Changing column names helps as it increases the readability and interpretability of the data set. The content and function of each column becomes clearer when the column titles are meaningful. When completing further data analysis or modeling activities, it adds context and clarity. Meaningful column names make it easier to locate and refer to certain dataset variables.

Construct an input/output matrix (I/O) for the MLP training/testing

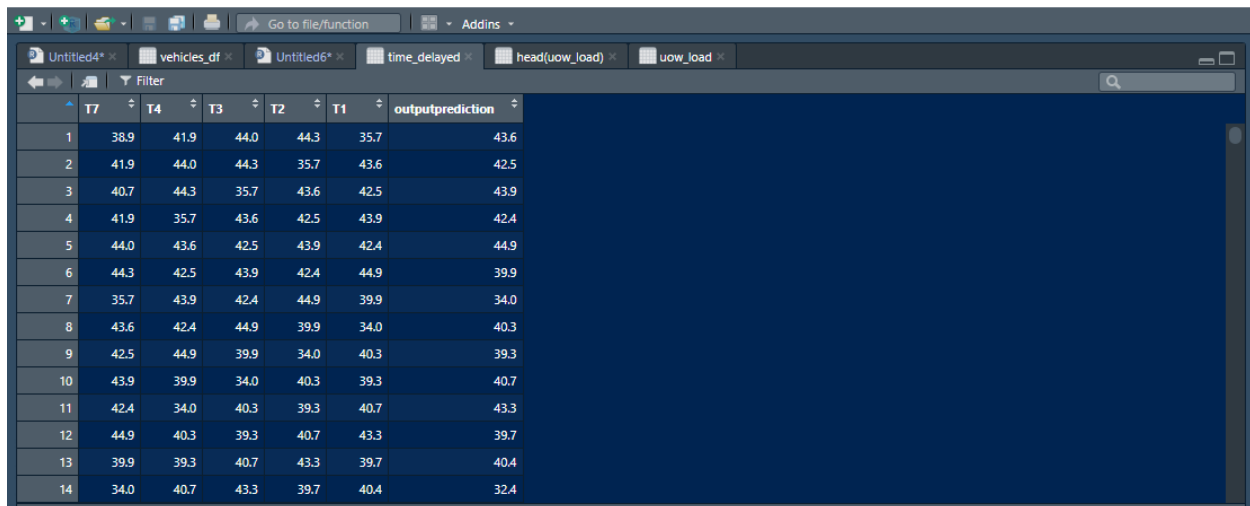
```
14 #creating time-delayed input variables along with I/O Matrix|
15 time_delayed <- bind_cols(
16   T7 = lag(uow_load$'20', 7),
17   T4 = lag(uow_load$'20', 4),
18   T3 = lag(uow_load$'20', 3),
19   T2 = lag(uow_load$'20', 2),
20   T1 = lag(uow_load$'20', 1),
21   outputprediction = lag(uow_load$'20', 0)
22 )
23 time_delayed
```

The snippet of code is used to provide time-delayed input variables for estimating power load. It generates a new data frame called time\_delayed that contains lag values for the "20" column from the uow\_load data frame, which is used to represent the electrical load.

The purpose of the lag() function is to create time-shifted or lagged versions of a variable, which can be useful for time series analysis, forecasting, or creating lagged features in predictive modeling.



The above function provides the result,



	T7	T4	T3	T2	T1	outputprediction
1	38.9	41.9	44.0	44.3	35.7	43.6
2	41.9	44.0	44.3	35.7	43.6	42.5
3	40.7	44.3	35.7	43.6	42.5	43.9
4	41.9	35.7	43.6	42.5	43.9	42.4
5	44.0	43.6	42.5	43.9	42.4	44.9
6	44.3	42.5	43.9	42.4	44.9	39.9
7	35.7	43.9	42.4	44.9	39.9	34.0
8	43.6	42.4	44.9	39.9	34.0	40.3
9	42.5	44.9	39.9	34.0	40.3	39.3
10	43.9	39.9	34.0	40.3	39.3	40.7
11	42.4	34.0	40.3	39.3	40.7	43.3
12	44.9	40.3	39.3	40.7	43.3	39.7
13	39.9	39.3	40.7	43.3	39.7	40.4
14	34.0	40.7	43.3	39.7	40.4	32.4

### Removal of outliers to further analyze data

```
25 # Remove rows with missing values
26 time_delayed <- na.omit(time_delayed)
27 time_delayed
```

The function `na.omit()` is a built-in function in R that removes rows with any missing or NA values from a data frame or matrix. In this case, it is applied to the `time_delayed` data frame.

The initial `uow_load` data set had 470 observations and since the outliers have been removed it contains 463 observations.

```
time_delayed      463 obs. of 6 variables
uow_load          470 obs. of 4 variables
```

To guarantee that the dataset utilized for analysis or modeling is comprehensive and devoid of any missing data, rows with missing values should be removed. Missing numbers can happen for a few reasons, such as inaccurate data collection or insufficient measurements. You may make sure the remaining data is appropriate for further analysis or modeling activities by removing rows with missing values.

```

29 # Construction of different time-delayed input vectors and related i/o matrices.
30 t_1 <- cbind(time_delayed$t1, time_delayed$outputprediction)
31 colnames(t_1) <- c("Input", "Output")
32
33 t_2 <- cbind(time_delayed$t1, time_delayed$t2, time_delayed$outputprediction)
34 colnames(t_2) <- c("Input_1", "Input_2", "Output")
35
36 t_3 <- cbind(time_delayed$t1, time_delayed$t2, time_delayed$t3, time_delayed$outputprediction)
37 colnames(t_3) <- c("Input_1", "Input_2", "Input_3", "Output")
38
39 t_4 <- cbind(time_delayed$t1, time_delayed$t2, time_delayed$t3, time_delayed$t4, time_delayed$outputprediction)
40 colnames(t_4) <- c("Input_1", "Input_2", "Input_3", "Input_4", "Output")
41
42 t_5 <- cbind(time_delayed$t1, time_delayed$t2, time_delayed$t3, time_delayed$t4, time_delayed$t5, time_delayed$outputprediction)
43 colnames(t_5) <- c("Input_1", "Input_2", "Input_3", "Input_4", "Input_5", "Output")

```

We may experiment with different combinations of lagged variables as inputs in our MLP model by constructing these various time-delayed input vectors and matrices. With varied amounts of time delay, each matrix reflects a unique set of input characteristics. The MLP model for forecasting energy load may then be trained and tested using these matrices as input data.

### c) Normalization

The error must be calculated from the inputs and starting weights, which must fall within the same range, in order to use Back Propagation or other optimization techniques. Therefore, it is necessary to prevent wide variances in the input data sets (features). To do this, normalization is used. Additionally, the network converges more quickly, "overrating" is avoided, and the possibility of becoming trapped in local optima is reduced after the inputs have been normalized.

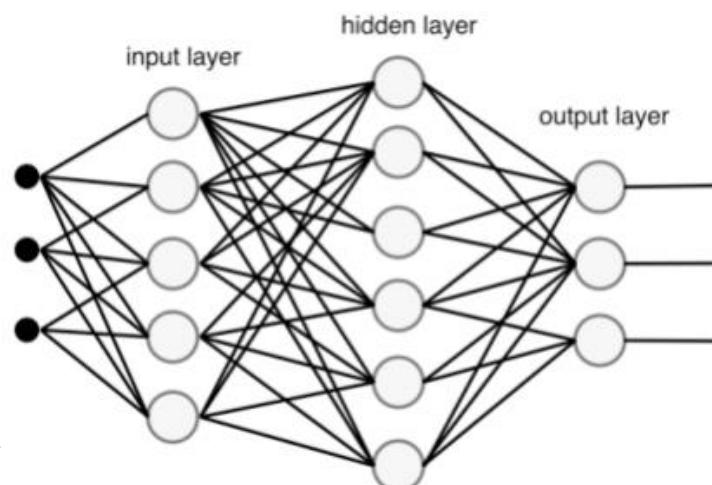
This is done by a user defined function below,

```

45 #Defining the normalization function (Min-Max Normalization)
46 norma <- function(x) {
47   return ((x - min(x)) / (max(x)-min(x)))
48 }
49
50 #Normalizing the I/O Matrices
51 normat_1 <- norma(t_1)
52 normat_2 <- norma(t_2)
53 normat_3 <- norma(t_3)
54 normat_4 <- norma(t_4)
55 normat_5 <- norma(t_5)

```

To better explain why it important to normalize your values we will use this MLP,



It would need more than 100 parameters to successfully finish training the model, as we shall see if we attempt to determine the number of trainable parameters. Therefore, we must utilize the normalized value in order to achieve excellent accuracy while utilizing a minimal amount of processing power. When scaling data into a specific range, normalizing the data has little impact on the quality of the data. Standardizing the data also promotes quicker learning and MLP convergence.

MLP networks' input characteristics frequently reflect various physical values measured in various units. By bringing all the features to a single scale through normalization, all unit-related inconsistencies are removed. This prevents the network from being impacted by the arbitrary measurement units and lets it to concentrate on discovering the underlying patterns and correlations in the data.

Backpropagation, the act of adjusting the network's weights based on the error signal, can assist stabilize the gradients. significant values in some features can produce significant gradients when the input data is not normalized, which can result in unstable learning and delayed convergence. The gradients are kept within a tolerable range by normalizing the input data, resulting in more steady and effective learning.

### **Training Different Inputs & NN Configurations (AR Approach)**

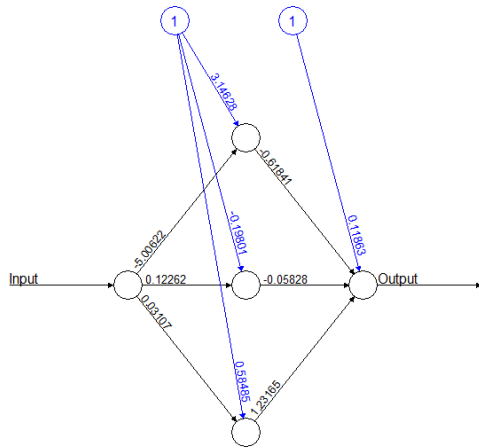
First, we define the training set and testing set for each I/O matrix above,

```
56
57 #define the training sets and testing sets for each I/O matrix
58 Train_t_1 <- normat_1[1:380,]
59 Test_t_1 <- normat_1[381: nrow(normat_1),]
60
61 Train_t_2 <- normat_2[1:380,]
62 Test_t_2 <- normat_2[381: nrow(normat_2),]
63
64 Train_t_3 <- normat_3[1:380,]
65 Test_t_3 <- normat_3[381: nrow(normat_3),]
66
67 Train_t_4 <- normat_4[1:380,]
68 Test_t_4 <- normat_4[381: nrow(normat_4),]
69
70 Train_t_5 <- normat_5[1:380,]
71 Test_t_5 <- normat_5[381: nrow(normat_5),]
72
```

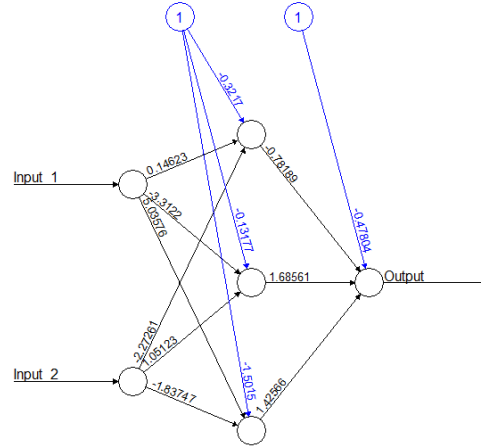
### **Input Vector t-5**

```
73 # Training the Neural network for normat_1
74 # One hidden layer neural networks version #1
75
76 T_1_NN1 <- neuralnet(Output ~ Input, data = Train_t_1, hidden = 3, linear.output = TRUE)
77 plot(T_1_NN1)
78
79 T_2_NN1 <- neuralnet(Output ~ Input_1 + Input_2, data = Train_t_2, hidden = 3, linear.output = TRUE)
80 plot(T_2_NN1)
81
82 T_3_NN1 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3, data = Train_t_3, hidden = 3, linear.output = TRUE)
83 plot(T_3_NN1)
84
85 T_4_NN1 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4, data = Train_t_4, hidden = 3, linear.output = TRUE)
86 plot(T_4_NN1)
87
88 T_5_NN1 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4 + Input_5, data = Train_t_5, hidden = 3, linear.output = TRUE)
89 plot(T_5_NN1)
90
91
92
93
```

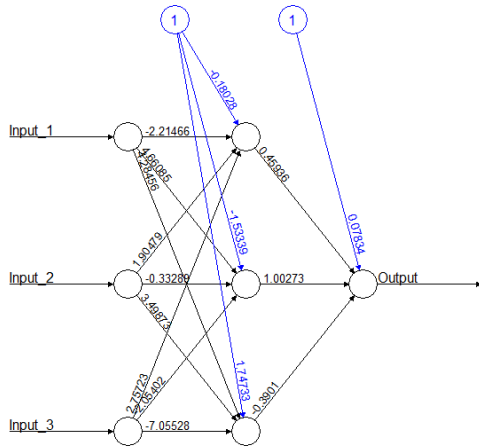
This neural network consists of 5 input vectors with 1 hidden layer that consists of 3 neurons.



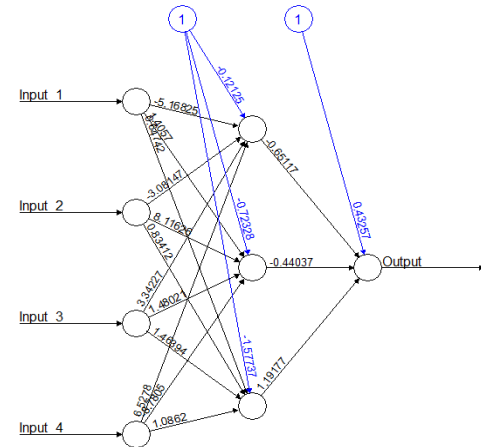
Error: 3.902397 Steps: 1663



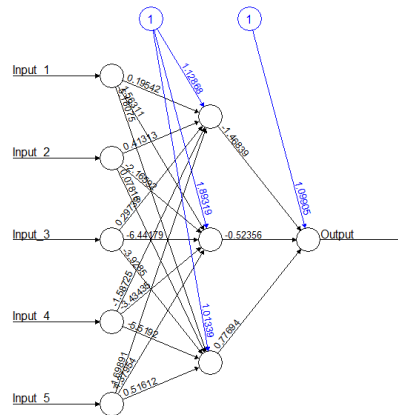
Error: 3.785027 Steps: 10049



Error: 3.618255 Steps: 2132



Error: 3.36829 Steps: 6696



Error: 2.818786 Steps: 9421

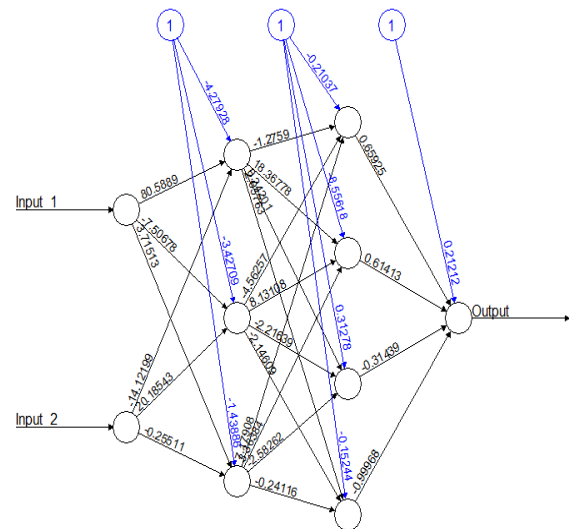
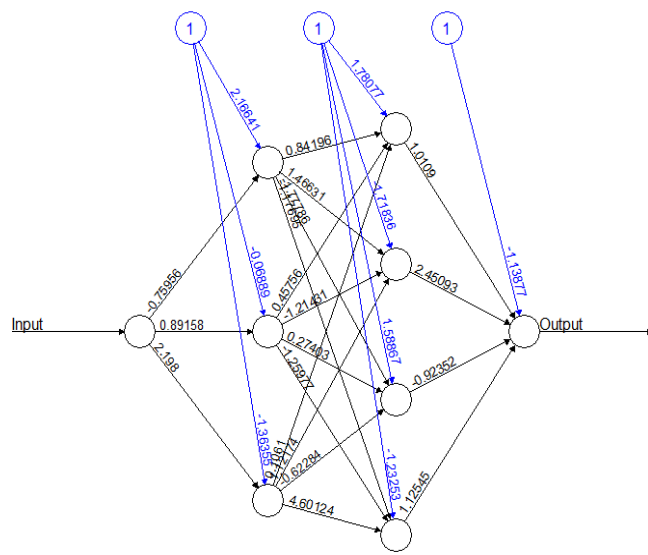
## Input Vector t-5

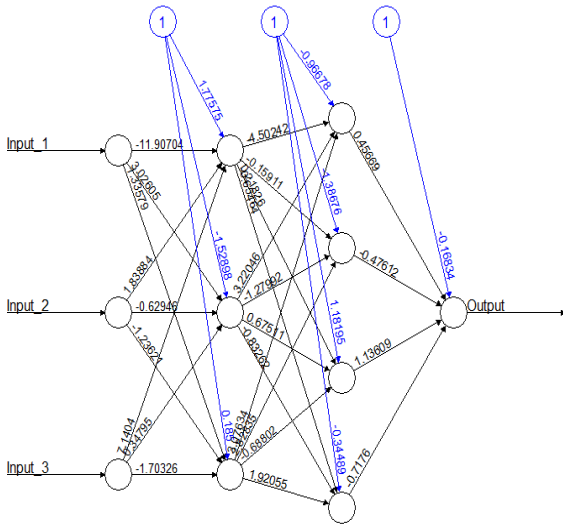
```

91
92 # Two hidden layer Neural networks
93
94 T_1_NN2 <- neuralnet(Output ~ Input, data = Train_t_1, hidden = c(3,4), linear.output = TRUE)
95 plot(T_1_NN2)
96
97 T_2_NN2 <- neuralnet(Output ~ Input_1 + Input_2, data = Train_t_2, hidden = c(3,4), linear.output = TRUE)
98 plot(T_2_NN2)
99
100 T_3_NN2 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3, data = Train_t_3, hidden = c(3,4), linear.output = TRUE)
101 plot(T_3_NN2)
102
103 T_4_NN2 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4, data = Train_t_4, hidden = c(3,4), linear.output = TRUE)
104 plot(T_4_NN2)
105
106 T_5_NN2 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4 + Input_5, data = Train_t_5, hidden = c(3,4), linear.output = TRUE)
107 plot(T_5_NN2)
108

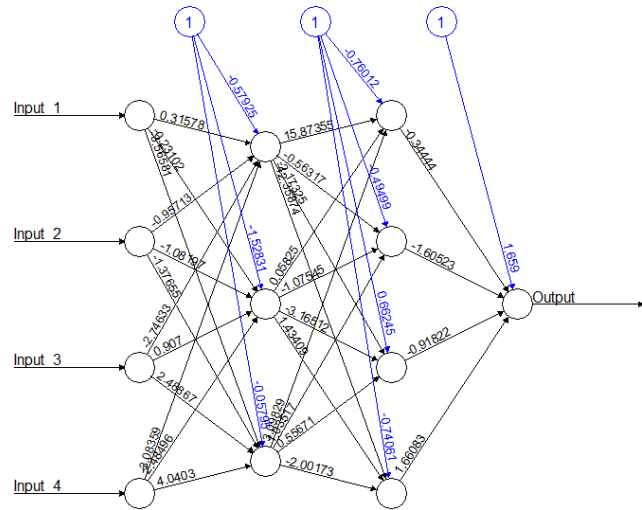
```

This neural network consists of 5 input vectors with 2 hidden layers that consists of 3,4 neurons respectively.

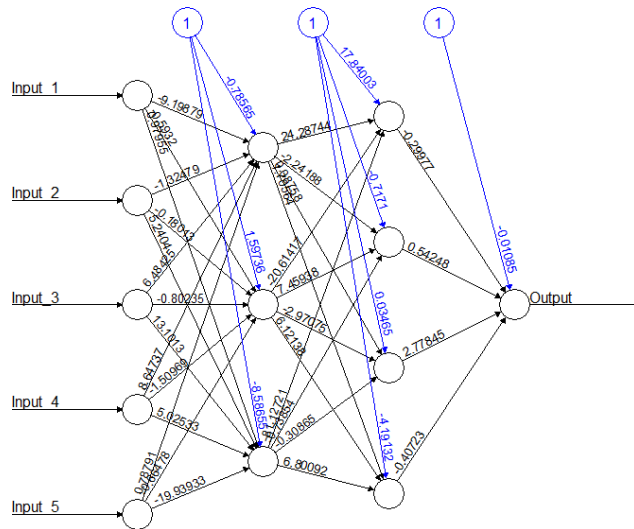




Error: 3.638188 Steps: 1089



Error: 3.464764 Steps: 4302



Error: 2.547519 Steps: 10378

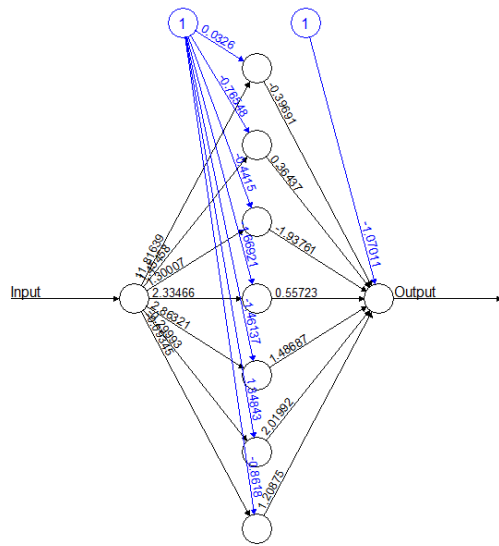
## Input Vector t-5

```

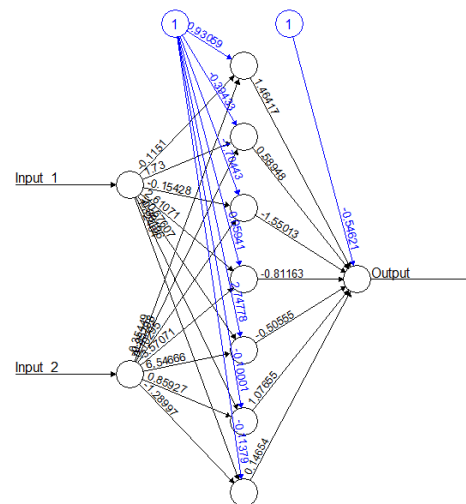
109 # one hidden layer neural networks version #2
110
111 T_1_NN3 <- neuralnet(Output ~ Input, data = Train_t_1, hidden = 7, linear.output = TRUE)
112 plot(T_1_NN3)
113
114 T_2_NN3 <- neuralnet(Output ~ Input_1 + Input_2, data = Train_t_2, hidden = 7, linear.output = TRUE)
115 plot(T_2_NN3)
116
117 T_3_NN3 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3, data = Train_t_3, hidden = 7, linear.output = TRUE)
118 plot(T_3_NN3)
119
120 T_4_NN3 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4, data = Train_t_4, hidden = 7, linear.output = TRUE)
121 plot(T_4_NN3)
122
123 T_5_NN3 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4 + Input_5, data = Train_t_5, hidden = 7, linear.output = TRUE)
124 plot(T_5_NN3)
125

```

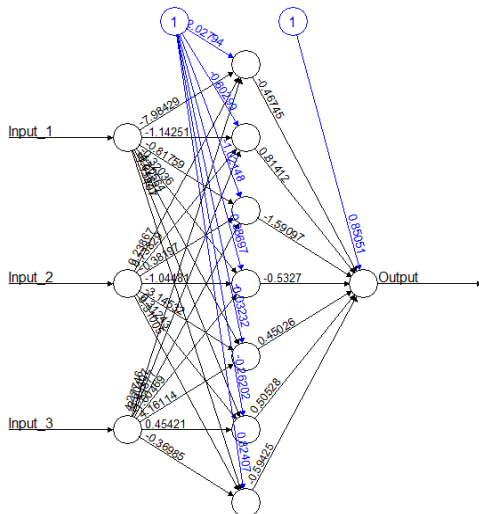
This neural network consists of 5 input vectors with 1 hidden layer that consists of 7 neurons.



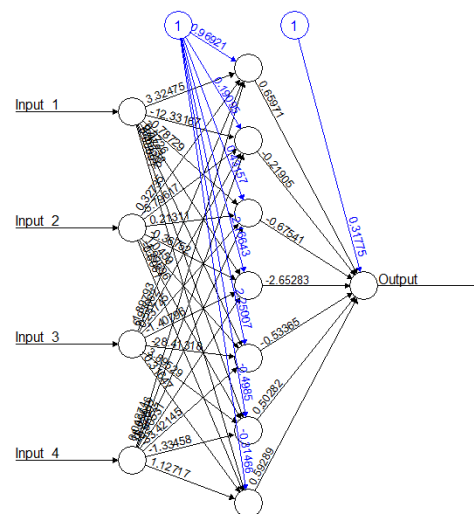
Error: 3.875454 Steps: 3631



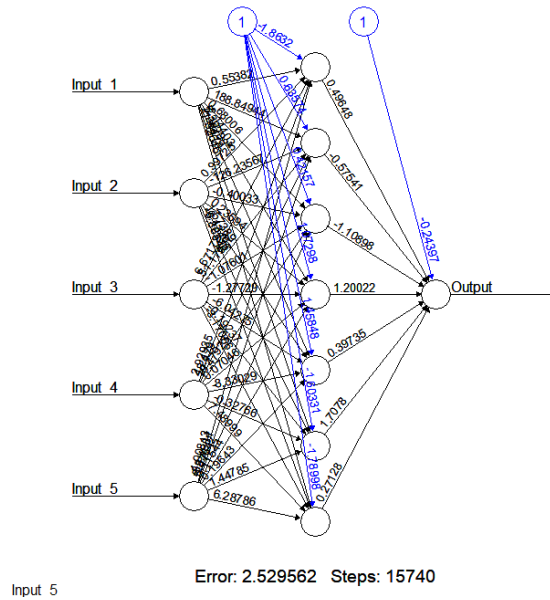
Error: 3.763438 Steps: 4126



Error: 3.62362 Steps: 1101



Error: 3.41931 Steps: 1315



```

126 # Using performance indicators to determine the optimal NN topologies
127
128 # Calculation of the actual output of each I/O matrix's testing data
129
130 T_1_actual_output <- Test_t_1[, "output"]
131 T_2_actual_output <- Test_t_2[, "output"]
132 T_3_actual_output <- Test_t_3[, "output"]
133 T_4_actual_output <- Test_t_4[, "output"]
134 T_5_actual_output <- Test_t_5[, "output"]
135
136 #Then the predicted output from each model is calculated
137
138 T_1_predic_output1 <- predict(object = T_1_NN1, Test_t_1)
139 T_1_predic_output2 <- predict(object = T_1_NN2, Test_t_1)
140 T_1_predic_output3 <- predict(object = T_1_NN3, Test_t_1)
141
142 T_2_predic_output1 <- predict(object = T_2_NN1, Test_t_2)
143 T_2_predic_output2 <- predict(object = T_2_NN2, Test_t_2)
144 T_2_predic_output3 <- predict(object = T_2_NN3, Test_t_2)
145
146 T_3_predic_output1 <- predict(object = T_3_NN1, Test_t_3)
147 T_3_predic_output2 <- predict(object = T_3_NN2, Test_t_3)
148 T_3_predic_output3 <- predict(object = T_3_NN3, Test_t_3)
149
150 T_4_predic_output1 <- predict(object = T_4_NN1, Test_t_4)
151 T_4_predic_output2 <- predict(object = T_4_NN2, Test_t_4)
152 T_4_predic_output3 <- predict(object = T_4_NN3, Test_t_4)
153
154 T_5_predic_output1 <- predict(object = T_5_NN1, Test_t_5)
155 T_5_predic_output2 <- predict(object = T_5_NN2, Test_t_5)
156 T_5_predic_output3 <- predict(object = T_5_NN3, Test_t_5)
157

```

Performance indicators may be created to evaluate the accuracy and efficacy of each model by comparing the expected outputs from the models to the actual outputs. This research aids in choosing the best neural network design for the specific forecasting electricity load problem.



## Unnormalize to find performance metrics

```
158 # Define the unnormalize function
159 unnormalize <- function(x, min_val, max_val) {
160   return (x * (max_val - min_val) + min_val)
161 }
162
163 # Unnormalize the predicted outputs and actual outputs for each model
164
165 T_1_predic_output1_unnorma <- unnormalize(T_1_predic_output1, min(Train_t_1[, "Output"]), max(Train_t_1[, "Output"]))
166 T_1_predic_output2_unnorma <- unnormalize(T_1_predic_output2, min(Train_t_1[, "Output"]), max(Train_t_1[, "Output"]))
167 T_1_predic_output3_unnorma <- unnormalize(T_1_predic_output3, min(Train_t_1[, "Output"]), max(Train_t_1[, "Output"]))
168
169 T_1_act_output_unnorma <- unnormalize(T_1_actual_output, min(Train_t_1[, "Output"]), max(Train_t_1[, "Output"]))
170
171 T_2_predic_output1_unnorma <- unnormalize(T_2_predic_output1, min(Train_t_2[, "Output"]), max(Train_t_2[, "Output"]))
172 T_2_predic_output2_unnorma <- unnormalize(T_2_predic_output2, min(Train_t_2[, "Output"]), max(Train_t_2[, "Output"]))
173 T_2_predic_output3_unnorma <- unnormalize(T_2_predic_output3, min(Train_t_2[, "Output"]), max(Train_t_2[, "Output"]))
174
175 T_2_act_output_unnorma <- unnormalize(T_2_actual_output, min(Train_t_2[, "Output"]), max(Train_t_2[, "Output"]))
176
177 T_3_predic_output1_unnorma <- unnormalize(T_3_predic_output1, min(Train_t_3[, "Output"]), max(Train_t_3[, "Output"]))
178 T_3_predic_output2_unnorma <- unnormalize(T_3_predic_output2, min(Train_t_3[, "Output"]), max(Train_t_3[, "Output"]))
179 T_3_predic_output3_unnorma <- unnormalize(T_3_predic_output3, min(Train_t_3[, "Output"]), max(Train_t_3[, "Output"]))
180
181 T_3_act_output_unnorma <- unnormalize(T_3_actual_output, min(Train_t_3[, "Output"]), max(Train_t_3[, "Output"]))
182
183 T_4_predic_output1_unnorma <- unnormalize(T_4_predic_output1, min(Train_t_4[, "Output"]), max(Train_t_4[, "Output"]))
184 T_4_predic_output2_unnorma <- unnormalize(T_4_predic_output2, min(Train_t_4[, "Output"]), max(Train_t_4[, "Output"]))
185 T_4_predic_output3_unnorma <- unnormalize(T_4_predic_output3, min(Train_t_4[, "Output"]), max(Train_t_4[, "Output"]))
186
187 T_4_act_output_unnorma <- unnormalize(T_4_actual_output, min(Train_t_4[, "Output"]), max(Train_t_4[, "Output"]))
188
189 T_5_predic_output1_unnorma <- unnormalize(T_5_predic_output1, min(Train_t_5[, "Output"]), max(Train_t_5[, "Output"]))
190 T_5_predic_output2_unnorma <- unnormalize(T_5_predic_output2, min(Train_t_5[, "Output"]), max(Train_t_5[, "Output"]))
191 T_5_predic_output3_unnorma <- unnormalize(T_5_predic_output3, min(Train_t_5[, "Output"]), max(Train_t_5[, "Output"]))
192
193 T_5_act_output_unnorma <- unnormalize(T_5_actual_output, min(Train_t_5[, "Output"]), max(Train_t_5[, "Output"]))
194
```

The unnormalize function is used to denormalize the anticipated and actual outputs for each neural network model. The justifications for unnormalization are the minimum and maximum values of the training data for each model. The outputs that haven't been normalized are kept in separate variables with the prefix "unnorma" to show that they are still in the original scale. The purpose of unnormalization is to transform the normalized values back to their original scale or range.

The convergence and stability of the neural network training process are improved by normalizing the data. The outputs that were anticipated using the normalized data are, nevertheless, likewise in the normalized scale. We must rescale the projected outputs to the original scale in order to properly assess the performance of the models and compare them to the original data.

The unnormalize function is used to denormalize the anticipated and actual outputs for each neural network model. The justifications for unnormalization are the minimum and maximum values of the training data for each model. The outputs that haven't been normalized are kept in separate variables with the prefix "unnorma" to show that they are still in the original scale.

Once the data was denormalized the function was introduced to find the performance metrics,

```
195 #Introduce a function to find performance metrics
196
197 library(Metrics)
198 performance_metrics <- function(actu_output, predi_output){
199   return(list(RMSE = rmse(actu_output, predi_output),
200             MAE = mae(actu_output, predi_output),
201             MAPE = mape(actu_output, predi_output),
202             SMAPE = smape(actu_output, predi_output)))
203 }
```

A snippet of the different performance metrics is show below,

```
> T_1_NN1_performance
$RMSE
[1] 0.1147376

$MAE
[1] 0.09182907

$MAPE
[1] 0.181412

$SMAPE
[1] 0.1789915

> T_2_NN1_performance
$RMSE
[1] 0.1167291

$MAE
[1] 0.09365645

$MAPE
[1] 0.1857331

$SMAPE
[1] 0.1828456
```

d)

### RMSE

RMSE (Root Mean Square Error) is a frequently used statistic to assess the average size of discrepancies between expected and observed values. The average of the squared discrepancies between the expected and actual values is considered in its calculation. A prediction model's overall accuracy may be evaluated using RMSE. The RMSE is frequently used as the loss function to optimize the model during training in the setting of neural networks. Better performance is indicated by a lower RMSE, with a value of 0 denoting a perfect match.

### MAE

Another statistic for gauging the average size of discrepancies between expected and actual data is called MAE (Mean Absolute Error). It is derived by averaging the absolute disparities between the values that were anticipated and those that occurred. Since MAE does not square the errors, it is less sensitive to outliers than RMSE. In the context of neural networks, the MAE is widely employed as a different loss

function to optimize the model during training. A lower MAE implies higher performance, and a value of 0 denotes a perfect match, like RMSE.

### **MAPE**

The average percentage difference between expected and actual values is measured using the metric known as MAPE (Mean Absolute Percentage Error). The absolute % error is determined by averaging the absolute differences between the predicted and actual values divided by the actual value, multiplied by 100. In order to evaluate the accuracy of forecasts in relation to the actual values, MAPE is frequently employed in forecasting and demand planning. A lower MAPE number corresponds to more accuracy, with a value of 0 being the ideal fit. However, because division by zero is undefinable, MAPE might present issues when the real numbers are close to or contain zero values.

### **sMAPE**

A MAPE variant that overcomes some of MAPE's drawbacks is called sMAPE (Symmetric Mean Absolute Percentage Error). It is determined by averaging the absolute percentage differences between predicted and actual values. The absolute percentage difference is calculated by dividing the absolute difference between the predicted and actual values by the sum of the predicted and actual values, then by 200. When the actual and projected values can both be positive and negative, sMAPE offers a symmetric measure of percentage error that is resistant to extreme values. Similar to MAPE, a lower sMAPE score denotes more accuracy, with 0 denoting the ideal match.

e)

	Description	RMSE	MAE	MAPE	sMAPE
T_1_NN1_performance	1 input 1 hidden layer 3 nodes	0.1153313	0.0921429	0.1819827	0.1799046
T_2_NN1_performance	2 input 1 hidden layer 3 nodes	0.113877	0.0914781	0.1811966	0.1776396
T_3_NN1_performance	3 input 1 hidden layer 3 nodes	0.1133131	0.09086343	0.1790859	0.1759688
T_4_NN1_performance	4 input 1 hidden layer 3 nodes	0.1167263	0.09440628	0.1863288	0.1830204
T_5_NN1_performance	5 input 1 hidden layer 3 nodes	0.09443027	0.07512496	0.149125	0.1435385
T_1_NN2_performance	1 input 2 hidden layer 3,4 nodes	0.1144274	0.09183329	0.1814191	0.1788346
T_2_NN2_performance	2 input 2 hidden layer 3,4 nodes	0.1147527	0.09231206	0.1829044	0.179439
T_3_NN2_performance	3 input 2 hidden layer 3,4 nodes	0.1144001	0.09121519	0.1790019	0.1775238
T_4_NN2_performance	4 input 2 hidden layer 3,4 nodes	0.1140976	0.09030062	0.1824238	0.1762515
T_5_NN2_performance	5 input 2 hidden layer 3,4 nodes	0.09623449	0.0760473	0.1498663	0.1449559
	1 input	0.1155296	0.09257326	0.1831243	0.1807129

T_1_NN3_performance	1 hidden layer 7 nodes				
T_2_NN3_performance	2 input 1 hidden layer 7 nodes	0.1149252	0.09247564	0.1826339	0.1805615
T_3_NN3_performance	3 input 1 hidden layer 7 nodes	0.1160787	0.0920711	0.1806203	0.179455
T_4_NN3_performance	3 input 1 hidden layer 7 nodes	0.1163124	0.09353578	0.1839343	0.1813833
T_5_NN3_performance	3 input 1 hidden layer 7 nodes	0.09641611	0.07572348	0.147448	0.1461961

The best one-layer neural network with the least errors is **T\_5\_NN1\_performance** with 5 inputs and 3 nodes.

The best two-layer neural network with the least errors is **T\_5\_NN2\_performance** with 5 inputs and 3,4 nodes respectively.

**f)** A neural network's overall weight parameter count can reveal information about its complexity and potential effectiveness. The number of weight parameters can significantly affect how well a neural network performs, even if it is not the only factor to consider. The complexity of the neural network model is indicated by how many weight parameters there are. A model that can capture intricate patterns and connections in the data will often have a higher number of weight parameters. T\_5\_NN2 is this case's more sophisticated neural network. The amount of weight parameters influences the computation needed for training and inference. For models with more parameters', more computing power is often required. Although adding additional parameters may enhance performance on training data, this does not mean that enhanced performance will also be achieved on untrained data. A simpler model with fewer parameters may occasionally generalize better and exhibit improved efficiency because it prevents overfitting and reduces the risk of high variation.

## **Appendix**

### **1<sup>st</sup> Objective Partition Clustering – Source Code**

```
library(NbClust)

library(ggplot2)

library(gridExtra)

library(factoextra)

library(cluster)

library(MASS)


#Importing Data set

library(readxl)

vehicles_df <- read_excel("vehicles.xlsx")

sum(is.na(vehicles_df)) #Checks for missing data values


#Overview of dataset

View(vehicles_df)

summary(vehicles_df)


#dataset with outliers

boxplot(vehicles_df[,2:19])

#https://www.geeksforgeeks.org/how-to-remove-outliers-from-multiple-columns-in-r-dataframe/

#Columns with Outliers >>>>Rad.Ra , Pr.Axis.Ra , Max.L.Ra ,Sc.Var.Maxis , Sc.Var.maxis ,Skew.Maxis ,
Skew.maxis,Kurt.maxis

detect_outlier <- function(x) {

  #Uses the interquartile range to locate the outliers

  Q1 <- quantile(x, probs = 0.25)

  Q3 <- quantile(x, probs = 0.75)

  IQR <- Q3 - Q1
```

```

outliers <- x > (Q3 + 1.5 * IQR) | x < (Q1 - 1.5 * IQR)
return(outliers)
}

# define remove_outlier() function
remove_outlier <- function(dataframe) {
  for (i in 2:ncol(dataframe)) { # ignores the sample column
    if (is.numeric(dataframe[[i]])) { # skip non-numeric columns
      dataframe <- dataframe[!detect_outlier(dataframe[[i]]), ]
    }
  }
  message("Outliers have been removed")
  return(dataframe)
}

clean_data <- remove_outlier(vehicles_df)

#Calculate the Z-Score of clean_data
numeric_col <- clean_data[,2:19]
normalized_data <- scale(numeric_col)

#View the standardized data
normalized_data

boxplot(clean_data[,2:19])

#NbClust Method
set.seed(1445)

num_clusters <- NbClust(normalized_data,distance="euclidean",min.nc = 2,max.nc =8,method =
"kmeans",index="all")
table(num_clusters$Best.nc)

```



#Elbow Method

```
set.seed(63)
```

```
elbowmethod_plot<-fviz_nbclust(normalized_data, kmeans,method ="wss") +
```

```
  geom_vline(xintercept = 3, linetype = 2)+
```

```
  labs(subtitle = "Elbow method")
```

```
print(elbowmethod_plot)
```

#Gap Statistics Method

```
set.seed(123)
```

```
gap_stat<-fviz_nbclust(normalized_data,kmeans, nstart = 25, method = "gap_stat", nboot = 100,  
iter.max=50)+
```

```
  labs(subtitle = "Gap statistic method")
```

```
print(gap_stat)
```

#Silhouette Method

```
set.seed(467)
```

```
silhoutte_plot<-fviz_nbclust(normalized_data, kmeans, method ="silhouette")+
```

```
  labs(subtitle = "Silhouette method")
```

```
print(silhoutte_plot)
```

#k- Means Clustering

```
ApplykMeans <- function(normalized_data,k){
```

```
  Km <- kmeans(normalized_data[,-length(normalized_data)],k)
```

```
  C <- Km$centers
```

```
  S <- Km$size
```

```
  WSS<- Km$withinss
```

```
  BSS <-Km$betweenss
```

```
  TSS<- Km$betweenss/Km$totss
```

```
  I <- Km$iter
```

```

cluster_plot<-fviz_cluster(Km, data = normalized_data,
                           palette = c("#AE3FEA", "#00AFBB", "#E7B800", "#aff0e7"),
                           geom = "point", ellipse.type = "convex", ggtheme = theme_bw(),
                           silhouette_plot <- silhouette(Km$cluster, dist(normalized_data)),
                           print(fviz_silhouette(silhouette_plot))

)

return(list(Number_of_clusters = k,
           Number_of_points_for_each_cluster = S,
           Number_of_iterations = I,
           Centers = C,
           Withinss = WSS,
           Betweenss = BSS,
           Between_to_total_ratio = TSS,
           Visual_cluster_plot = cluster_plot)) #internal metrics
}

ApplykMeans(normalized_data, 3)

# Internal Evaluation Metrics

silhouette_plot <- silhouette(Km$cluster, dist(normalized_data))
print(fviz_silhouette(silhouette_plot))

#PCA

library(fpc)

get_pca_attr <- function(pca_data){
  eigenvalue <- pca_data$sdev^2
  eigenvector <- pca_data$rotation
  cumulative_score <-cumsum(eigenvalue / sum(eigenvalue))

```

```

print("Eigenvalues:")
print(eigenvalue[cumulative_score < 0.92])

print("Eigenvectors:")
print(eigenvector[, cumulative_score < 0.92])

print("Cumulative scores:")
print(cumulative_score[cumulative_score < 0.92])

return ( cumulative_score)

}

findCalinskiIndex <- function(data_frame, k){
  kmeans <- kmeans(data_frame, k)
  ch_index <- round(calinhara(normalized_data, kmeans$cluster), digits = 2)
  cat("Calinski-Harabasz Index:", ch_index, "\n")
}

pca_great_data <- prcomp(normalized_data, center = TRUE, scale = FALSE)
summary(pca_great_data)

# Identifies the data sets that are within the cum score threshold of <92%
cumulative_score <- get_pca_attr(pca_great_data)
pcadataframe <- pca_great_data$x[, cumulative_score < 0.92]

```

```
summary(pcadataframe)
```

```
#To determine the optimal number of Clusters
```

```
spotClusters <- function(data_frame){
```

```
  #NbClust Method
```

```
  set.seed(1445)
```

```
  num_clusters <- NbClust(normalized_data, distance="euclidean", min.nc = 2, max.nc = 8, method =  
  "kmeans", index="all")
```

```
  table(num_clusters$Best.nc)
```

```
  #Elbow Method
```

```
  set.seed(63)
```

```
  elbowmethod_plot <- fviz_nbclust(normalized_data, kmeans, method = "wss") +
```

```
    geom_vline(xintercept = 3, linetype = 2) +
```

```
    labs(subtitle = "Elbow method")
```

```
  print(elbowmethod_plot)
```

```
  #Gap Statistics Method
```

```
  set.seed(123)
```

```
  gap_stat <- fviz_nbclust(normalized_data, kmeans, nstart = 25, method = "gap_stat", nboot = 100,  
  iter.max = 50) +
```

```
    labs(subtitle = "Gap statistic method")
```

```
  print(gap_stat)
```

```
  #Silhouette Method
```

```
  set.seed(467)
```

```
  silhoutte_plot <- fviz_nbclust(normalized_data, kmeans, method = "silhouette") +
```

```
    labs(subtitle = "Silhouette method")
```

```
  print(silhoutte_plot)
```

```

} #Function defined by the user

spotClusters(pcadataframe)

#kmeansclustering

ApplykMeans <- function(pcadataframe,k){
  Km <- kmeans(pcadataframe[,-length(pcadataframe)],k)
  C <- Km$centers
  S <- Km$size
  WSS<- Km$withinss
  BSS <-Km$betweenss
  TSS<- Km$betweenss/Km$totss
  I <- Km$iter
  cluster_plot<-fviz_cluster(Km, data = pcadataframe,
                             palette = c("#AE3FEA", "#00AFBB", "#E7B800", "#aff0e7"),
                             geom = "point", ellipse.type = "convex", ggtheme = theme_bw(),
                             silhouette_plot <- silhouette(Km$cluster, dist(pcadataframe)),
                             print(fviz_silhouette(silhouette_plot))

)
  return(list(Number_of_clusters = k,
              Number_of_points_for_each_cluster = S,
              Number_of_iterations = I,
              Centers = C,
              Withinss = WSS,
              Betweenss = BSS,
              Between_to_total_ratio = TSS,
              Visual_cluster_plot = cluster_plot)) #internal metrics
}

```

```
#Gap Statistics as most favored clusters
```

```
#k=3
```

```
km3 <- kmeans(pcadataframe,3)
```

```
km3
```

```
WSS_G <- km3$withinss
```

```
WSS_G
```

```
BSS_G <- km3$betweenss
```

```
BSS_G
```

```
TSS_G <- km3$totss
```

```
TSS_G
```

```
BSS_G_TSS_G_ratio <- BSS_G/TSS_G
```

```
BSS_G_TSS_G_ratio
```

```
ApplykMeans(pcadataframe, 3)
```

```
#Clinski Index - Internal Evaluation Metrics
```

```
findCalinskiIndex(pcadataframe,3)
```

## **2nd Objective MLP – Source Code**

```
library(tidyverse)

library(neuralnet)

library(dplyr)

library(readxl)


#loading the dataset

uow_load <- read_excel("uow_consumption.xlsx")

uow_load


# Replace the column names with new relevant column names

colnames(uow_load) <- c("Date", "18", "19", "20")

View(head(uow_load))


#creating time-delayed input variables along with I/O Matrix

time_delayed <- bind_cols(
  T7 = lag(uow_load$'20', 7),
  T4 = lag(uow_load$'20', 4),
  T3 = lag(uow_load$'20', 3),
  T2 = lag(uow_load$'20', 2),
  T1 = lag(uow_load$'20', 1),
  outputprediction = lag(uow_load$'20', 0)
)

time_delayed


# Remove rows with missing values

time_delayed <- na.omit(time_delayed)

time_delayed
```

```

# Construction of different time-delayed input vectors and related i/o matrices.

t_1 <- cbind(time_delayed$T1, time_delayed$outputprediction)
colnames(t_1)<- c("Input", "Output")


t_2 <- cbind(time_delayed$T1, time_delayed$T2, time_delayed$outputprediction)
colnames(t_2)<- c("Input_1", "Input_2", "Output")


t_3 <- cbind(time_delayed$T1, time_delayed$T2, time_delayed$T3, time_delayed$outputprediction)
colnames(t_3)<- c("Input_1", "Input_2", "Input_3", "Output")


t_4 <- cbind(time_delayed$T1, time_delayed$T2, time_delayed$T3, time_delayed$T4,
time_delayed$outputprediction)
colnames(t_4)<- c("Input_1", "Input_2", "Input_3", "Input_4", "Output")


t_5 <- cbind(time_delayed$T1, time_delayed$T2, time_delayed$T3, time_delayed$T4,time_delayed$T7,
time_delayed$outputprediction)
colnames(t_5)<- c("Input_1", "Input_2", "Input_3", "Input_4", "Input_5", "Output")


#Defining the normalization function (Min-Max Normalization)

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


#Normalizing the I/O Matrices

normat_1 <- norma(t_1)
normat_2 <- norma(t_2)
normat_3 <- norma(t_3)
normat_4 <- norma(t_4)
normat_5 <- norma(t_5)

```



```
#define the training sets and testing sets for each I/O matrix
```

```
Train_t_1 <- normat_1[1:380,]
```

```
Test_t_1 <- normat_1[381: nrow(normat_1),]
```

```
Train_t_2 <- normat_2[1:380,]
```

```
Test_t_2 <- normat_2[381: nrow(normat_2),]
```

```
Train_t_3 <- normat_3[1:380,]
```

```
Test_t_3 <- normat_3[381: nrow(normat_3),]
```

```
Train_t_4 <- normat_4[1:380,]
```

```
Test_t_4 <- normat_4[381: nrow(normat_4),]
```

```
Train_t_5 <- normat_5[1:380,]
```

```
Test_t_5 <- normat_5[381: nrow(normat_5),]
```

```
# Training the Neural network for normat_1
```

```
# One hidden layer neural networks version #1
```

```
T_1_NN1 <- neuralnet(Output ~ Input, data = Train_t_1, hidden = 3, linear.output = TRUE)
```

```
plot(T_1_NN1)
```

```
T_2_NN1 <- neuralnet(Output ~ Input_1 + Input_2, data = Train_t_2, hidden = 3, linear.output = TRUE)
```

```
plot(T_2_NN1)
```

```
T_3_NN1 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3, data = Train_t_3, hidden = 3, linear.output  
= TRUE)
```

```
plot(T_3_NN1)
```

```
T_4_NN1 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4, data = Train_t_4, hidden = 3,  
linear.output = TRUE)
```

```
plot(T_4_NN1)
```

```
T_5_NN1 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4 + Input_5, data = Train_t_5,  
hidden = 3, linear.output = TRUE)
```

```
plot(T_5_NN1)
```

```
# Two hidden layer Neural networks
```

```
T_1_NN2 <- neuralnet(Output ~ Input, data = Train_t_1, hidden = c(3,4), linear.output = TRUE)
```

```
plot(T_1_NN2)
```

```
T_2_NN2 <- neuralnet(Output ~ Input_1 + Input_2, data = Train_t_2, hidden = c(3,4), linear.output =  
TRUE)
```

```
plot(T_2_NN2)
```

```
T_3_NN2 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3, data = Train_t_3, hidden = c(3,4),  
linear.output = TRUE)
```

```
plot(T_3_NN2)
```

```
T_4_NN2 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4, data = Train_t_4, hidden = c(3,4),  
linear.output = TRUE)
```

```
plot(T_4_NN2)
```

```
T_5_NN2 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4 + Input_5, data = Train_t_5,  
hidden = c(3,4), linear.output = TRUE)
```

```
plot(T_5_NN2)
```

```
# One hidden layer neural networks version #2
```

```
T_1_NN3 <- neuralnet(Output ~ Input, data = Train_t_1, hidden = 7, linear.output = TRUE)
```

```
plot(T_1_NN3)
```

```
T_2_NN3 <- neuralnet(Output ~ Input_1 + Input_2, data = Train_t_2, hidden = 7, linear.output = TRUE)
```

```
plot(T_2_NN3)
```

```
T_3_NN3 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3, data = Train_t_3, hidden = 7, linear.output  
= TRUE)
```

```
plot(T_3_NN3)
```

```
T_4_NN3 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4, data = Train_t_4, hidden = 7,  
linear.output = TRUE)
```

```
plot(T_4_NN3)
```

```
T_5_NN3 <- neuralnet(Output ~ Input_1 + Input_2 + Input_3 + Input_4 + Input_5, data = Train_t_5,  
hidden = 7, linear.output = TRUE)
```

```
plot(T_5_NN3)
```

```
# Using performance indicators to determine the optimal NN topologies
```

```
# Calculation of the actual output of each I/O matrix's testing data
```

```
T_1_actual_output <- Test_t_1[, "Output"]
```

```
T_2_actual_output <- Test_t_2[, "Output"]
```

```
T_3_actual_output <- Test_t_3[, "Output"]
```

```
T_4_actual_output <- Test_t_4[, "Output"]
```

```
T_5_actual_output <- Test_t_5[, "Output"]
```

#Then the predicted output from each model is calculated

```
T_1_predic_output1 <- predict(object = T_1_NN1, Test_t_1)
```

```
T_1_predic_output2 <- predict(object = T_1_NN2, Test_t_1)
```

```
T_1_predic_output3 <- predict(object = T_1_NN3, Test_t_1)
```

```
T_2_predic_output1 <- predict(object = T_2_NN1, Test_t_2)
```

```
T_2_predic_output2 <- predict(object = T_2_NN2, Test_t_2)
```

```
T_2_predic_output3 <- predict(object = T_2_NN3, Test_t_2)
```

```
T_3_predic_output1 <- predict(object = T_3_NN1, Test_t_3)
```

```
T_3_predic_output2 <- predict(object = T_3_NN2, Test_t_3)
```

```
T_3_predic_output3 <- predict(object = T_3_NN3, Test_t_3)
```

```
T_4_predic_output1 <- predict(object = T_4_NN1, Test_t_4)
```

```
T_4_predic_output2 <- predict(object = T_4_NN2, Test_t_4)
```

```
T_4_predic_output3 <- predict(object = T_4_NN3, Test_t_4)
```

```
T_5_predic_output1 <- predict(object = T_5_NN1, Test_t_5)
```

```
T_5_predic_output2 <- predict(object = T_5_NN2, Test_t_5)
```

```
T_5_predic_output3 <- predict(object = T_5_NN3, Test_t_5)
```

# Define the unnormalize function

```
unnormalize <- function(x, min_val, max_val) {
```

```
  return (x * (max_val - min_val) + min_val)
```

```
}
```

# Unnormalize the predicted outputs and actual outputs for each model

```
T_1_predic_output1_unnorma <- unnormailize(T_1_predic_output1, min(Train_t_1[, "Output"]),  
max(Train_t_1[, "Output"]))
```

```
T_1_predic_output2_unnorma <- unnormailize(T_1_predic_output2, min(Train_t_1[, "Output"]),  
max(Train_t_1[, "Output"]))
```

```
T_1_predic_output3_unnorma <- unnormailize(T_1_predic_output3, min(Train_t_1[, "Output"]),  
max(Train_t_1[, "Output"]))
```

```
T_1_act_output_unnorma <- unnormailize(T_1_actual_output, min(Train_t_1[, "Output"]),  
max(Train_t_1[, "Output"]))
```

```
T_2_predic_output1_unnorma <- unnormailize(T_2_predic_output1, min(Train_t_2[, "Output"]),  
max(Train_t_2[, "Output"]))
```

```
T_2_predic_output2_unnorma <- unnormailize(T_2_predic_output2, min(Train_t_2[, "Output"]),  
max(Train_t_2[, "Output"]))
```

```
T_2_predic_output3_unnorma <- unnormailize(T_2_predic_output3, min(Train_t_2[, "Output"]),  
max(Train_t_2[, "Output"]))
```

```
T_2_act_output_unnorma <- unnormailize(T_2_actual_output, min(Train_t_2[, "Output"]),  
max(Train_t_2[, "Output"]))
```

```
T_3_predic_output1_unnorma <- unnormailize(T_3_predic_output1, min(Train_t_3[, "Output"]),  
max(Train_t_3[, "Output"]))
```

```
T_3_predic_output2_unnorma <- unnormailize(T_3_predic_output2, min(Train_t_3[, "Output"]),  
max(Train_t_3[, "Output"]))
```

```
T_3_predic_output3_unnorma <- unnormailize(T_3_predic_output3, min(Train_t_3[, "Output"]),  
max(Train_t_3[, "Output"]))
```

```
T_3_act_output_unnorma <- unnormailize(T_3_actual_output, min(Train_t_3[, "Output"]),  
max(Train_t_3[, "Output"]))
```

```
T_4_predic_output1_unnorma <- unnormailize(T_4_predic_output1, min(Train_t_4[, "Output"]),  
max(Train_t_4[, "Output"]))
```

```
T_4_predic_output2_unnorma <- unnormailize(T_4_predic_output2, min(Train_t_4[, "Output"]),  
max(Train_t_4[, "Output"]))
```

```
T_4_predic_output3_unnorma <- unnormailze(T_4_predic_output3, min(Train_t_4[, "Output"]),
max(Train_t_4[, "Output"]))
```

```
T_4_act_output_unnorma <- unnormailze(T_4_actual_output, min(Train_t_4[, "Output"]),
max(Train_t_4[, "Output"]))
```

```
T_5_predic_output1_unnorma <- unnormailze(T_5_predic_output1, min(Train_t_5[, "Output"]),
max(Train_t_5[, "Output"]))
```

```
T_5_predic_output2_unnorma <- unnormailze(T_5_predic_output2, min(Train_t_5[, "Output"]),
max(Train_t_5[, "Output"]))
```

```
T_5_predic_output3_unnorma <- unnormailze(T_5_predic_output3, min(Train_t_5[, "Output"]),
max(Train_t_5[, "Output"]))
```

```
T_5_act_output_unnorma <- unnormailze(T_5_actual_output, min(Train_t_5[, "Output"]),
max(Train_t_5[, "Output"]))
```

```
#Introduce a function to find performance metrics
```

```
library(Metrics)
```

```
performance_metrics <- function(actu_output, predi_output){
  return(list(RMSE = rmse(actu_output, predi_output),
             MAE = mae(actu_output, predi_output),
             MAPE = mape(actu_output, predi_output),
             SMAPE = smape(actu_output, predi_output)))
}
```

```
# calculon of performance metrics for each model
```

```
T_1_NN1_performance <- performance_metrics(T_1_actual_output, T_1_predic_output1)
```

```
T_2_NN1_performance <- performance_metrics(T_2_actual_output, T_2_predic_output1)
```

```
T_3_NN1_performance <- performance_metrics(T_3_actual_output, T_3_predic_output1)
```

```
T_4_NN1_performance <- performance_metrics(T_4_actual_output, T_4_predic_output1)
```

```
T_5_NN1_performance <- performance_metrics(T_5_actual_output, T_5_predic_output1)
```

```
T_1_NN2_performance <- performance_metrics(T_1_actual_output, T_1_predic_output2)
```

```
T_2_NN2_performance <- performance_metrics(T_2_actual_output, T_2_predic_output2)
```

```
T_3_NN2_performance <- performance_metrics(T_3_actual_output, T_3_predic_output2)
```

```
T_4_NN2_performance <- performance_metrics(T_4_actual_output, T_4_predic_output2)
```

```
T_5_NN2_performance <- performance_metrics(T_5_actual_output, T_5_predic_output2)
```

```
T_1_NN3_performance <- performance_metrics(T_1_actual_output, T_1_predic_output3)
```

```
T_2_NN3_performance <- performance_metrics(T_2_actual_output, T_2_predic_output3)
```

```
T_3_NN3_performance <- performance_metrics(T_3_actual_output, T_3_predic_output3)
```

```
T_4_NN3_performance <- performance_metrics(T_4_actual_output, T_4_predic_output3)
```

```
T_5_NN3_performance <- performance_metrics(T_5_actual_output, T_5_predic_output3)
```

```
T_1_NN1_performance
```

```
T_2_NN1_performance
```

```
T_3_NN1_performance
```

```
T_4_NN1_performance
```

```
T_5_NN1_performance
```

```
T_1_NN2_performance
```

```
T_2_NN2_performance
```

```
T_3_NN2_performance
```

```
T_4_NN2_performance
```

```
T_5_NN2_performance
```

T\_1\_NN3\_performance

T\_2\_NN3\_performance

T\_3\_NN3\_performance

T\_4\_NN3\_performance

T\_5\_NN3\_performance



## **REFERENCES**

- Hong, T., Fan, S., & Gao, W. (2004). Long-term load forecasting using system identification. *IEEE Transactions on Power Systems*, 19(2), 834-840.
- Kim, H. M., Kim, H. J., & Kim, S. W. (2002). An integrated approach to short-term load forecasting using wavelet transform and neural network. *IEEE Transactions on Power Systems*, 17(2), 489-495.
- AlRashidi, M. R., & El-Naggar, K. M. (2011). Short-term load forecasting based on artificial neural networks. *IEEE Transactions on Power Systems*, 26(2), 937-944.
- Díaz, G., Sánchez, R., Otero, A., & Marroyo, L. (2002). Estimation of energy consumption in buildings using weather data. *Energy and Buildings*, 34(9), 959-965.
- Hong, T., & Fan, S. (2006). A long-term probabilistic electricity load forecasting model using similar days. *Energy*, 31(14), 2957-2965.
- Hong, T. and Fan, S., 2007. Short-term electricity load forecasting using the similar days approach. *Energy*, 32(9), pp.1671-1676.
- Khosravi, A., Nahavandi, S., Creighton, D., & Atiya, A. F. (2011). Dynamic evolving neural-fuzzy inference system (DENFIS) for electricity load forecasting. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 41(6), 829-839.
- Taylor, J. W. (2003). Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of the Operational Research Society*, 54(8), 799-805.