INFOMATICS INSTITUTE OF TECHNOLOGY

BEng (Hons) Software Engineering

5DATA002W
Machine Learning & Data Mining Coursework (2022/23)

Student Name – Srinayaka Palliyaguru Ge Kalindu Lokith

IIT NO – 20210332

UOW NO – w1867088

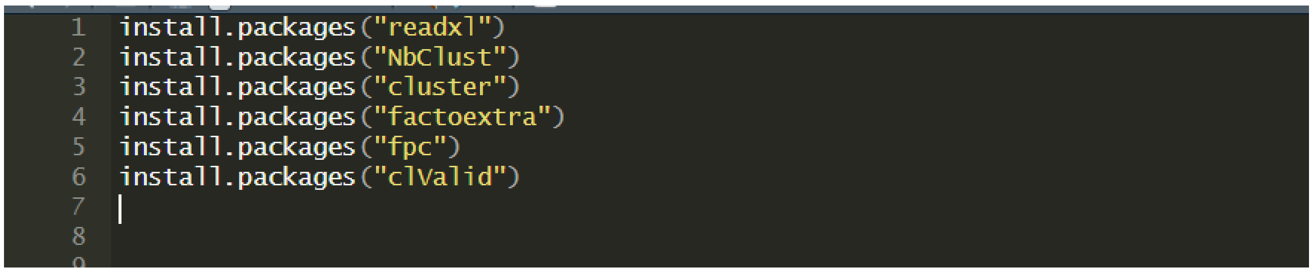
Table of Contents

Partitioning clustering:

1.1 1st Subtask Objectives

1.1.1 Install relevant Libraries

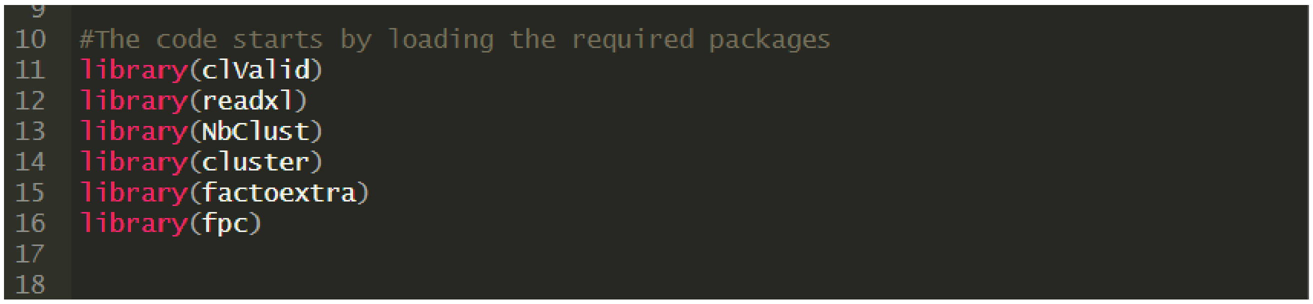
Import all of the relevant libraries first, as they are needed for the clustering part. It is mentioned as the primary objective



```r
install.packages("readxl")
install.packages("NbClust")
install.packages("cluster")
install.packages("factoextra")
install.packages("fpc")
install.packages("clValid")
```

1.1.2 Import Necessary Libraries

Import all the necessary libraries that are used in the partitioning clustering part.

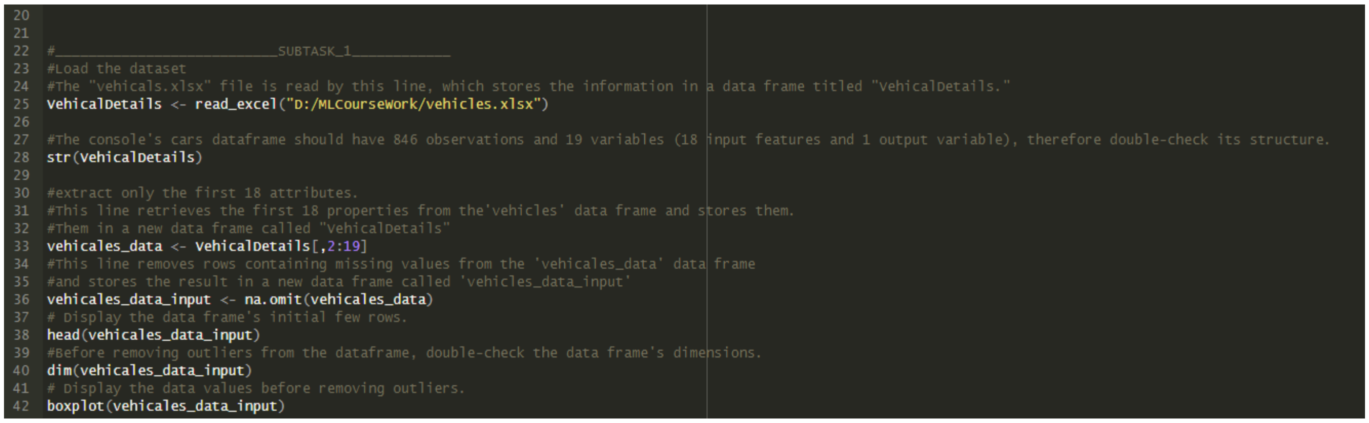


```r
#The code starts by loading the required packages
library(clValid)
library(readxl)
library(NbClust)
library(cluster)
library(factoextra)
library(fpc)
```

1.1.3 Read in the excel file

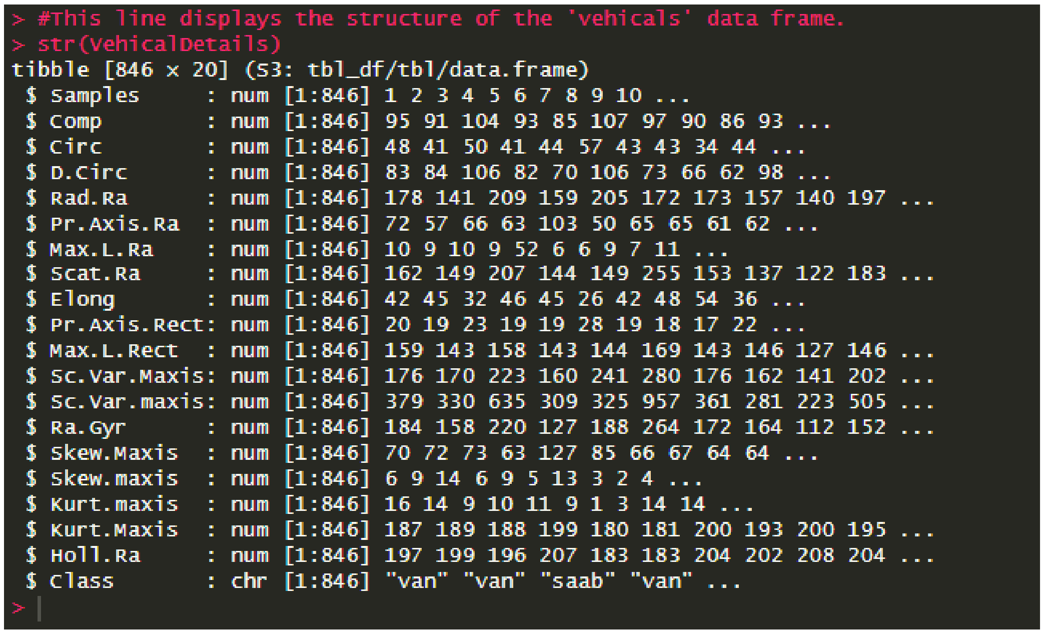
Observe the 846 observations/vehicle samples defined by 18 attributes (input variables) and one output variable (class) in the Excel file vehicles.xlsx. Only the first 18 attributes will be used for calculations in this clustering part. The clustering is an unsupervised technique, the information contained in the "class" attribute cannot be consumed. The required Excel file (vehicles.xlsx) is read using the read_excel() function.

The code:



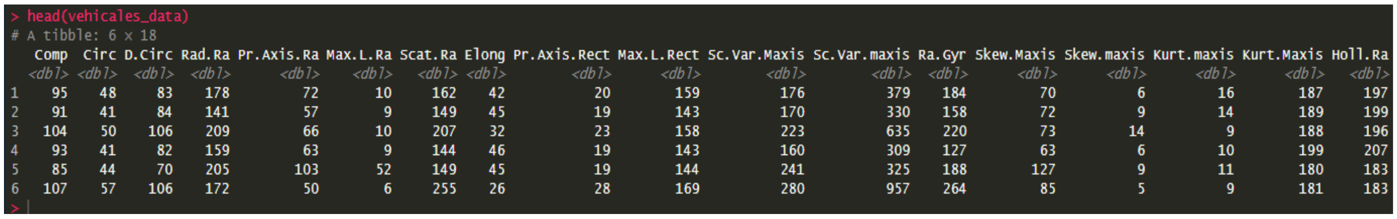
```


#____________________________SUBTASK_1____________
#Load the dataset
#The "vehicals.xlsx" file is read by this line, which stores the information in a data frame titled "VehicalDetails."
VehicalDetails <- read_excel("D:/MLCourseWork/vehicles.xlsx")

#The console's cars dataframe should have 846 observations and 19 variables (18 input features and 1 output variable), therefore double-check its structure.
str(VehicalDetails)

#extract only the first 18 attributes.
#This line retrieves the first 18 properties from the'vehicles' data frame and stores them.
#Them in a new data frame called "VehicalDetails"
vehicales_data <- VehicalDetails[,2:19]
#This line removes rows containing missing values from the 'vehicales_data' data frame
#and stores the result in a new data frame called 'vehicles_data_input'
vehicales_data_input <- na.omit(vehicales_data)
# Display the data frame's initial few rows.
head(vehicales_data_input)
#Before removing outliers from the dataframe, double-check the data frame's dimensions.
dim(vehicales_data_input)
# Display the data values before removing outliers.
boxplot(vehicales_data_input)
```

Retrieved data structure:



```
> #This line displays the structure of the 'vehicals' data frame.
> str(VehicalDetails)
tibble [846 x 20] (S3: tbl_df/tbl/data.frame)
 $ Samples     : num [1:846] 1 2 3 4 5 6 7 8 9 10 ...
 $ Comp        : num [1:846] 95 91 104 93 85 107 97 90 86 93 ...
 $ Circ        : num [1:846] 48 41 50 41 44 57 43 43 34 44 ...
 $ D.Circ      : num [1:846] 83 84 106 82 70 106 73 66 62 98 ...
 $ Rad.Ra      : num [1:846] 178 141 209 159 205 172 173 157 140 197 ...
 $ Pr.Axis.Ra  : num [1:846] 72 57 66 63 103 50 65 65 61 62 ...
 $ Max.L.Ra    : num [1:846] 10 9 10 9 52 6 6 9 7 11 ...
 $ Scat.Ra     : num [1:846] 162 149 207 144 149 255 153 137 122 183 ...
 $ Elong       : num [1:846] 42 45 32 46 45 26 42 48 54 36 ...
 $ Pr.Axis.Rect: num [1:846] 20 19 23 19 19 28 19 18 17 22 ...
 $ Max.L.Rect  : num [1:846] 159 143 158 143 144 169 143 146 127 146 ...
 $ Sc.Var.Maxis: num [1:846] 176 170 223 160 241 280 176 162 141 202 ...
 $ Sc.Var.maxis: num [1:846] 379 330 635 309 325 957 361 281 223 505 ...
 $ Ra.Gyr      : num [1:846] 184 158 220 127 188 264 172 164 112 152 ...
 $ Skew.Maxis  : num [1:846] 70 72 73 63 127 85 66 67 64 64 ...
 $ Skew.maxis  : num [1:846] 6 9 14 6 9 5 13 3 2 4 ...
 $ Kurt.maxis  : num [1:846] 16 14 9 10 11 9 1 3 14 14 ...
 $ Kurt.Maxis  : num [1:846] 187 189 188 199 180 181 200 193 200 195 ...
 $ Holl.Ra     : num [1:846] 197 199 196 207 183 183 204 202 208 204 ...
 $ Class       : chr [1:846] "van" "van" "saab" "van" ...
> 
```

Retrieved data summary:



```
> head(vehicales_data)
# A tibble: 6 x 18
   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 Ra.Gyr Skew.Maxis Skew.maxis Kurt.maxis Kurt.Maxis Holl.Ra
  <dbl> <dbl>  <dbl>  <dbl>      <dbl>    <dbl>   <dbl> <dbl>        <dbl>      <dbl>        <dbl>        <dbl>  <dbl>      <dbl>      <dbl>      <dbl>      <dbl>   <dbl>
1    95    48     83    178         72       10     162    42           20        159          176          379    184         70          6         16        187     197
2    91    41     84    141         57        9     149    45           19        143          170          330    158         72          9         14        189     199
3   104    50    106    209         66       10     207    32           23        158          223          635    220         73         14          9        188     196
4    93    41     82    159         63        9     144    46           19        143          160          309    127         63          6         10        199     207
5    85    44     70    205        103       52     149    45           19        144          241          325    188        127          9         11        180     183
6   107    57    106    172         50        6     255    26           28        169          280          957    264         85          5          9        181     183
> 
```

1.1.4 Data pre-processing

To ensure that the data is acceptable for the clustering algorithm, pre-processing tasks have to be performed before beginning k-means clustering.

There are two main pre-processing tasks that can be completed in this case.

1. Outlier removal
2. Scaling
3. The order of scaling and outliers removal

The order in which pre-processing tasks, scaling, and outlier detection/removal are completed is important. Scaling is always done after outlier detection and removal. This is because outliers can have a big impact on scaling, which causes scaling to be skewed. Therefore, it is typically advised to eliminate outliers before scaling the data. The data can be scaled once the outliers have been eliminated to ensure that each feature is on the same scale. This is essential because larger-scale properties may predominate in the clustering process and some clustering algorithms, such as k-means clustering, depend on the distance between data points.

1. Outlier removal

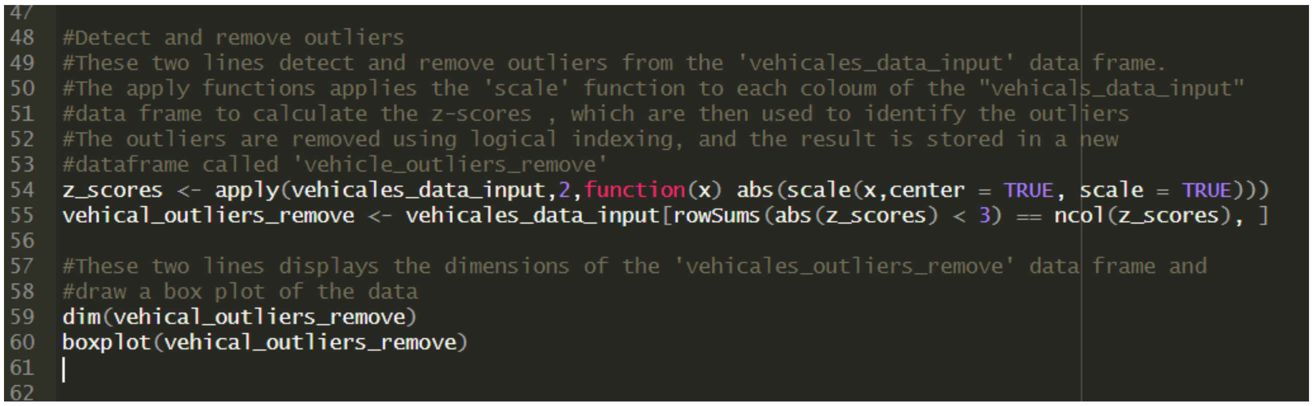
Outliers have a major impact on the performance of k-means clustering. The reason for this is that each data point's sum of squared distances from the cluster centroid is minimized during k-means clustering. Outliers, or data points that significantly depart from the rest of the data, can distort the distances between data points and the centroids of clusters, generating biased clustering results.

As an example, if there are outliers with extremely big values, their distance from the cluster centroids may be significantly larger than their distance from the other data points. Therefore, even if the outliers do not represent various groups in the data, the algorithm could place them into their own clusters.

As a result, before applying k-means clustering, outliers must be identified and eliminated. Z-scores, which determine how much a data point deviates from the mean in terms of standard deviations, are a popular technique for locating outliers. Data items that have z-scores higher than a preset threshold, in this example 3, are considered outliers. Any data points with a z-score greater than 3 (more than 3 standard deviations from the mean) for any feature will be removed from the dataset before clustering.

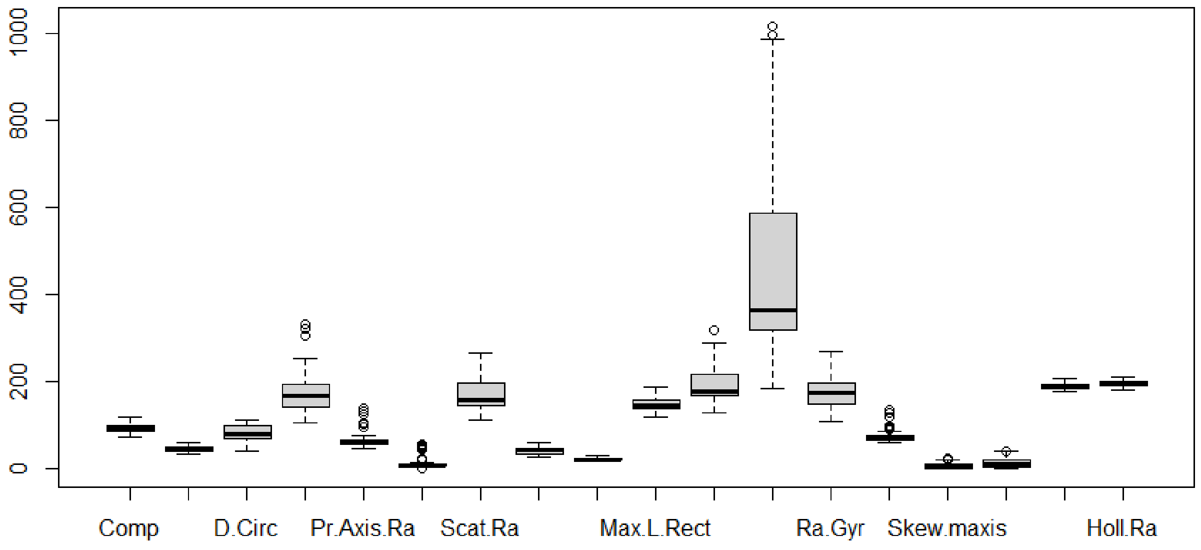
In general, removing outliers before clustering can help to guarantee that clustering results are more precise and indicative of the underlying data distribution.

The relevant code:

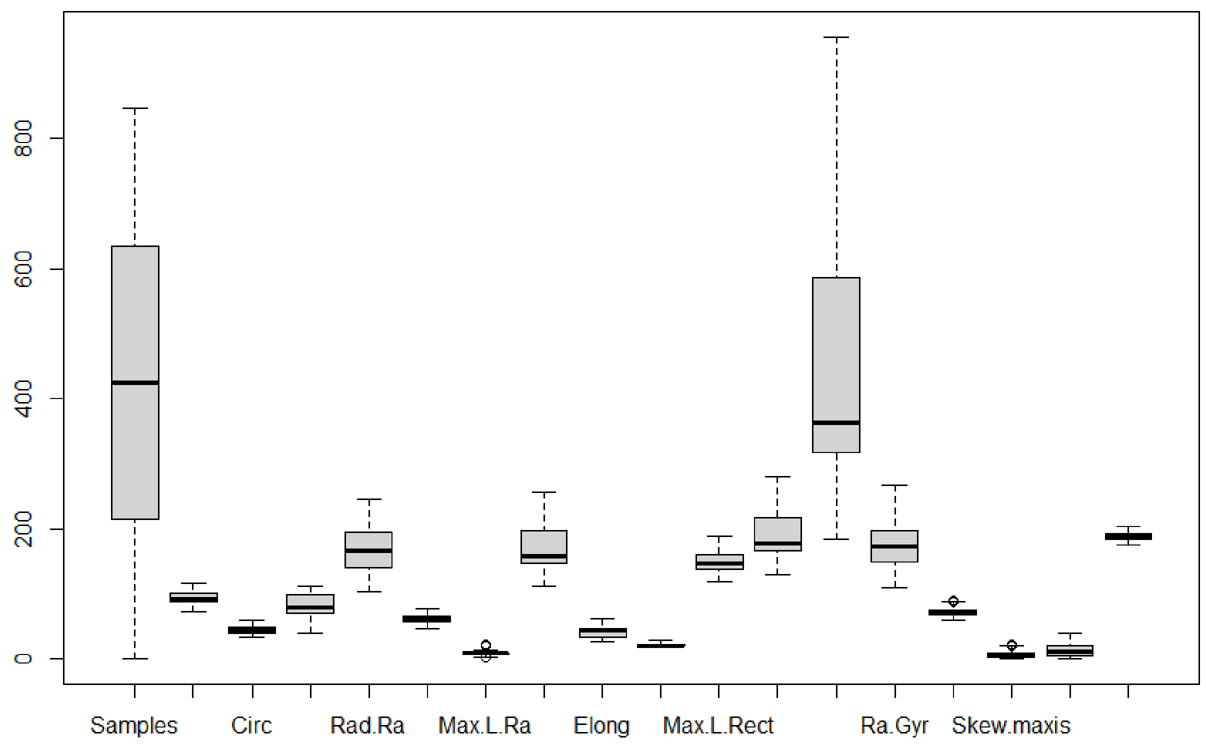


```r
#Detect and remove outliers
#These two lines detect and remove outliers from the 'vehicales_data_input' data frame.
#The apply functions applies the 'scale' function to each coloum of the "vehicals_data_input"
#data frame to calculate the z-scores , which are then used to identify the outliers
#The outliers are removed using logical indexing, and the result is stored in a new
#dataframe called 'vehicle_outliers_remove'
z_scores <- apply(vehicales_data_input,2,function(x) abs(scale(x,center = TRUE, scale = TRUE)))
vehical_outliers_remove <- vehicales_data_input[rowSums(abs(z_scores) < 3) == ncol(z_scores), ]

#These two lines displays the dimensions of the 'vehicales_outliers_remove' data frame and
#draw a box plot of the data
dim(vehical_outliers_remove)
boxplot(vehical_outliers_remove)
```

Data values display the outlier removal:



Data values display after the outlier removal:



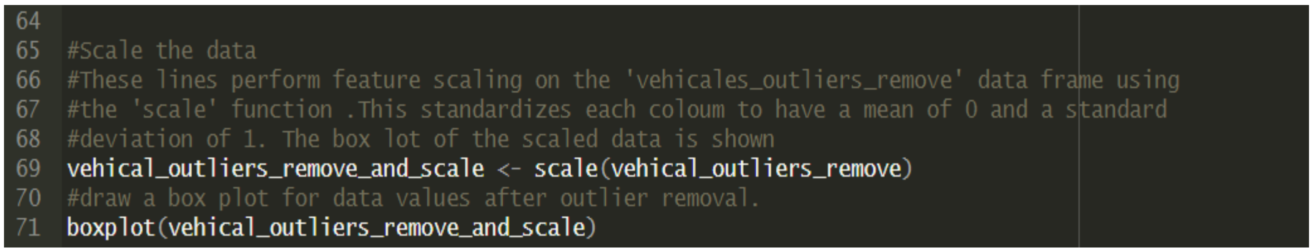
2. Scaling

In k-means clustering, scaling is a key pre-processing step since it can alter the clustering algorithm's efficacy and accuracy. Scaling input characteristics can be accomplished in two ways: standardization (using the R scale() function) and min-max scaling (using the R sapply() function). There are benefits and disadvantages to each technique on its own.

When applying k-means clustering, the scale() method normalizes the input characteristics to have a mean of 0 and a standard deviation of 1. This method is prone to outliers because of the way that outliers might change the size of the input features. Since the z-score method already gets rid of outliers in this case, the input features have been standardized using the scale() function.

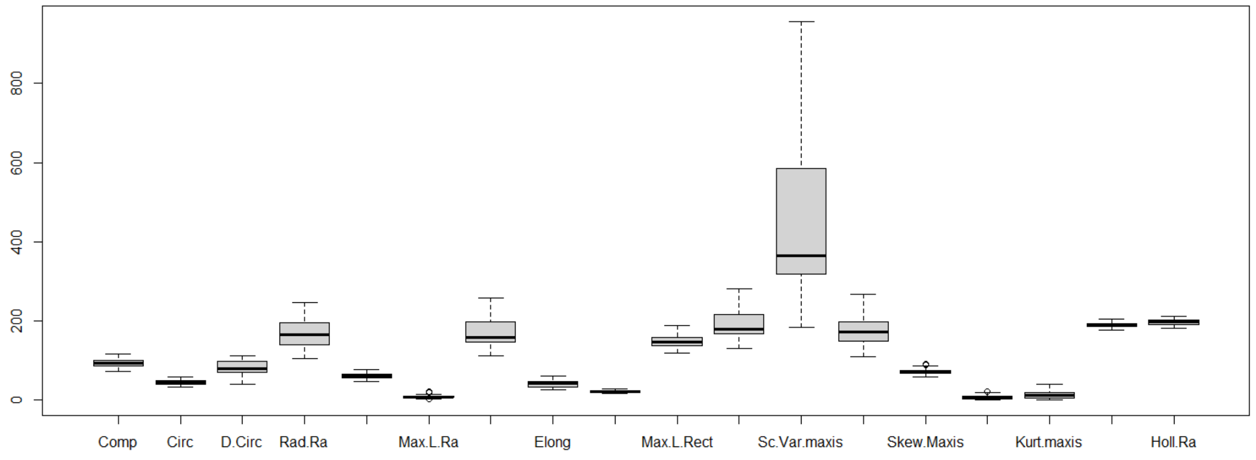
To preserve the distribution of the data and enhance its readability, min-max scaling limits the input features to a range of 0 to 1. This approach, meanwhile, can be sensitive to outliers and result in information loss if the input feature range is not well-defined.

The relevant code:

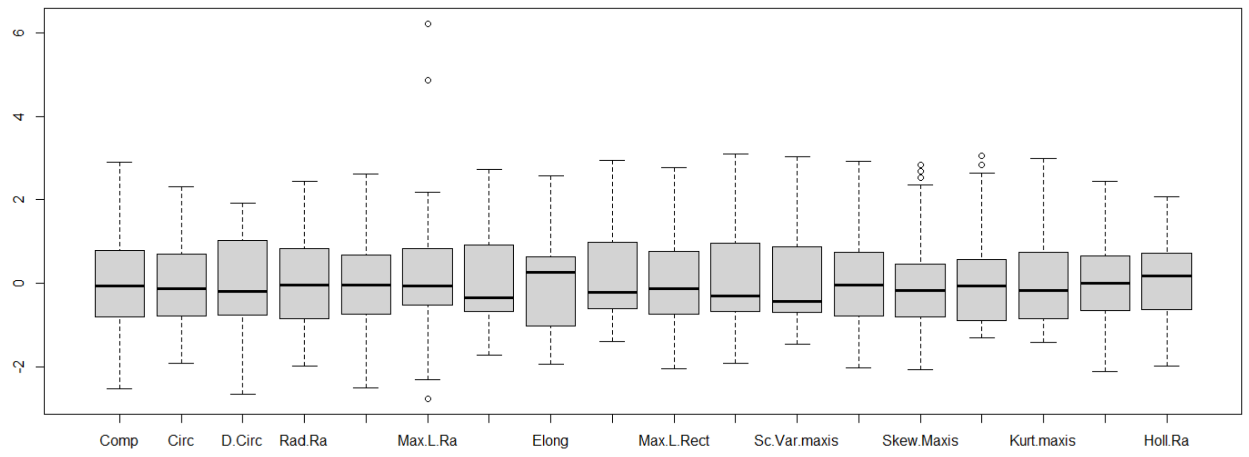


```

#Scale the data
#These lines perform feature scaling on the 'vehicales_outliers_remove' data frame using
#the 'scale' function .This standardizes each coloum to have a mean of 0 and a standard
#deviation of 1. The box lot of the scaled data is shown
vehical_outliers_remove_and_scale <- scale(vehical_outliers_remove)
#draw a box plot for data values after outlier removal.
boxplot(vehical_outliers_remove_and_scale)
```

Data values before the scaling:



Data values display after the scaling:



Data is ready for clustering once outliers have been removed and it has been scaled.

3. The order of scaling and outliers removal

Usually, it is suggested to eliminate outliers before scaling the data. The reason for this is that outliers can drastically change a dataset's mean and standard deviation, which are widely used in scaling approaches like Z-score scaling or Min-Max scaling. The scaling may become skewed if outliers are not removed before it is done.

1. Eliminate outliers from the dataset

2. Scale the data using appropriate scaling techniques.

This approach will ensure that the final dataset is free of outliers and appropriately scaled, which will improve the accuracy and performance of statistical analyses and machine learning models.

1.1.5 Number of cluster Centers Using Automated Tools

To determine the ideal number of cluster centers for k-means clustering, a variety of automated tools can be used:

- **NBclust**

  Installing and importing the package as explained in sections 1.1.1 and 1.1.2 is required before using the NBclust package to calculate the ideal number of clusters for k-means clustering.

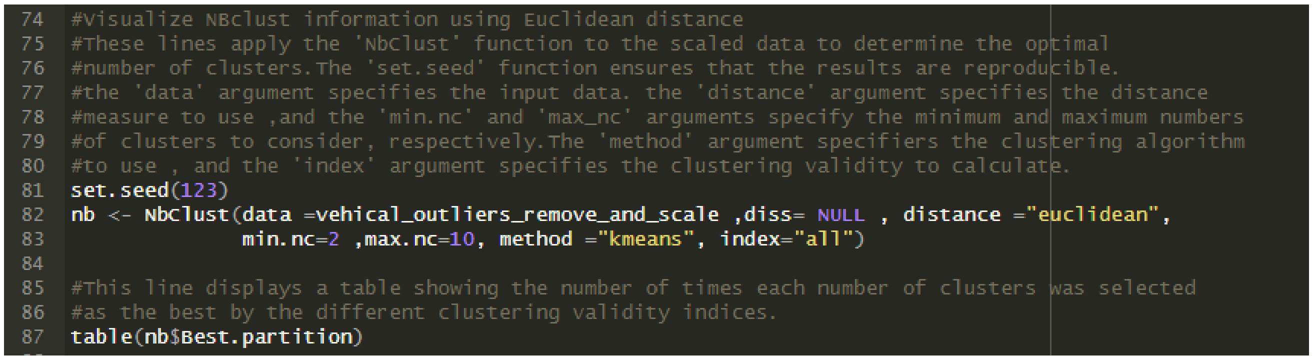
  A number of indices are provided by the R package NbClust to count the number of clusters in a given data collection. It comprises of 30 frequently used cluster analysis indices, including the Calinski-Harabasz index, gap statistic, and silhouette width. These indices can be used to determine the ideal number of clusters in the data set by comparing the values across various numbers of clusters. These indices are implemented by the NbClust function, which also provides the number of clusters that, in accordance with the majority rule, ought to be used. For automatic clustering analysis, it is a well-known R tool.

- **Using Euclidean distance**

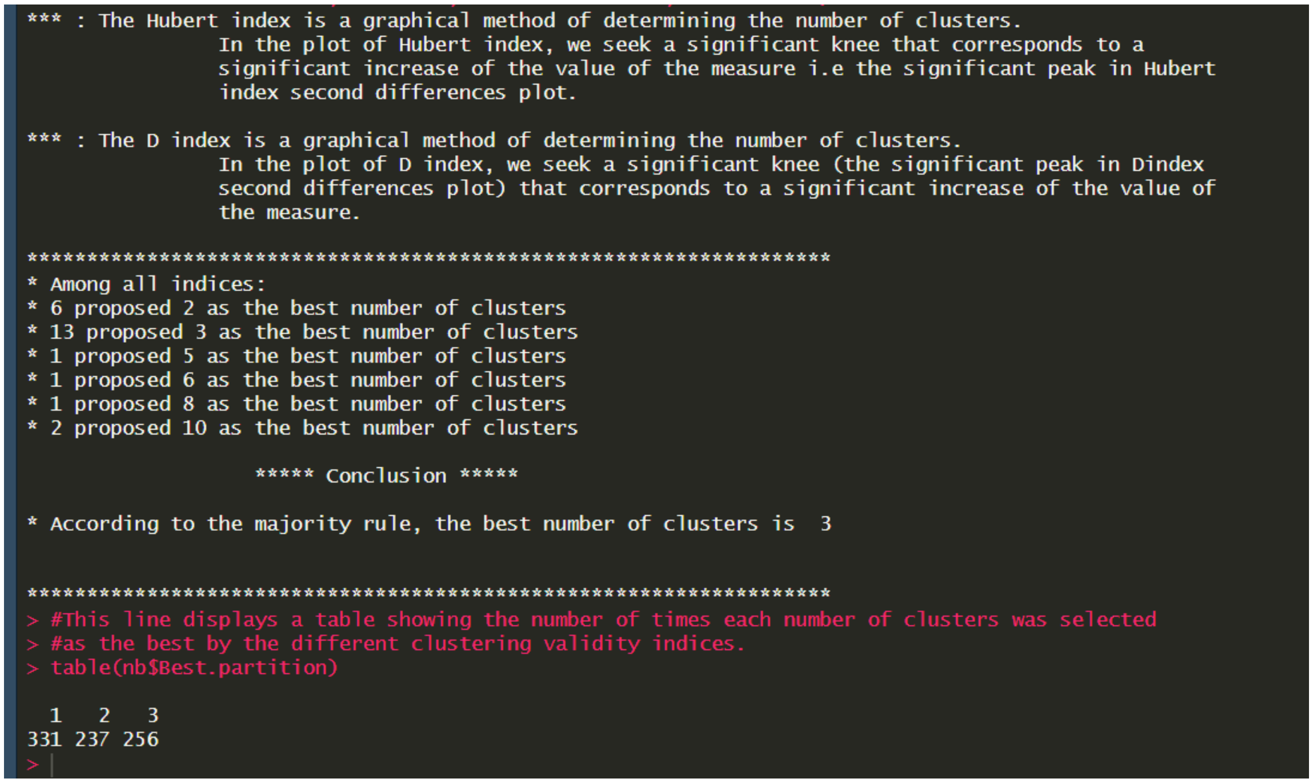
  The R package NBclust offers numerous automated methods for counting the number of clusters in a dataset. Using a variety of clustering methods and internal validation metrics, these tools suggested the ideal number of clusters for the data.

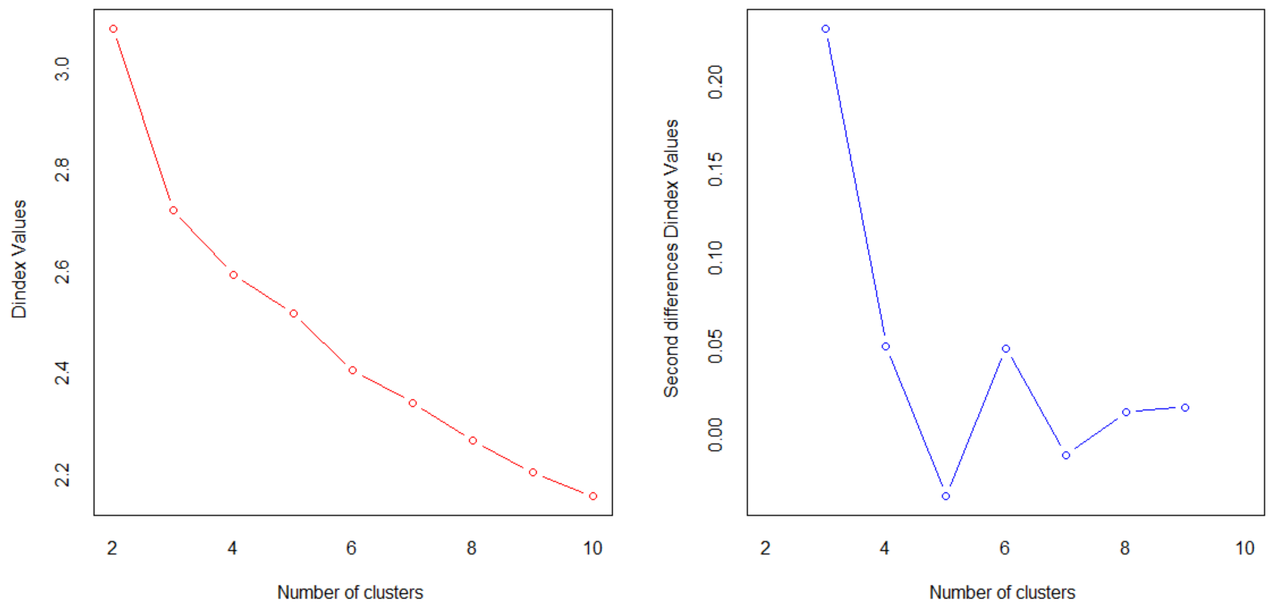
It's essential to understand the benefits and drawbacks of using automated tools like NBclust. One of the main advantages of automated systems is their capacity to quickly evaluate large datasets and provide a suggested number of clusters without the need for manual data investigation or in-depth topic knowledge. Additionally, different automated programs may produce different results depending on the precise algorithms and validation techniques used. Therefore, it is recommended to compare the results of different automated techniques and to enhance these results with domain knowledge and data visualization.

The relevant code (Euclidean distance):


```
#Visualize NBclust information using Euclidean distance
#These lines apply the 'NbClust' function to the scaled data to determine the optimal
#number of clusters.The 'set.seed' function ensures that the results are reproducible.
#the 'data' argument specifies the input data. the 'distance' argument specifies the distance
#measure to use ,and the 'min.nc' and 'max_nc' arguments specify the minimum and maximum numbers
#of clusters to consider, respectively.The 'method' argument specifiers the clustering algorithm
#to use , and the 'index' argument specifies the clustering validity to calculate.
set.seed(123)
nb <- NbClust(data =vehical_outliers_remove_and_scale ,diss= NULL , distance ="euclidean",
              min.nc=2 ,max.nc=10, method ="kmeans", index="all")

#This line displays a table showing the number of times each number of clusters was selected
#as the best by the different clustering validity indices.
table(nb$Best.partition)
```

The relevant output (Using the Euclidean Distance):

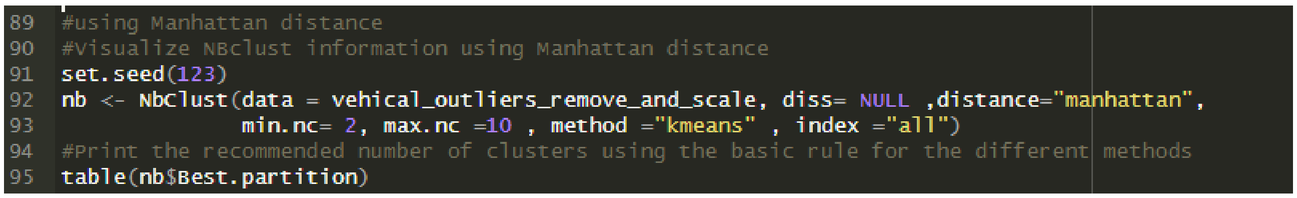

```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index 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
* 13 proposed 3 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 2 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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


*******************************************************************
> #This line displays a table showing the number of times each number of clusters was selected
> #as the best by the different clustering validity indices.
> table(nb$Best.partition)

  1   2   3 
331 237 256 
> 
```



According to the above results, NBclust suggests that 3 clusters are the optimum number of clusters when using the Euclidean Distance.

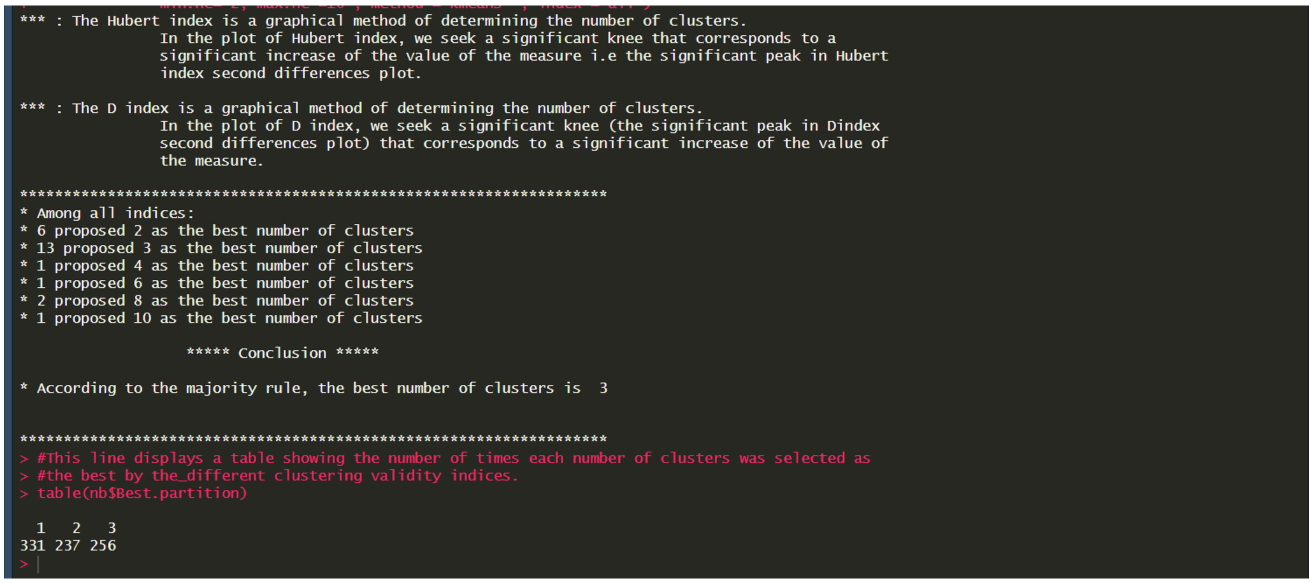
- **Using Manhattan distance**

The relevant code (Using the Manhattan Distance)

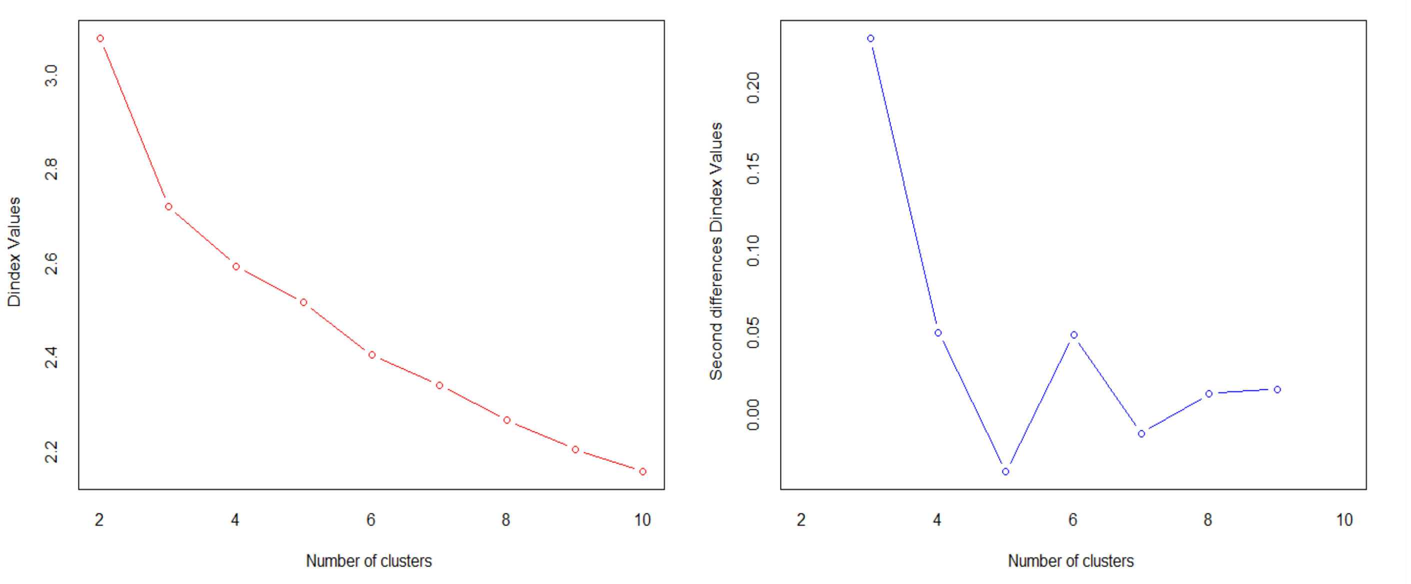


```
#using Manhattan distance
#visualize NBclust information using Manhattan distance
set.seed(123)
nb <- NbClust(data = vehical_outliers_remove_and_scale, diss= NULL ,distance="manhattan",
              min.nc= 2, max.nc =10 , method ="kmeans" , index ="all")
#Print the recommended number of clusters using the basic rule for the different methods
table(nb$Best.partition)
```

The relevant output (Using the Manhattan Distance)



```
*** : 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
* 13 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 2 proposed 8 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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


*******************************************************************
> #This line displays a table showing the number of times each number of clusters was selected as
> #the best by the_different clustering validity indices.
> table(nb$Best.partition)

  1   2   3
331 237 256
>
```

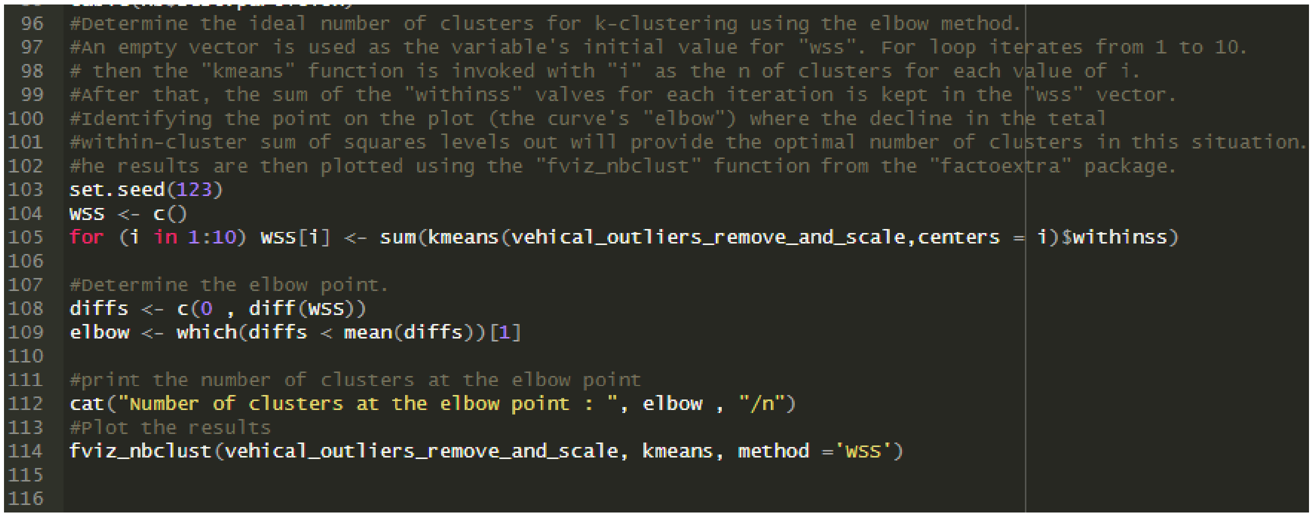


According to the above results, NBclust suggests that 3 clusters are the optimum number of clusters when using the Manhattan Distance.

Elbow

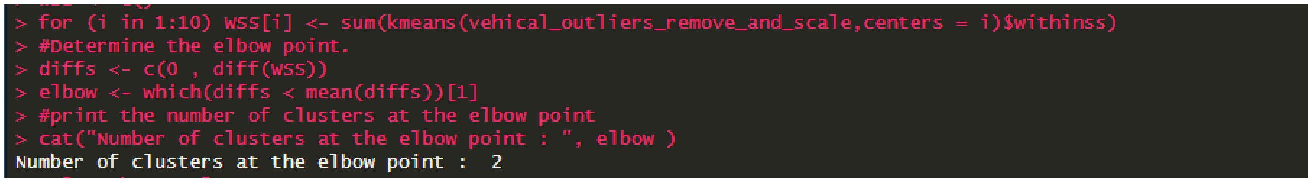
The ideal number of clusters for k-means clustering is determined using the elbow approach. The idea is to graph the overall within-cluster sum of squares (WSS) versus the cluster count (K). Each data point's squared Euclidean distance from its allocated centroid is added to determine the overall WSS. The WSS tends to decrease as the number of clusters rises because the data points cluster more closely around their allocated centroids. The drop in WSS does, however, tend to level off after a certain number of clusters, giving the plot an "elbow" shape. The "elbow" of the plot is typically where the ideal number of clusters is chosen.

The relevant code:



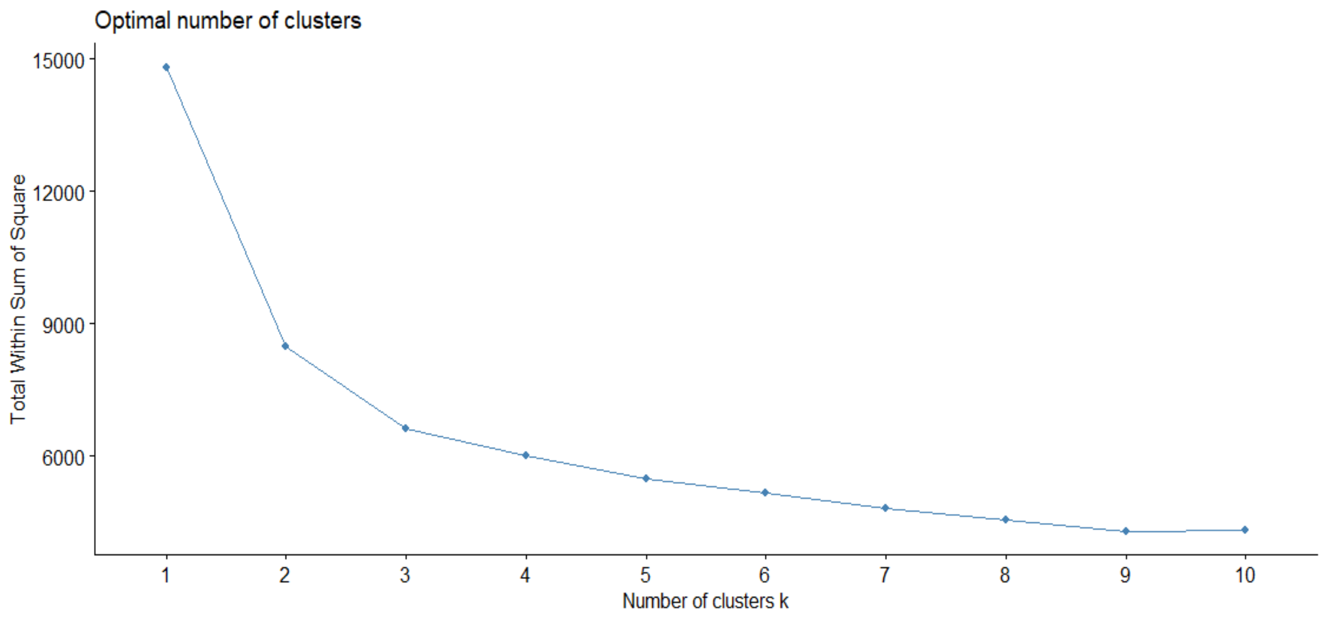
```
#Determine the ideal number of clusters for k-clustering using the elbow method.
#An empty vector is used as the variable's initial value for "wss". For loop iterates from 1 to 10.
# then the "kmeans" function is invoked with "i" as the n of clusters for each value of i.
#After that, the sum of the "withinss" valves for each iteration is kept in the "wss" vector.
#Identifying the point on the plot (the curve's "elbow") where the decline in the tetal
#within-cluster sum of squares levels out will provide the optimal number of clusters in this situation.
#he results are then plotted using the "fviz_nbclust" function from the "factoextra" package.
set.seed(123)
wss <- c()
for (i in 1:10) wss[i] <- sum(kmeans(vehical_outliers_remove_and_scale,centers = i)$withinss)

#Determine the elbow point.
diffs <- c(0 , diff(wss))
elbow <- which(diffs < mean(diffs))[1]

#print the number of clusters at the elbow point
cat("Number of clusters at the elbow point : ", elbow , "/n")
#Plot the results
fviz_nbclust(vehical_outliers_remove_and_scale, kmeans, method ='wss')
```

The relevant Output:



```
> for (i in 1:10) wss[i] <- sum(kmeans(vehical_outliers_remove_and_scale,centers = i)$withinss)
> #Determine the elbow point.
> diffs <- c(0 , diff(wss))
> elbow <- which(diffs < mean(diffs))[1]
> #print the number of clusters at the elbow point
> cat("Number of clusters at the elbow point : ", elbow )
Number of clusters at the elbow point :  2
```

Optimal number of clusters:



It is important to identify the elbow point on the plot because here is where the rate of WSS (Within Cluster Sum of Squares) decrease and begins to level off. The ideal number of clusters is considered to be the number of clusters corresponding to the elbow point.

According to the above results, Elbow method suggests that 2 clusters are the optimum number of clusters.

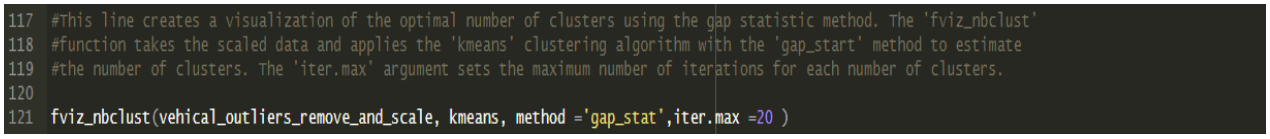
- **Gap statistics**

  Finding the number of clusters in a dataset can be done by using the gap statistics method. The overall within-cluster variation for different k (number of clusters) values is compared to the variation expected for a random data set with no clustering.
  The value of k that causes the observed within-cluster variance to deviate the most from the expected variation is used to calculate the optimal number cluster.

  The Gap statistic is calculated as the difference between the expected log of within-cluster variation and the observed log of within-cluster variation.

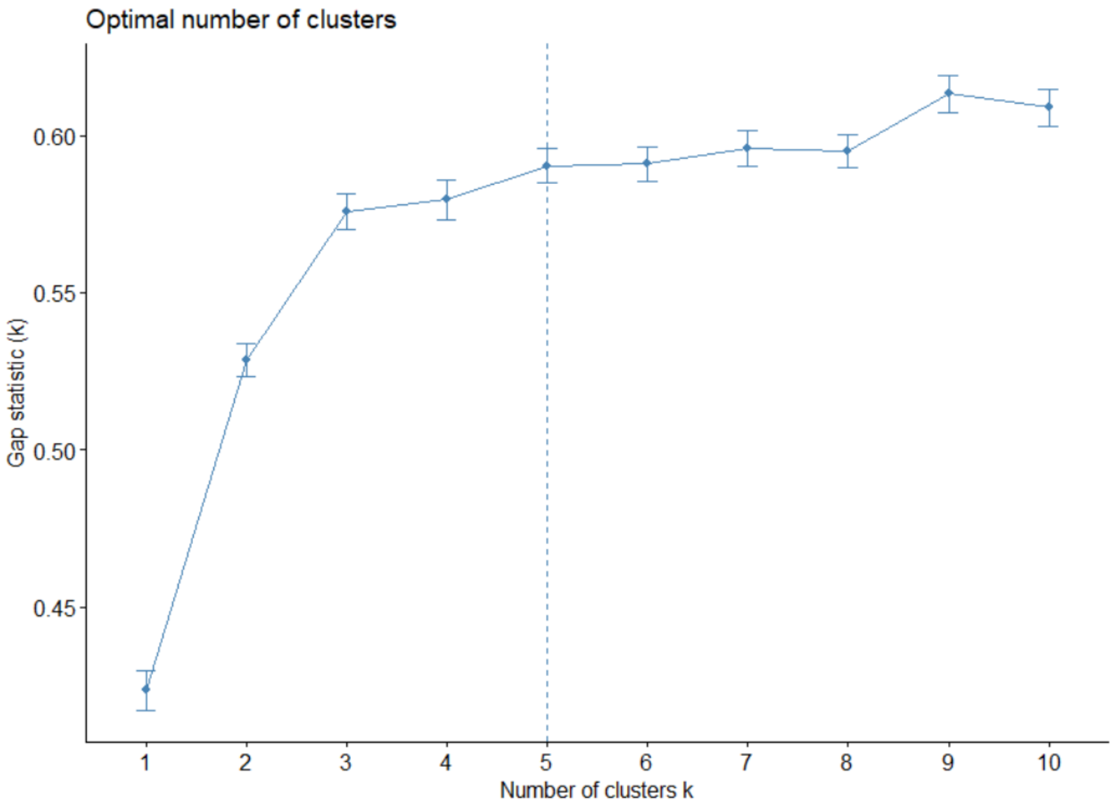
The data are more accurately clustered into k-clusters, the larger the Gap statistic.

The relevant code:



```
#This line creates a visualization of the optimal number of clusters using the gap statistic method. The 'fviz_nbclust'
#function takes the scaled data and applies the 'kmeans' clustering algorithm with the 'gap_start' method to estimate
#the number of clusters. The 'iter.max' argument sets the maximum number of iterations for each number of clusters.

fviz_nbclust(vehical_outliers_remove_and_scale, kmeans, method ='gap_stat',iter.max =20 )
```

The relevant output:



- **Silhouette**

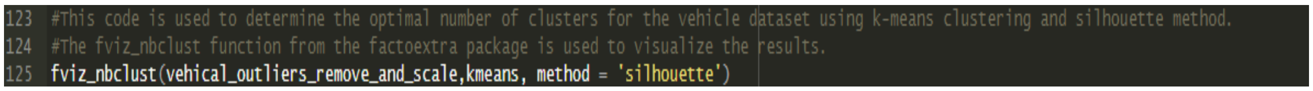
A method for analyzing a clustering solution's quality is silhouette analysis. It determines the cohesion (similarity) to its own cluster, in comparison to other clusters (separation).

The silhouette coefficient for each data point varies from -1 to 1, where a value of 1 denotes a point that is very well matched to its own cluster but poorly matched to nearby clusters, a value of -1 denotes the opposite, and a value close to 0 denotes a point that is in between clusters..

By identifying the number of clusters with the highest average silhouette coefficient across all data points, silhouette analysis can help in determining the best number of clusters.

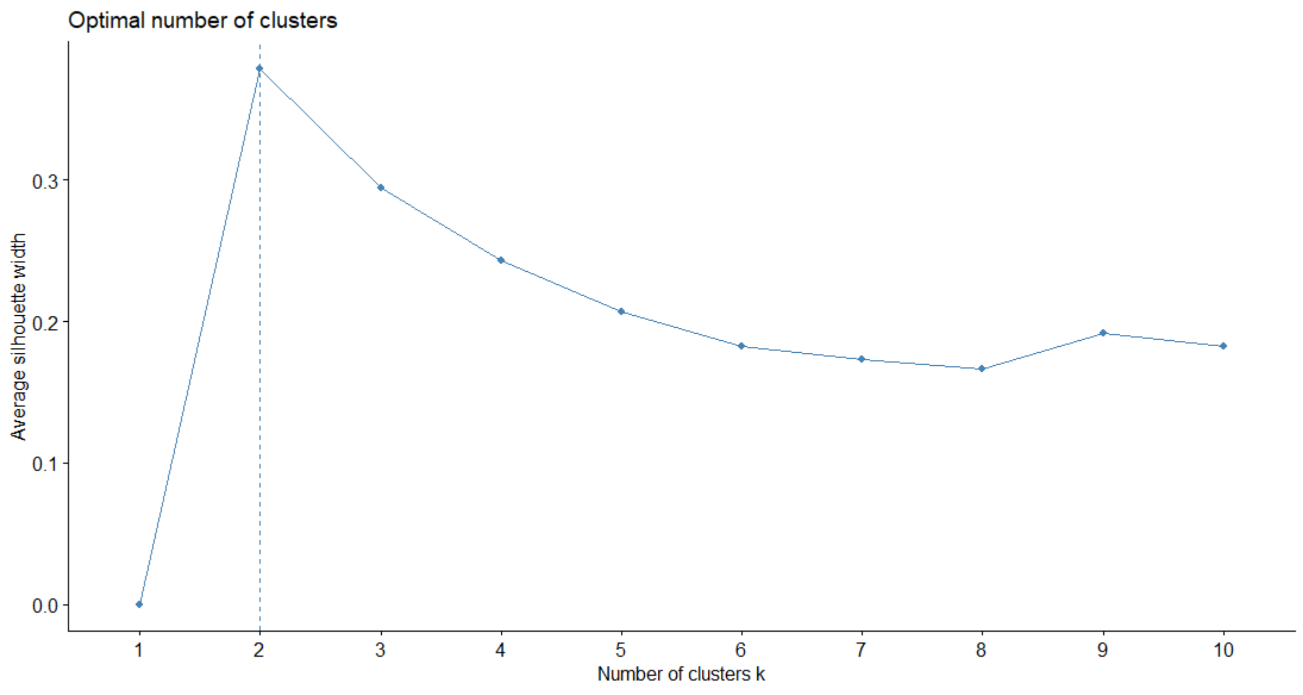
When the average silhouette coefficient is large, the clustering solution is suitable; however, when it is low, the data points are poorly clustered and might benefit from a different number of clusters.

The relevant code:



```
#This code is used to determine the optimal number of clusters for the vehicle dataset using k-means clustering and silhouette method.
#The fviz_nbclust function from the factoextra package is used to visualize the results.
fviz_nbclust(vehical_outliers_remove_and_scale,kmeans, method = 'silhouette')
```

The relevant output:



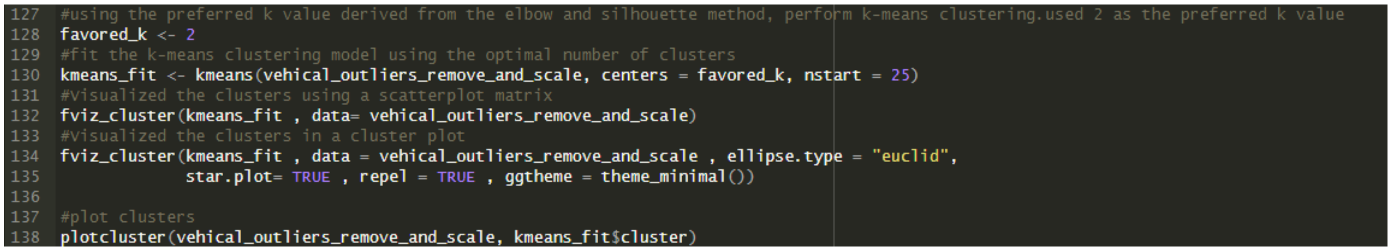
According to the above outcome, the Silhouette approach suggests that the optimal number of clusters is 2. The resulting plot shows that 2 clusters optimize the average Silhouette width, with 3 clusters being the second optimal number of clusters.

1.1.6 The k-means Clustering Investigation & Evaluation

As the Silhouette and Elbow methods both suggest that the optimal number of clusters is two, it is decided that two is the optimal k value.

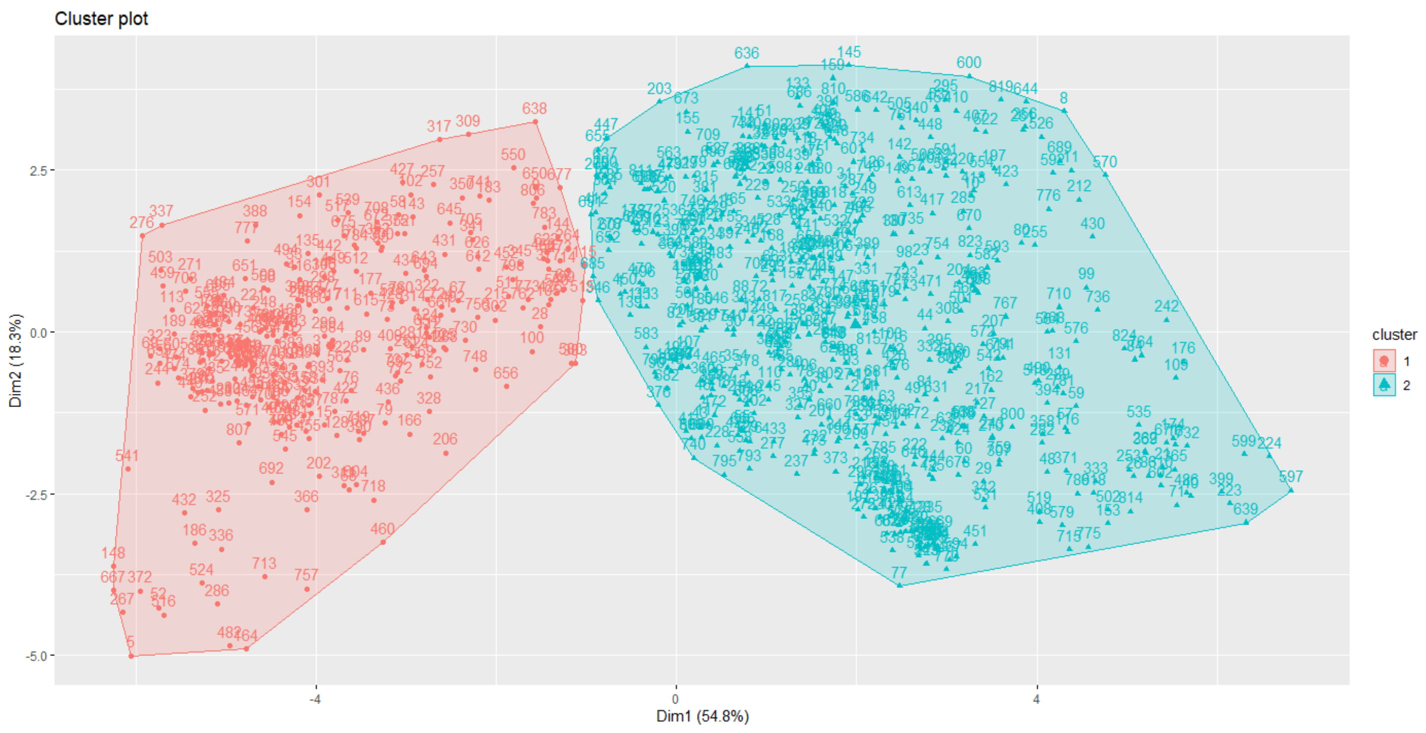
Apply k-means clustering on the optimal k values obtained from the automated methods.

The relevant code:

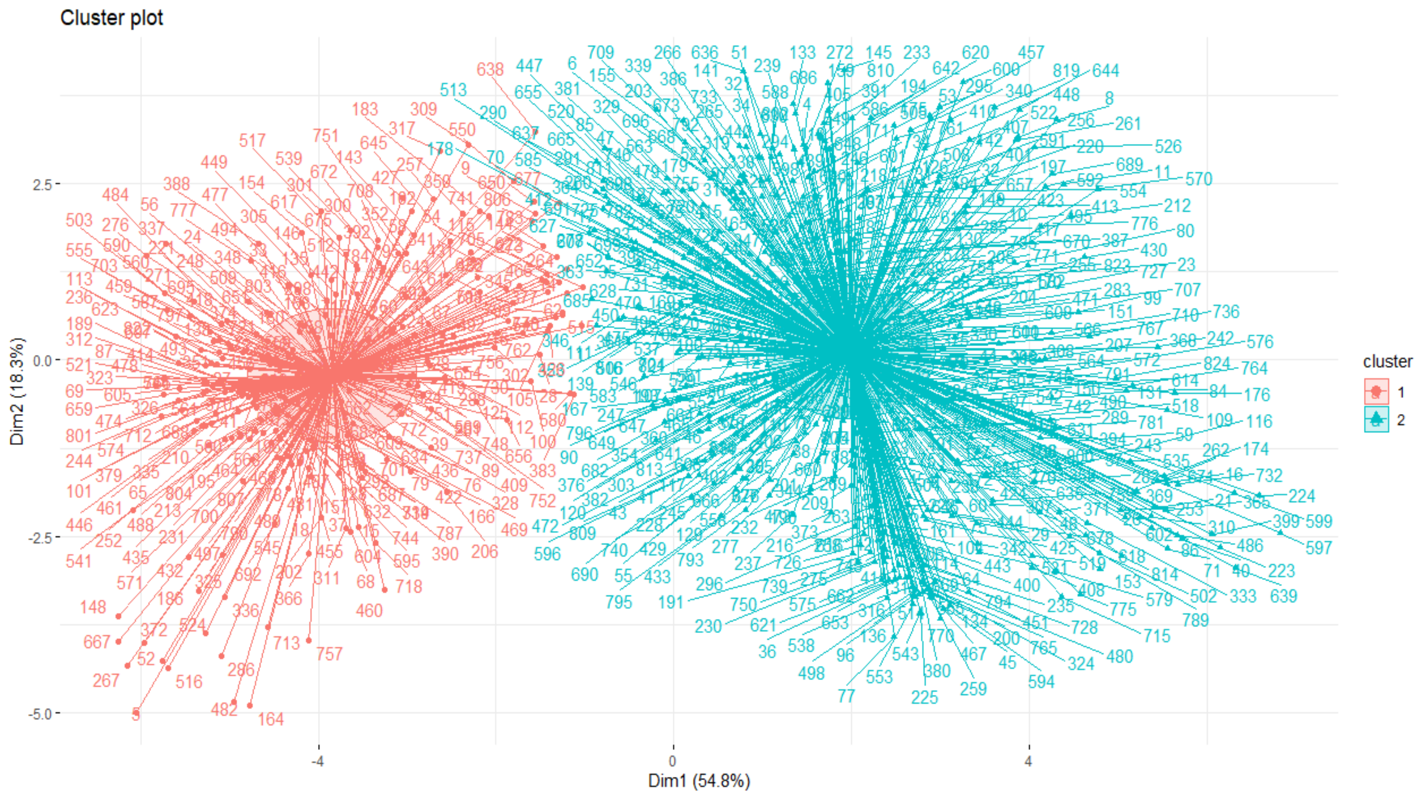


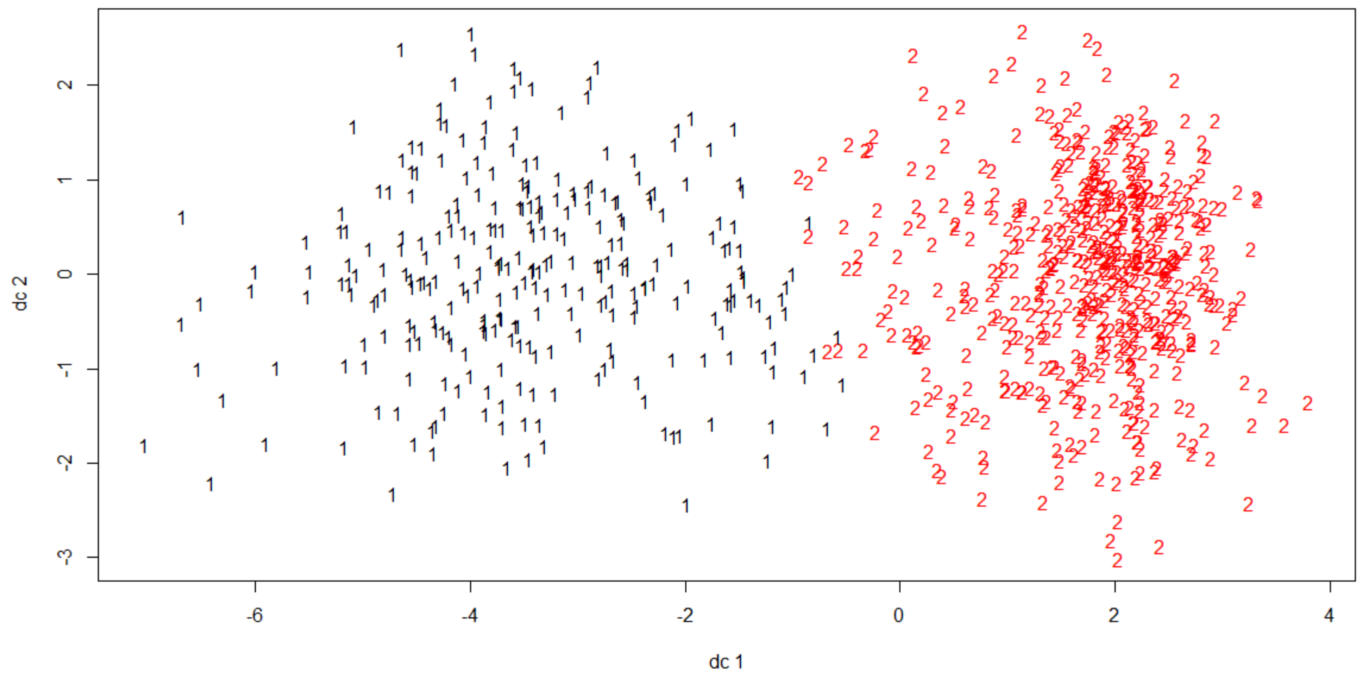
```
#using the preferred k value derived from the elbow and silhouette method, perform k-means clustering.used 2 as the preferred k value
favored_k <- 2
#fit the k-means clustering model using the optimal number of clusters
kmeans_fit <- kmeans(vehical_outliers_remove_and_scale, centers = favored_k, nstart = 25)
#Visualized the clusters using a scatterplot matrix
fviz_cluster(kmeans_fit , data= vehical_outliers_remove_and_scale)
#Visualized the clusters in a cluster plot
fviz_cluster(kmeans_fit , data = vehical_outliers_remove_and_scale , ellipse.type = "euclid",
             star.plot= TRUE , repel = TRUE , ggtheme = theme_minimal())

#plot clusters
plotcluster(vehical_outliers_remove_and_scale, kmeans_fit$cluster)
```

The relevant output:



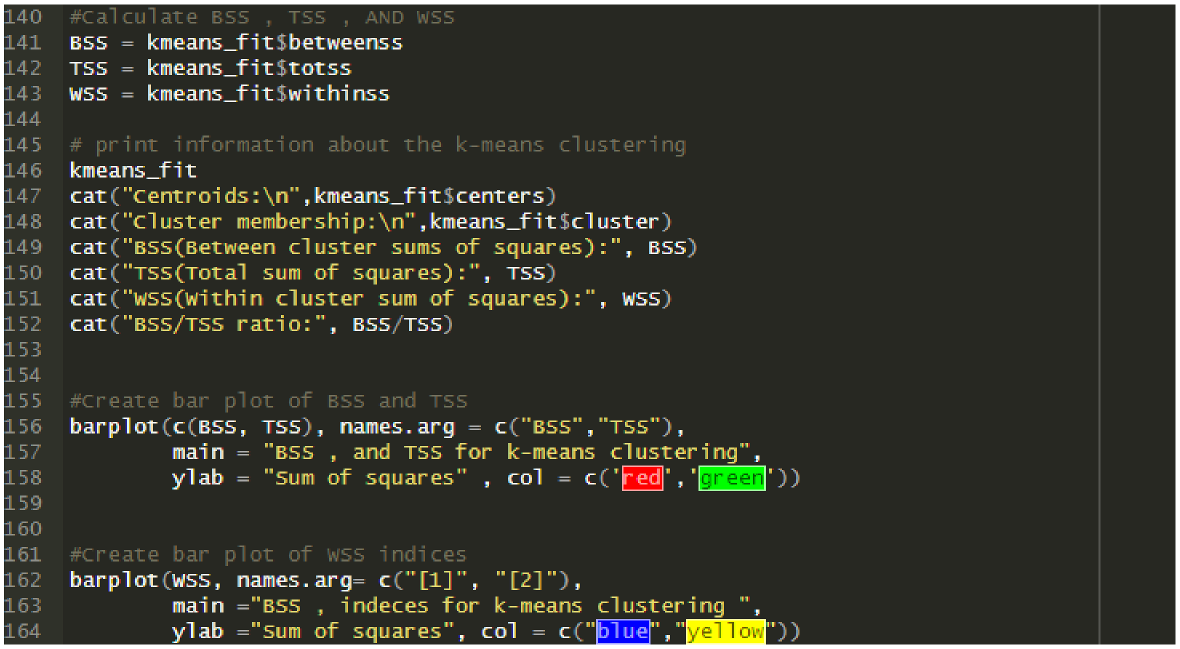
Cluster plot:





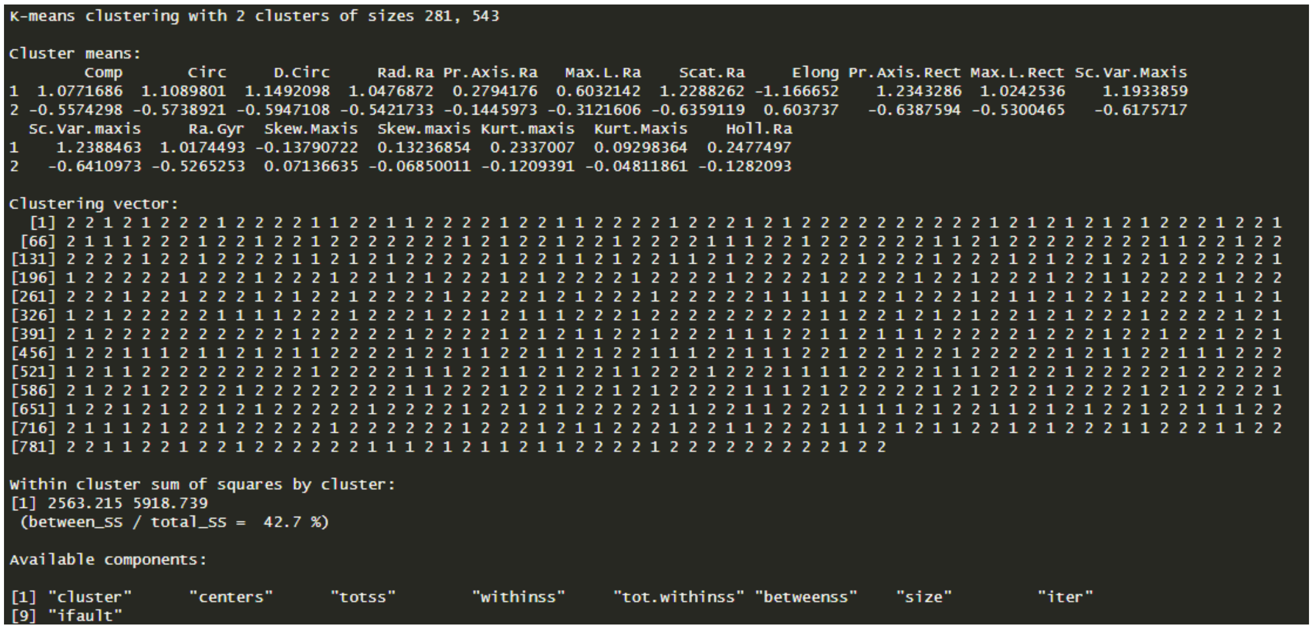
The information for the centroids, cluster vector (cluster membership), ratio of Between Cluster Sums of Squares (BSS) over Total Sum of Squares (TSS) - (BSS/TSS), calculation and illustration of the Between Cluster Sums of Squares (BSS) and Within Cluster Sums of Squares (WSS) indices (internal evaluation metrics), are all displayed below.

The relevant code:

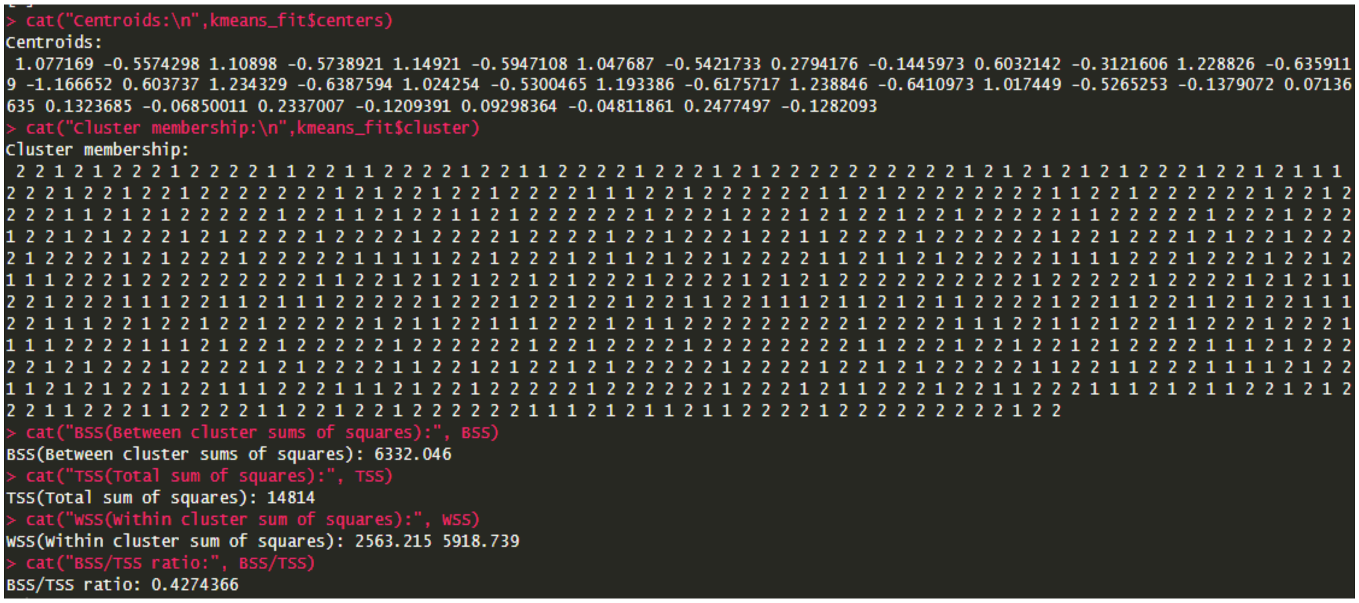


```r
#Calculate BSS , TSS , AND WSS
BSS = kmeans_fit$betweenss
TSS = kmeans_fit$totss
WSS = kmeans_fit$withinss

# print information about the k-means clustering
kmeans_fit
cat("Centroids:\n",kmeans_fit$centers)
cat("Cluster membership:\n",kmeans_fit$cluster)
cat("BSS(Between cluster sums of squares):", BSS)
cat("TSS(Total sum of squares):", TSS)
cat("WSS(Within cluster sum of squares):", WSS)
cat("BSS/TSS ratio:", BSS/TSS)


#Create bar plot of BSS and TSS
barplot(c(BSS, TSS), names.arg = c("BSS","TSS"),
        main = "BSS , and TSS for k-means clustering",
        ylab = "Sum of squares" , col = c('red','green'))


#Create bar plot of WSS indices
barplot(WSS, names.arg= c("[1]", "[2]"),
        main ="BSS , indeces for k-means clustering ",
        ylab ="Sum of squares", col = c("blue","yellow"))
```

The relevant output:

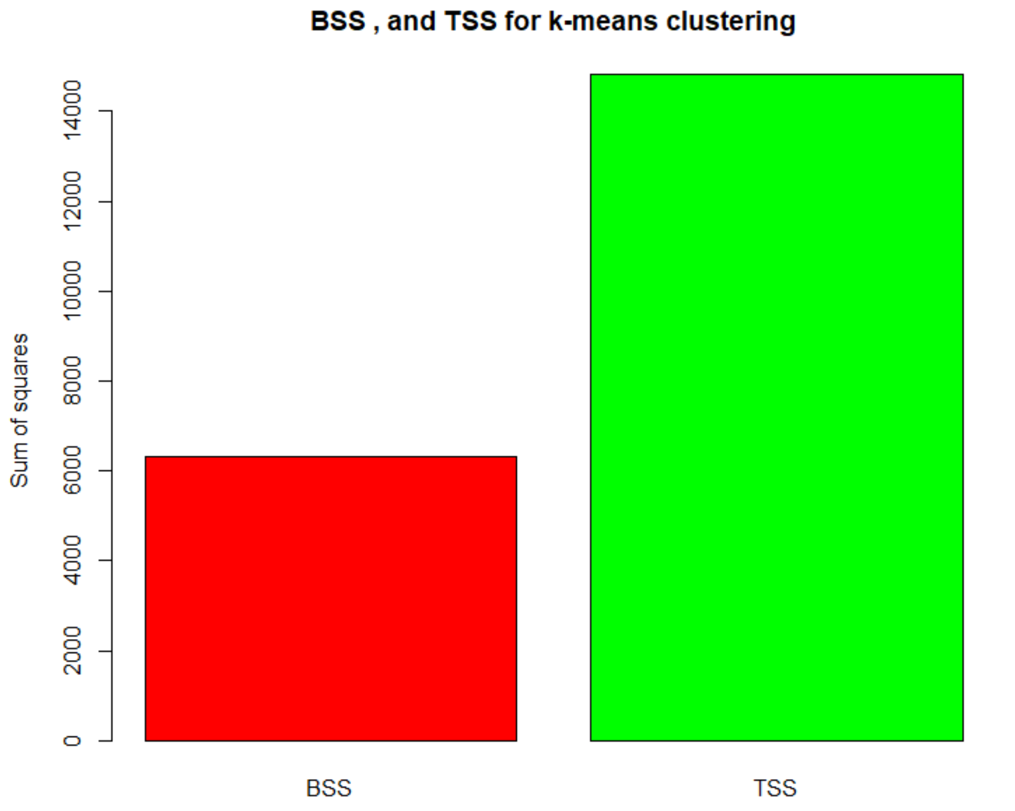


```
K-means clustering with 2 clusters of sizes 281, 543

Cluster means:
        Comp       Circ     D.Circ     Rad.Ra Pr.Axis.Ra   Max.L.Ra    Scat.Ra     Elong Pr.Axis.Rect Max.L.Rect Sc.Var.Maxis
1  1.0771686  1.1089801  1.1492098  1.0476872  0.2794176  0.6032142  1.2288262 -1.166652    1.2343286  1.0242536    1.1933859
2 -0.5574298 -0.5738921 -0.5947108 -0.5421733 -0.1445973 -0.3121606 -0.6359119  0.603737   -0.6387594 -0.5300465   -0.6175717
  Sc.Var.maxis     Ra.Gyr  Skew.Maxis  Skew.maxis Kurt.maxis  Kurt.Maxis    Holl.Ra
1    1.2388463  1.0174493 -0.13790722  0.13236854  0.2337007  0.09298364  0.2477497
2   -0.6410973 -0.5265253  0.07136635 -0.06850011 -0.1209391 -0.04811861 -0.1282093

Clustering vector:
  [1] 2 2 1 2 1 2 2 2 1 2 2 2 2 1 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 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 2 1
 [66] 2 1 1 1 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 1 1 1 2 2 1 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2
[131] 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 1 2 1 2 2 2 2 2 1 2 2 1 1 2 1 2 2 1 1 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 2 1
[196] 1 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1 2 1 2 2 2 1 2 1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 1 2 2 2 1 2 2 1 1 2 2 2 2 1 2 2 2
[261] 2 2 2 1 2 2 1 2 2 2 1 2 1 2 2 1 2 2 2 2 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 1 1 1 1 1 2 2 1 2 2 2 1 2 1 1 2 1 2 2 1 2 2 2 2 1 1 2 1
[326] 1 2 1 2 2 2 2 2 1 1 1 1 2 2 2 1 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1
[391] 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 1 2 1 1 2 2 1 2 2 2 1 1 1 2 2 1 1 2 1 1 1 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 1 2 2 1
[456] 1 2 2 1 1 1 2 1 1 2 1 2 1 1 2 2 2 2 1 2 2 1 1 2 2 1 1 2 1 2 2 1 1 1 2 2 1 1 1 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 2 1 1 2 2 1 1 1 2 2 2
[521] 1 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 1 2 2 1 1 2 1 2 2 1 1 2 2 2 1 2 2 2 1 1 1 1 2 2 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2
[586] 2 1 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 2 2 1 2 1 2 2 2 2 1 1 1 2 1 2 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1 2 2 2 2 1
[651] 1 2 2 1 2 1 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 1 1 2 2 1 1 2 2 2 1 1 1 1 2 1 2 2 1 1 2 1 2 1 2 2 1 2 2 1 1 1 2 2
[716] 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 1 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 1 2 2 2 1 1 2 2
[781] 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2 1 1 1 2 1 2 1 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2

Within cluster sum of squares by cluster:
[1] 2563.215 5918.739
 (between_SS / total_SS =  42.7 %)

Available components:

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

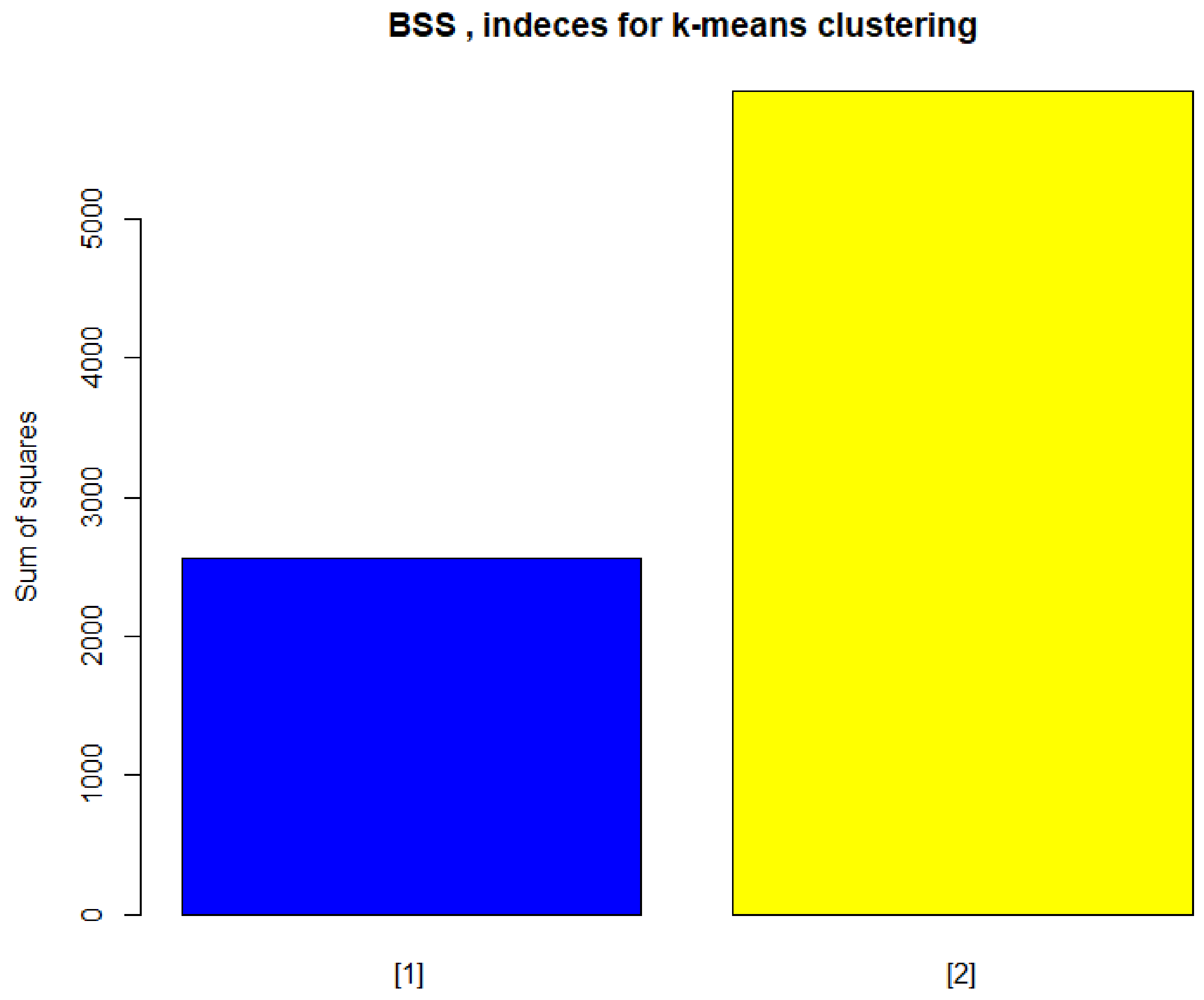


```
> cat("Centroids:\n",kmeans_fit$centers)
Centroids:
 1.077169 -0.5574298 1.10898 -0.5738921 1.14921 -0.5947108 1.047687 -0.5421733 0.2794176 -0.1445973 0.6032142 -0.3121606 1.228826 -0.635911
9 -1.166652 0.603737 1.234329 -0.6387594 1.024254 -0.5300465 1.193386 -0.6175717 1.238846 -0.6410973 1.017449 -0.5265253 -0.1379072 0.07136
635 0.1323685 -0.06850011 0.2337007 -0.1209391 0.09298364 -0.04811861 0.2477497 -0.1282093
> cat("Cluster membership:\n",kmeans_fit$cluster)
Cluster membership:
 2 2 1 2 1 2 2 2 1 2 2 2 2 1 1 2 2 1 1 2 2 2 2 1 2 2 1 1 2 2 2 2 1 2 2 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 2 1 2 1 1 1
2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 1 1 1 2 2 1 2 1 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2 2 1 2 2 1 2
2 2 2 1 1 2 1 2 1 2 2 2 2 2 1 2 2 1 1 2 1 2 2 1 1 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 1 1 2 2 2 2 2 1 2 2 2 1 2 2 2
1 2 2 1 2 1 2 2 2 1 2 1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 1 2 2 2 1 2 2 1 1 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 1 2 1 2 2 1 2 2 2
2 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 1 1 1 1 1 2 2 1 2 2 2 1 2 1 1 2 1 2 2 1 2 2 2 2 1 1 2 1 1 2 1 2 2 2 2 2 1 1 1 1 2 2 2 1 2 2 2 1 2 2 1 2
1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 1 2 1 1
2 2 1 2 2 2 1 1 1 2 2 1 1 2 1 1 1 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 1 2 2 1 1 2 2 1 1 1 2 1 1 2 1 2 1 1 2 2 2 2 1 2 2 1 1 2 2 1 1 2 1 2 2 1 1 1
2 2 1 1 1 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 2 1 1 2 2 1 1 1 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 1 2 2 1 1 2 1 2 2 1 1 2 2 2 1 2 2 2 1
1 1 1 2 2 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 2 2 1 2 1 2 2 2 2 1 1 1 2 1 2 2 2
2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1 2 2 2 2 1 1 2 2 1 2 1 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 1 1 2 2 1 1 2 2 2 1 1 1 1 2 1 2 2
1 1 2 1 2 1 2 2 1 2 2 1 1 1 2 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 1 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 1 2 2 2 1 1 2 2 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2 1 1 1 2 1 2 1 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2
> cat("BSS(Between cluster sums of squares):", BSS)
BSS(Between cluster sums of squares): 6332.046
> cat("TSS(Total sum of squares):", TSS)
TSS(Total sum of squares): 14814
> cat("WSS(Within cluster sum of squares):", WSS)
WSS(Within cluster sum of squares): 2563.215 5918.739
> cat("BSS/TSS ratio:", BSS/TSS)
BSS/TSS ratio: 0.4274366
```

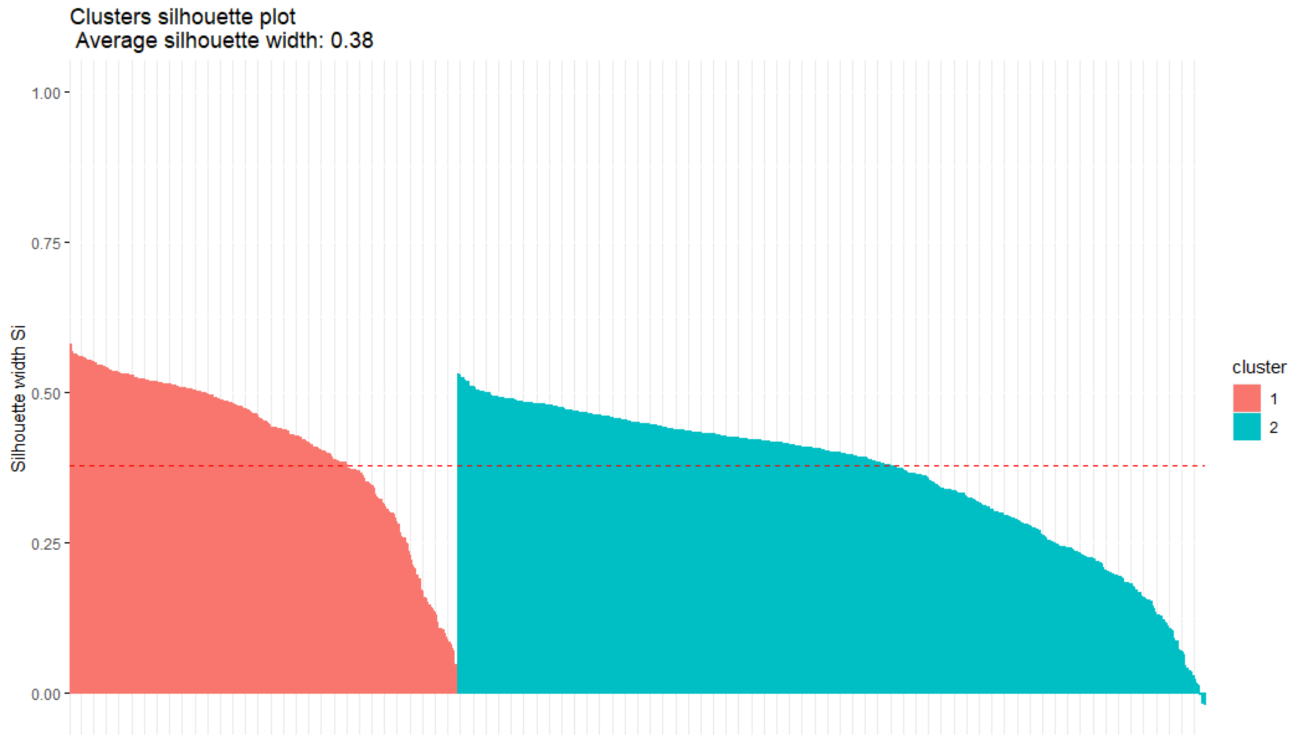
BSS, and TSS for k-means clustering



WSS indeces for k-mean clustering:



Another internal evaluation measure used in clustering studies to display the silhouette coefficient of each data point graphically is the silhouette plot. The equivalent value of a data point to its own cluster is compared to the equivalent values of other clusters via the silhouette coefficient. Each data point in a silhouette plot is shown as a vertical line, with the height of the line representing the silhouette coefficient for that particular point. The graph might be used to evaluate how well the clustering was done and spot any issues with cluster assignment.



A value closer to 1 denotes greater clustering. The range of the silhouette width is from -1 to 1. The clusters are moderately well-separated and considered acceptable with an average Silhouette width score of 0.38, which shows that they still have potential for growth.

However, it is not a very high value, indicating that some data points may be assigned to the wrong cluster and that there may be some cluster overlap between clusters. In general, the context and specific scenario under consideration has an impact on how the silhouette width value is perceived.

It's also important to note that silhouette width is only one of several metrics used to evaluate the effectiveness of a clustering approach. It is typically advised to employ a number of metrics and visualizations to acquire a broad view of the clustering results.

1.2. Second Subtask

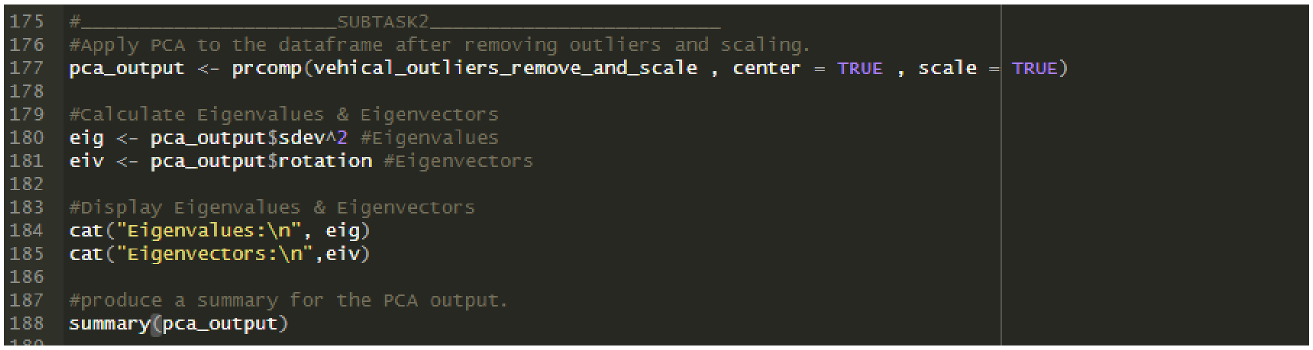
Principal Component Analysis (PCA) is an abbreviation for Principal Component Analysis, a statistical method for reducing the dimensionality of a dataset while maintaining as much information as possible. In other words, it is a technique for turning a high-dimensional dataset into a lower-dimensional space in which the new variables (referred to as primary components) are uncorrelated and represent the vast majority of the variability in the original data.

PCA can be helpful in a number of ways,

- Initially, it can minimize the complexity of a dataset, and make it easier to interpret and visualize.
- It can aid in finding patterns and linkages in data that might otherwise be hidden by the original variables.
- It may assist in eliminating noise and redundancy from data.

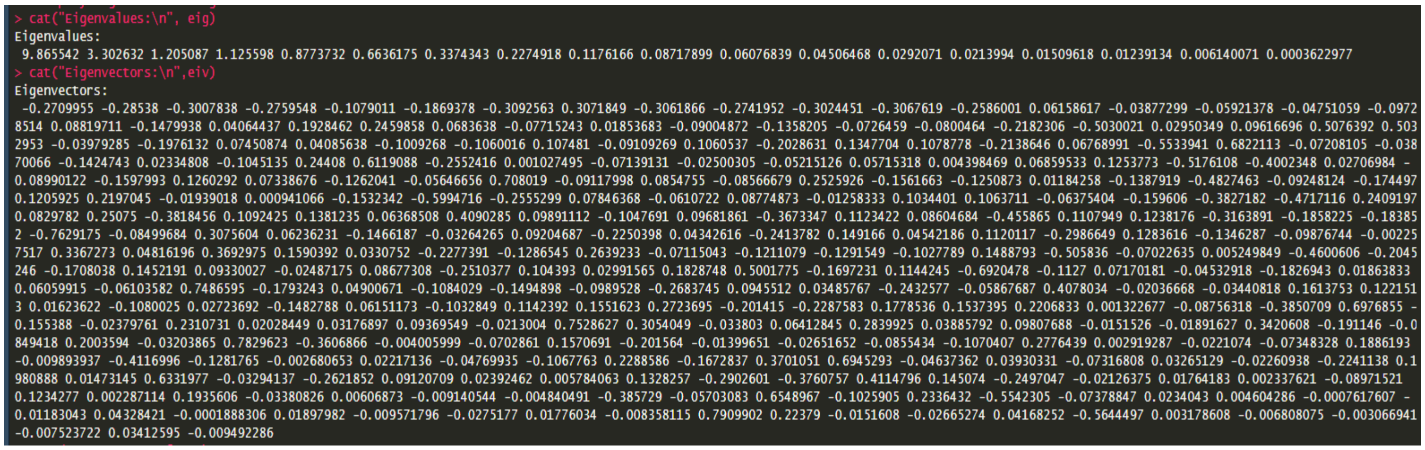
1.2.1 Apply PCA

The relevant code:

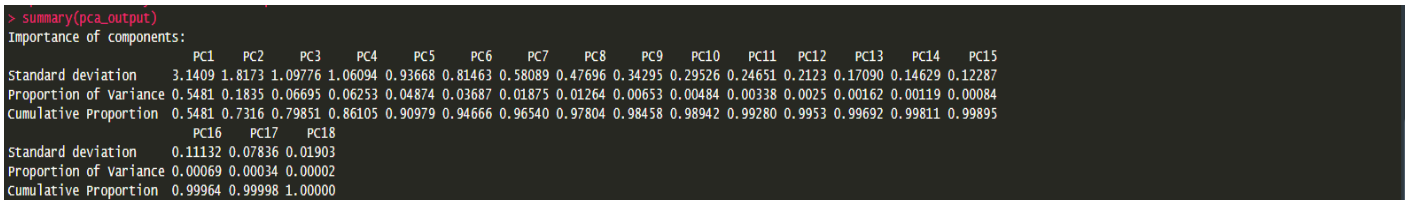


```r
#_______________________SUBTASK2__________________________
#Apply PCA to the dataframe after removing outliers and scaling.
pca_output <- prcomp(vehical_outliers_remove_and_scale , center = TRUE , scale = TRUE)

#Calculate Eigenvalues & Eigenvectors
eig <- pca_output$sdev^2 #Eigenvalues
eiv <- pca_output$rotation #Eigenvectors

#Display Eigenvalues & Eigenvectors
cat("Eigenvalues:\n", eig)
cat("Eigenvectors:\n",eiv)

#produce a summary for the PCA output.
summary(pca_output)
```

The relevant output:



```
> cat("Eigenvalues:\n", eig)
Eigenvalues:
 9.865542 3.302632 1.205087 1.125598 0.8773732 0.6636175 0.3374343 0.2274918 0.1176166 0.08717899 0.06076839 0.04506468 0.0292071 0.0213994 0.01509618 0.01239134 0.006140071 0.0003622977
> cat("Eigenvectors:\n",eiv)
Eigenvectors:
 -0.2709955 -0.28538 -0.3007838 -0.2759548 -0.1079011 -0.1869378 -0.3092563 0.3071849 -0.3061866 -0.2741952 -0.3024451 -0.3067619 -0.2586001 0.06158617 -0.03877299 -0.05921378 -0.04751059 -0.0972
8514 0.08819711 -0.1479938 0.04064437 0.1928462 0.2459858 0.0683638 -0.07715243 0.01853683 -0.09004872 -0.1358205 -0.0726459 -0.0800464 -0.2182306 -0.5030021 0.02950349 0.09616696 0.5076392 0.503
2953 -0.03979285 -0.1976132 0.07450874 0.04085638 -0.1009268 -0.1060016 0.107481 -0.09109269 0.1060537 -0.2028631 0.1347704 0.1078778 -0.2138646 0.06768991 -0.5533941 0.6822113 -0.07208105 -0.038
70066 -0.1424743 0.02334808 -0.1045135 0.24408 0.6119088 -0.2552416 0.001027495 -0.07139131 -0.02500305 -0.05215126 0.05715318 0.004398469 0.06859533 0.1253773 -0.5176108 -0.4002348 0.02706984 -
0.08990122 -0.1597993 0.1260292 0.07338676 -0.1262041 -0.05646656 0.708019 -0.09117998 0.0854755 -0.08566679 0.2525926 -0.1561663 -0.1250873 0.01184258 -0.1387919 -0.4827463 -0.09248124 -0.174497
0.1205925 0.2197045 -0.01939018 0.000941066 -0.1532342 -0.5994716 -0.2555299 0.07846368 -0.0610722 0.08774873 -0.01258333 0.1034401 0.1063711 -0.06375404 -0.159606 -0.3827182 -0.4717116 0.2409197
0.0829782 0.25075 -0.3818456 0.1092425 0.1381235 0.06368508 0.4090285 0.09891112 -0.1047691 0.09681861 -0.3673347 0.1123422 0.08604684 -0.455865 0.1107949 0.1238176 -0.3163891 -0.1858225 -0.18385
2 -0.7629175 -0.08499684 0.3075604 0.06236231 -0.1466187 -0.03264265 0.09204687 -0.2250398 0.04342616 -0.2413782 0.149166 0.04542186 0.1120117 -0.2986649 0.1283616 -0.1346287 -0.09876744 -0.00225
7517 0.3367273 0.04816196 0.3692975 0.1590392 0.0330752 -0.2277391 -0.1286545 0.2639233 -0.07115043 -0.1211079 -0.1291549 -0.1027789 0.1488793 -0.505836 -0.07022635 0.005249849 -0.4600606 -0.2045
246 -0.1708038 0.1452191 0.09330027 -0.02487175 0.08677308 -0.2510377 0.104393 0.02991565 0.1828748 0.5001775 -0.1697231 0.1144245 -0.6920478 -0.1127 0.07170181 -0.04532918 -0.1826943 0.01863833
0.06059915 -0.06103582 0.7486595 -0.1793243 0.04900671 -0.1084029 -0.1494898 -0.0989528 -0.2683745 0.0945512 0.03485767 -0.2432577 -0.05867687 0.4078034 -0.02036668 -0.03440818 0.1613753 0.122151
3 0.01623622 -0.1080025 0.02723692 -0.1482788 0.06151173 -0.1032849 0.1142392 0.1551623 0.2723695 -0.201415 -0.2287583 0.1778536 0.1537395 0.2206833 0.001322677 -0.08756318 -0.3850709 0.6976855 -
0.155388 -0.02379761 0.2310731 0.02028449 0.03176897 0.09369549 -0.0213004 0.7528627 0.3054049 -0.033803 0.06412845 0.2839925 0.03885792 0.09807688 -0.0151526 -0.01891627 0.3420608 -0.191146 -0.0
849418 0.2003594 -0.03203865 0.7829623 -0.3606866 -0.004005999 -0.0702861 0.1570691 -0.201564 -0.01399651 -0.02651652 -0.0855434 -0.1070407 0.2776439 0.002919287 -0.0221074 -0.07348328 0.1886193
-0.009893937 -0.4116996 -0.1281765 -0.002680653 0.02217136 -0.04769935 -0.1067763 0.2288586 -0.1672837 0.3701051 0.6945293 -0.04637362 0.03930331 -0.07316808 0.03265129 -0.02260938 -0.2241138 0.1
980888 0.01473145 0.6331977 -0.03294137 -0.2621852 0.09120709 0.02392462 0.005784063 0.1328257 -0.2902601 -0.3760757 0.4114796 0.145074 -0.2497047 -0.02126375 0.01764183 0.002337621 -0.08971521
0.1234277 0.002287114 0.1935606 -0.03380826 0.00606873 -0.009140544 -0.004840491 -0.385729 -0.05703083 0.6548967 -0.1025905 0.2336432 -0.5542305 -0.07378847 0.0234043 0.004604286 -0.0007617607 -
0.01183043 0.04328421 -0.0001888306 0.01897982 -0.009571796 -0.0275177 0.01776034 -0.008358115 0.7909902 0.22379 -0.0151608 -0.02665274 0.04168252 -0.5644497 0.003178608 -0.006808075 -0.003066941
-0.007523722 0.03412595 -0.009492286
```



```
> summary(pca_output)
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11   PC12    PC13    PC14    PC15
Standard deviation     3.1409 1.8173 1.09776 1.06094 0.93668 0.81463 0.58089 0.47696 0.34295 0.29526 0.24651 0.2123 0.17090 0.14629 0.12287
Proportion of Variance 0.5481 0.1835 0.06695 0.06253 0.04874 0.03687 0.01875 0.01264 0.00653 0.00484 0.00338 0.0025 0.00162 0.00119 0.00084
Cumulative Proportion  0.5481 0.7316 0.79851 0.86105 0.90979 0.94666 0.96540 0.97804 0.98458 0.98942 0.99280 0.9953 0.99692 0.99811 0.99895
                          PC16    PC17    PC18
Standard deviation     0.11132 0.07836 0.01903
Proportion of Variance 0.00069 0.00034 0.00002
Cumulative Proportion  0.99964 0.99998 1.00000
```

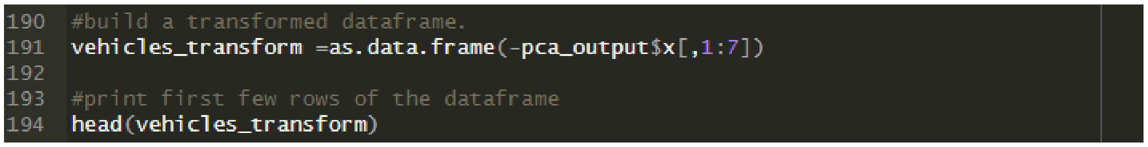
1.2.2 Create a new transformed dataframe

The amount of variance that each principal component contributes to the explanation determines the number of principal components to be used. In order to avoid overfitting, we want to select a sufficient number of components that will effectively capture the bulk of the variance in the initial dataset.

Additionally, by examining the cumulative score for each component, we can select the fewest number of components that yield a total score higher than a predetermined cutoff, in this example, 92%. The first seven key components have a total score based on the cumulative scores shown above of 0.96540, which is higher than the task's basic criterion of 0.92. According to this, the top seven major factors can explain "97% of the dataset." As a result, we can create a new dataset for k-means using the first seven primary components. The Cumulative Proportion significantly rises when comparing the first 11 PCs to the last 11 PCs. Because of this, even if the first 6 PCs offer a Cumulative Proportion of 0.84666, which is greater than 0.92000, the first 7 PCs are chosen rather than the first 6 PCs.

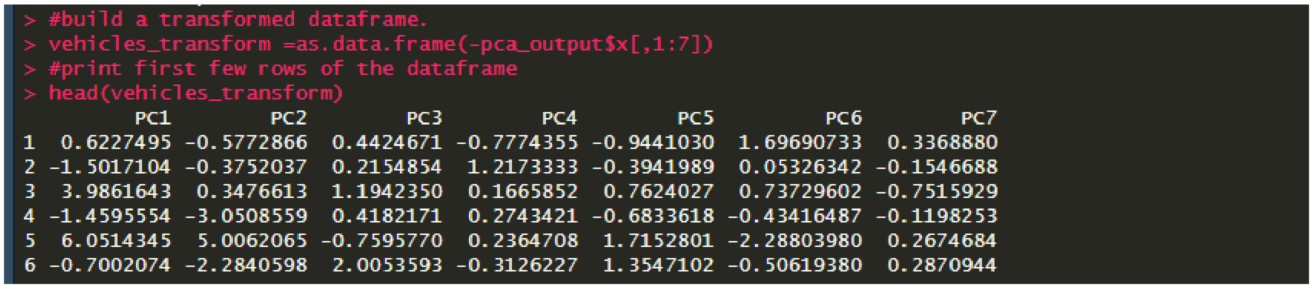
Once we've established the right number of components, we may convert the original dataset into a new dataset with attributes representing the principal components. It will be simpler to visualize and evaluate this new dataset due to its lower dimensionality.

The relevant code:



```
#build a transformed dataframe.
vehicles_transform =as.data.frame(-pca_output$x[,1:7])

#print first few rows of the dataframe
head(vehicles_transform)
```

The relevant Output:



```
> #build a transformed dataframe.
> vehicles_transform =as.data.frame(-pca_output$x[,1:7])
> #print first few rows of the dataframe
> head(vehicles_transform)
         PC1        PC2        PC3        PC4        PC5         PC6        PC7
1  0.6227495 -0.5772866  0.4424671 -0.7774355 -0.9441030  1.69690733  0.3368880
2 -1.5017104 -0.3752037  0.2154854  1.2173333 -0.3941989  0.05326342 -0.1546688
3  3.9861643  0.3476613  1.1942350  0.1665852  0.7624027  0.73729602 -0.7515929
4 -1.4595554 -3.0508559  0.4182171  0.2743421 -0.6833618 -0.43416487 -0.1198253
5  6.0514345  5.0062065 -0.7595770  0.2364708  1.7152801 -2.28803980  0.2674684
6 -0.7002074 -2.2840598  2.0053593 -0.3126227  1.3547102 -0.50619380  0.2870944
```

1.2.3Determine the Number of cluster centers for a newly created Dataset

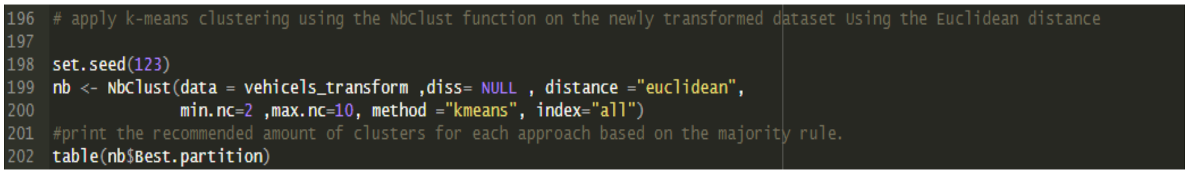
For our newly constructed dataset, there are several automated methods that may be used to calculate the optimal number of cluster centers for k-means clustering:

- **NBclust**

  These indices are implemented by the NbClust function, which generates the number of clusters to use based on the majority rule. It's a popular R tool for automated clustering analysis.

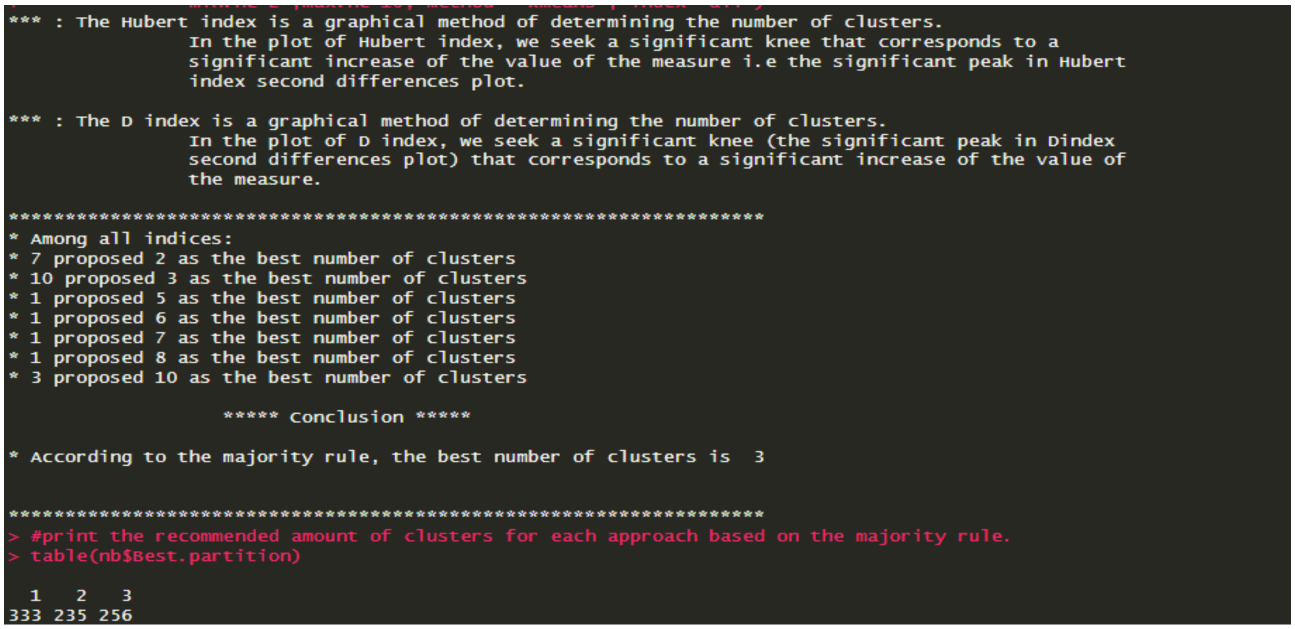
  Using Euclidean Distance

  The relevant code (Using Euclidean Distance):

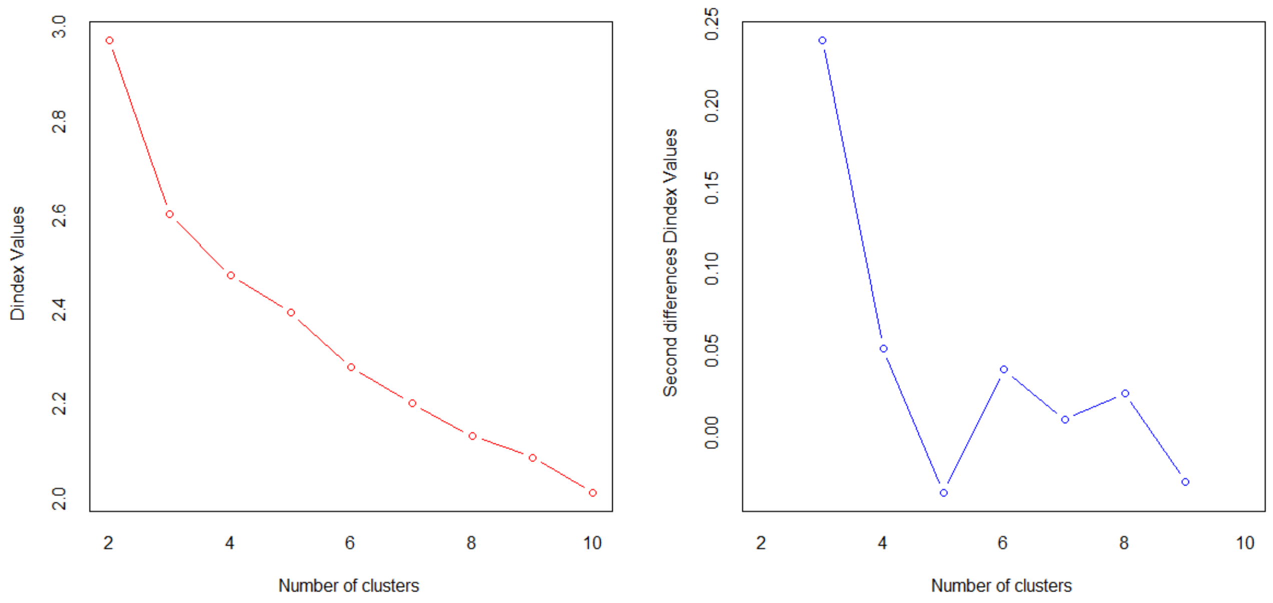


```
# apply k-means clustering using the NbClust function on the newly transformed dataset Using the Euclidean distance

set.seed(123)
nb <- NbClust(data = vehicels_transform ,diss= NULL , distance ="euclidean",
              min.nc=2 ,max.nc=10, method ="kmeans", index="all")
#print the recommended amount of clusters for each approach based on the majority rule.
table(nb$Best.partition)
```

  The relevant output using (Using Euclidean Distance):



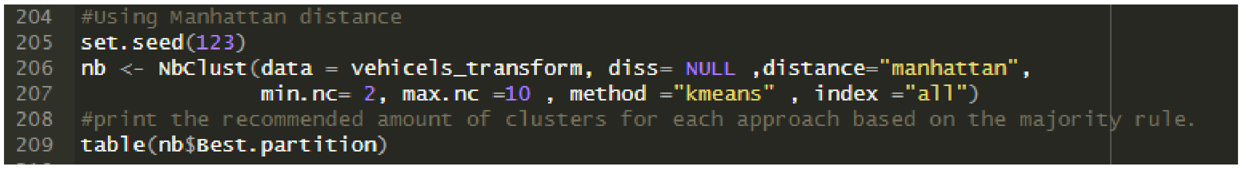
```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index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

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

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

                   ***** Conclusion *****

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


*******************************************************************
> #print the recommended amount of clusters for each approach based on the majority rule.
> table(nb$Best.partition)

  1   2   3
333 235 256
```



The Nbclust method (Using the Euclidean Distance) shows that the optimum number of clusters for a newly created dataset is three, based on the results mentioned above.

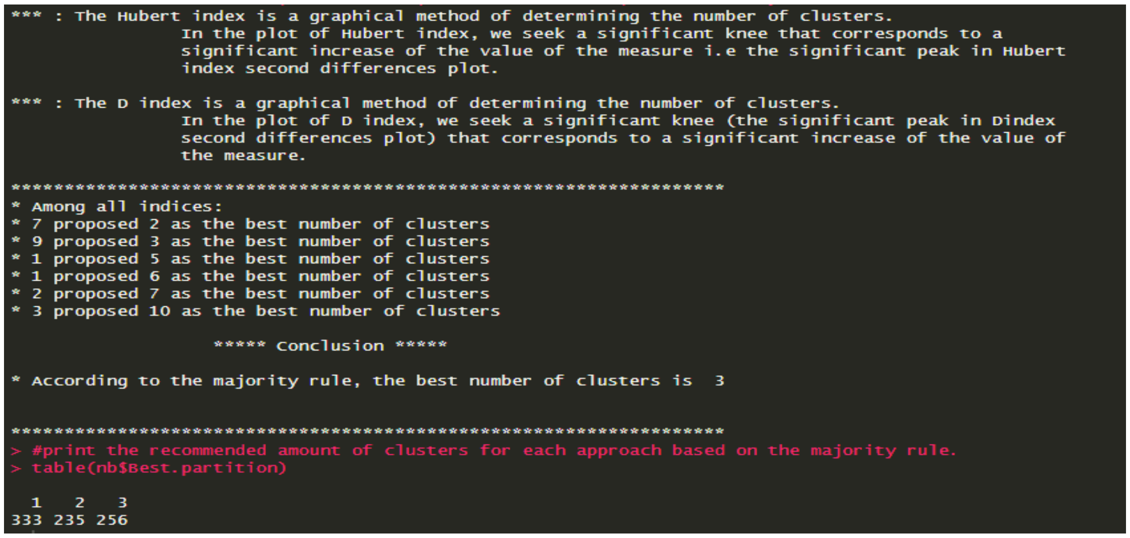
Using Manhattan Distance

The relevant code (Using Manhattan Distance):

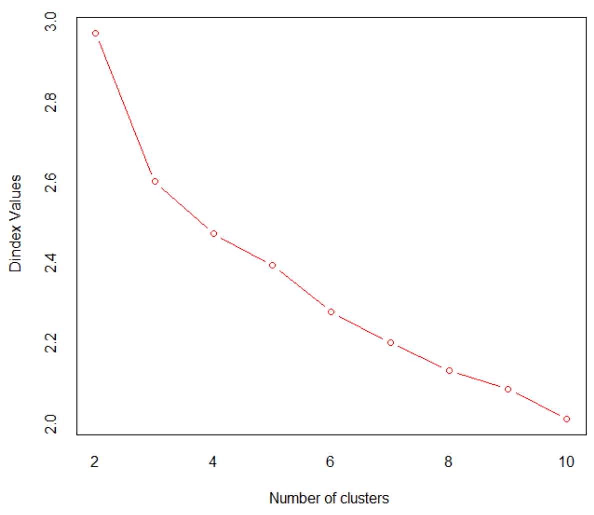


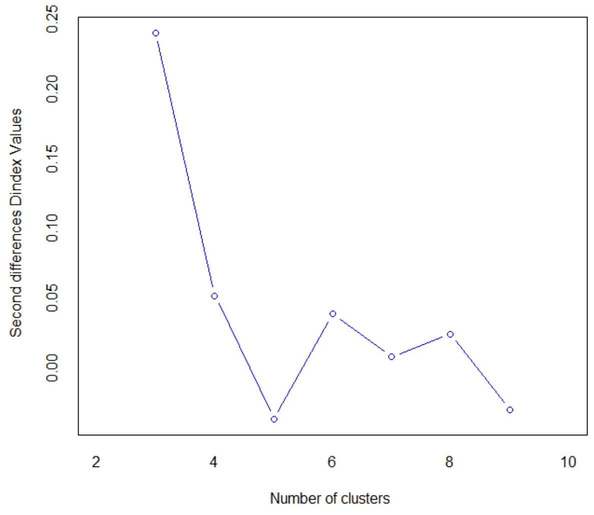
```r
#Using Manhattan distance
set.seed(123)
nb <- NbClust(data = vehicels_transform, diss= NULL ,distance="manhattan",
              min.nc= 2, max.nc =10 , method ="kmeans" , index ="all")
#print the recommended amount of clusters for each approach based on the majority rule.
table(nb$Best.partition)
```

The relevant output using (Using Manhattan Distance):



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

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

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

                   ***** Conclusion *****

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


*******************************************************************
> #print the recommended amount of clusters for each approach based on the majority rule.
> table(nb$Best.partition)

  1   2   3
333 235 256
```



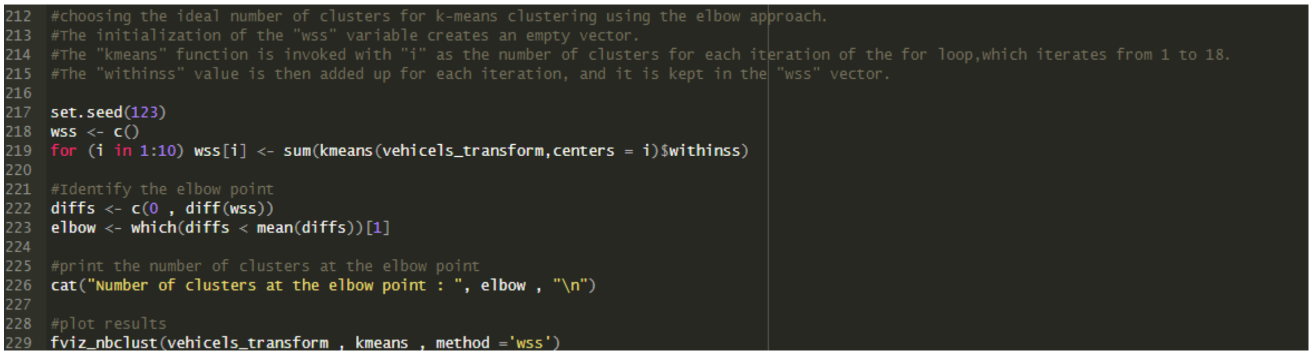


The Nbclust method (Using the Manhattan Distance) shows that the optimum number of clusters for a newly created dataset is three, based on the results mentioned above.

- **Elbow**

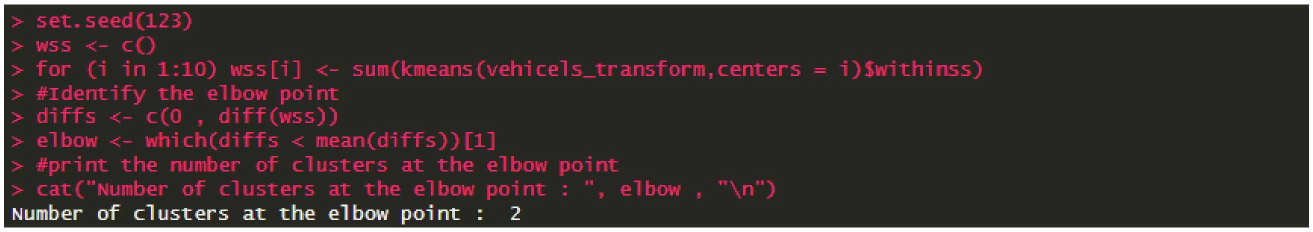
For k-means clustering, the elbow approach is used to calculate the ideal number of clusters. The concept is to plot the total within-cluster sum of squares (WSS) vs. the number of clusters (K).The sum of the squared Euclidean distances between each data point and its assigned centroid is the total WSS.

The relevant code:

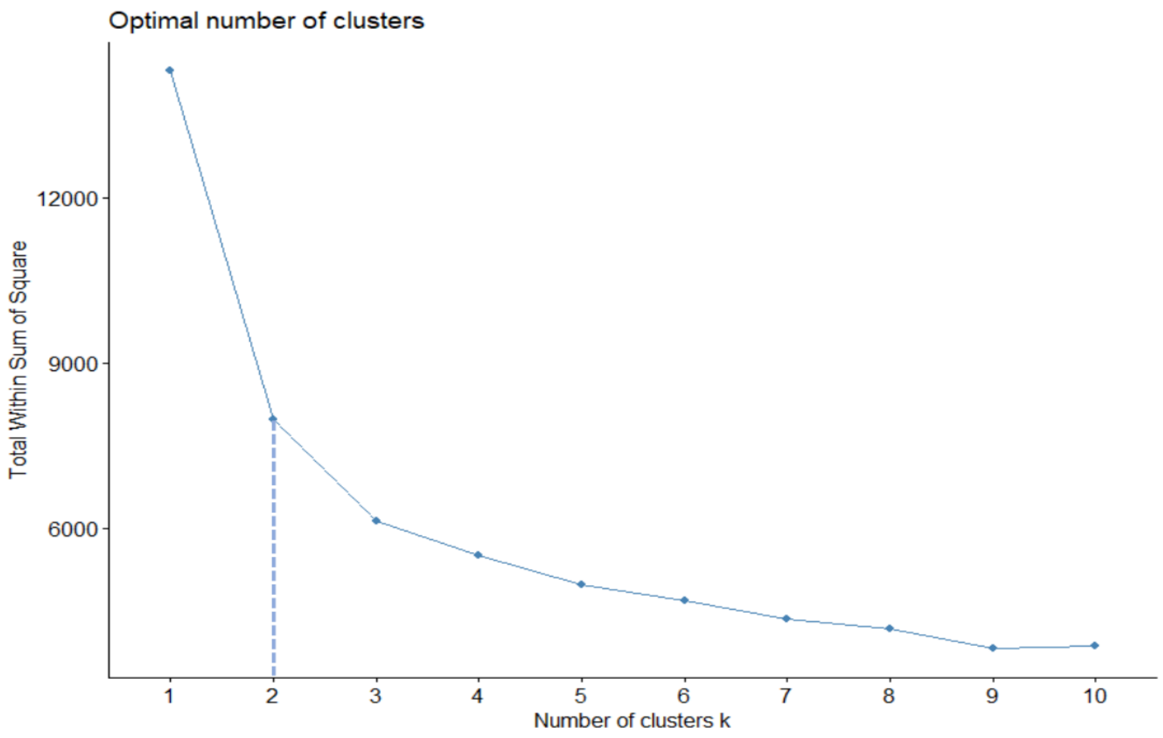


```
#choosing the ideal number of clusters for k-means clustering using the elbow approach.
#The initialization of the "wss" variable creates an empty vector.
#The "kmeans" function is invoked with "i" as the number of clusters for each iteration of the for loop,which iterates from 1 to 18.
#The "withinss" value is then added up for each iteration, and it is kept in the "wss" vector.

set.seed(123)
wss <- c()
for (i in 1:10) wss[i] <- sum(kmeans(vehicels_transform,centers = i)$withinss)

#Identify the elbow point
diffs <- c(0 , diff(wss))
elbow <- which(diffs < mean(diffs))[1]

#print the number of clusters at the elbow point
cat("Number of clusters at the elbow point : ", elbow , "\n")

#plot results
fviz_nbclust(vehicels_transform , kmeans , method ='wss')
```

The relevant output**:**



```
> set.seed(123)
> wss <- c()
> for (i in 1:10) wss[i] <- sum(kmeans(vehicels_transform,centers = i)$withinss)
> #Identify the elbow point
> diffs <- c(0 , diff(wss))
> elbow <- which(diffs < mean(diffs))[1]
> #print the number of clusters at the elbow point
> cat("Number of clusters at the elbow point : ", elbow , "\n")
Number of clusters at the elbow point :  2
```


Optimal number of clusters
Total Within Sum of Square
12000
9000
6000
1
2
3
4
5
6
7
8
9
10
Number of clusters k

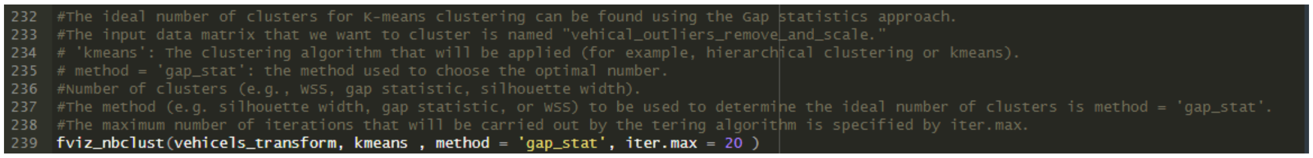
The elbow point on the plot must be determined to be the point at which the rate of decrease in WSS (Within the Cluster Sum of Squares) begins to level off. The number of clusters corresponding to the elbow point is considered to be the ideal number.

According to the above results, the Elbow technique shows that the optimal number of clusters for a newly constructed dataset is 2.

• Gap statistics

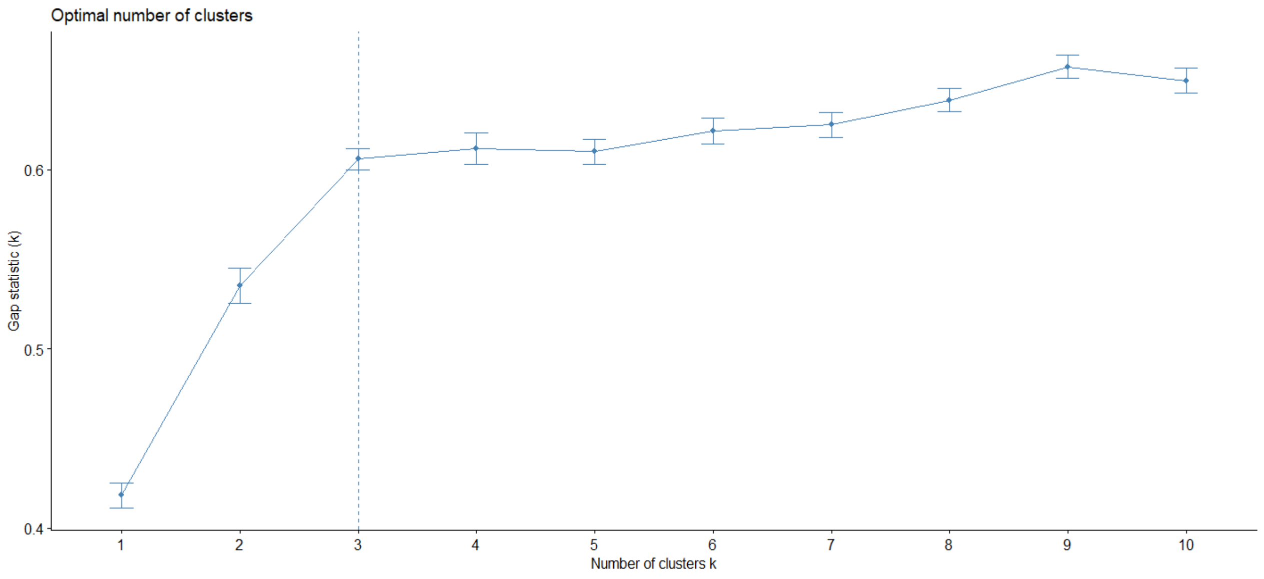
To determine the number of clusters in a dataset, the gap statistics approach can be used. It compares the overall within-cluster variation across various k (number of clusters) values to the variation expected for a random data set with no clustering.

The relevant code:



```
#The ideal number of clusters for K-means clustering can be found using the Gap statistics approach.
#The input data matrix that we want to cluster is named "vehical_outliers_remove_and_scale."
# 'kmeans': The clustering algorithm that will be applied (for example, hierarchical clustering or kmeans).
# method = 'gap_stat': the method used to choose the optimal number.
#Number of clusters (e.g., WSS, gap statistic, silhouette width).
#The method (e.g. silhouette width, gap statistic, or WSS) to be used to determine the ideal number of clusters is method = 'gap_stat'.
#The maximum number of iterations that will be carried out by the tering algorithm is specified by iter.max.
fviz_nbclust(vehicels_transform, kmeans , method = 'gap_stat', iter.max = 20 )
```

The relevant output:

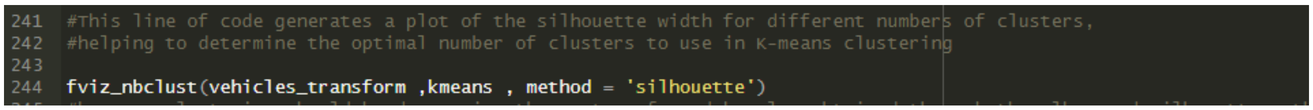


The Gap statistical method's findings indicate that 3 clusters are an ideal amount for a newly created dataset.

• Silhouette

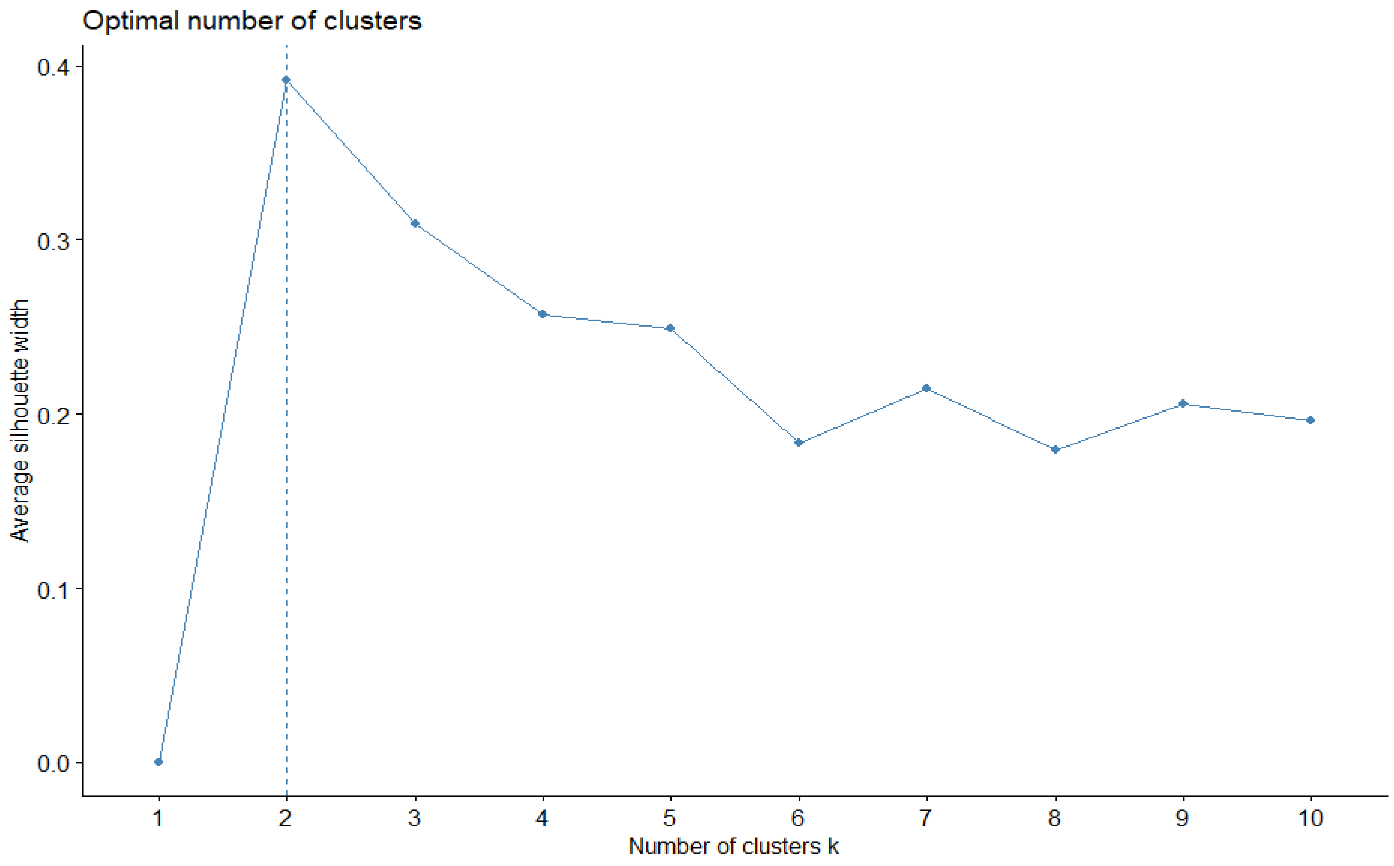
The technique of silhouette analysis is used to evaluate the quality of a clustering solution. It compares how similar each data point is to its own cluster (cohesion) to other clusters (separation).

The relevant code:



```
#This line of code generates a plot of the silhouette width for different numbers of clusters,
#helping to determine the optimal number of clusters to use in K-means clustering

fviz_nbclust(vehicles_transform ,kmeans , method = 'silhouette')
```

The relevant output:



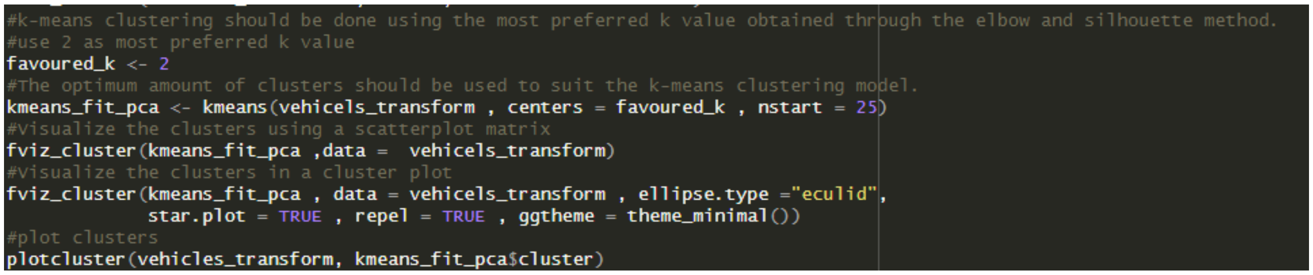
The Silhouette technique suggests that two clusters are the optimal number of clusters based on the above-mentioned results. The resulting plot shows that three clusters are the second optimal number of clusters, while two clusters maximum the average silhouette width.

1.2.4 The kmeans Clustering Investigation & Evaluation for Newly Created Dataset

Because the Elbow and Silhouette methods show that the ideal number of clusters is 2, it is decided to choose 2 as the optimal K value for the newly created dataset.

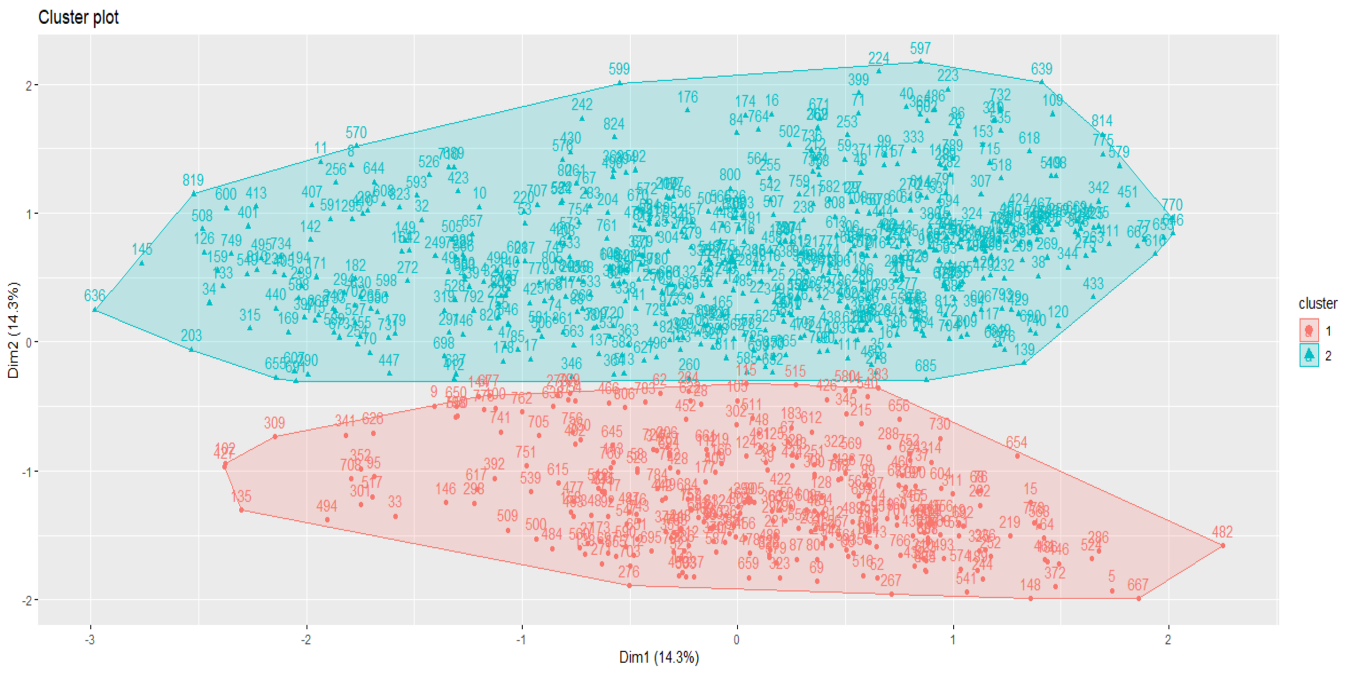
Apply the optimal k value determined from the automated methods to perform k-means clustering.

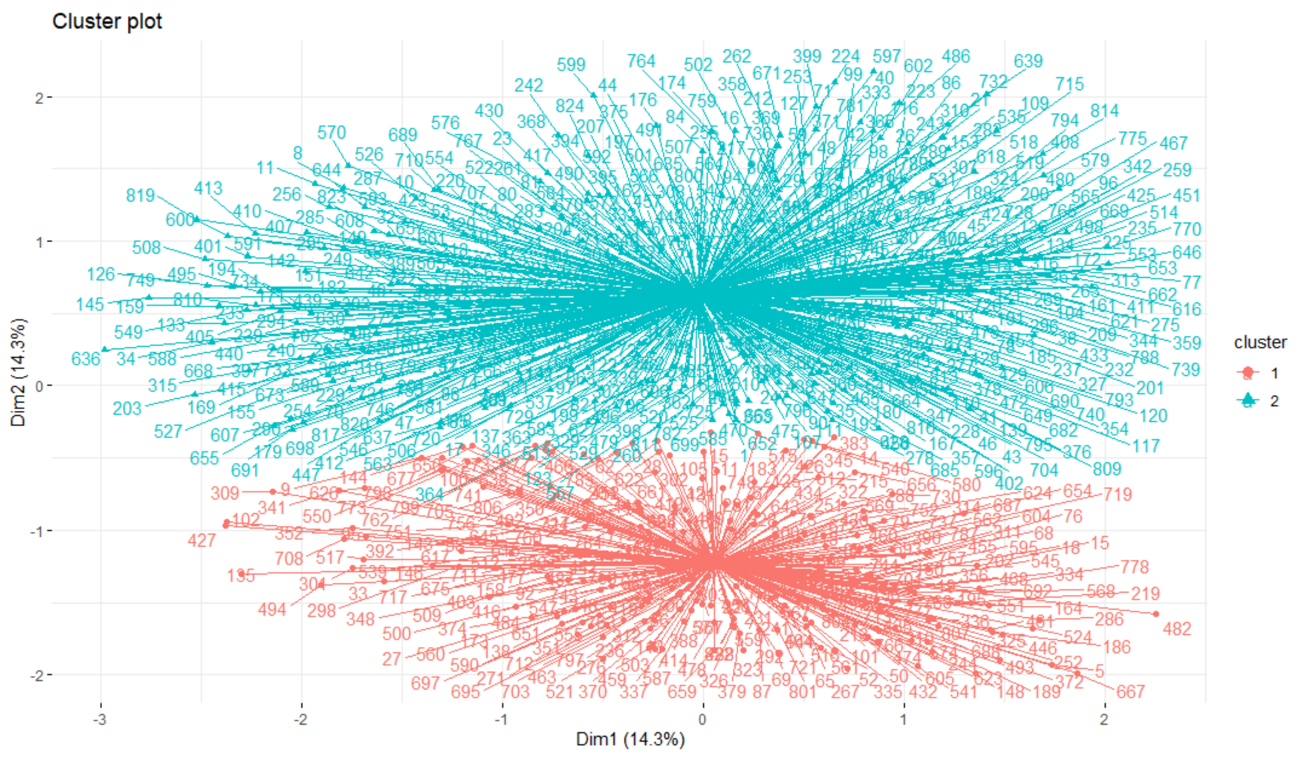
The relevant code:

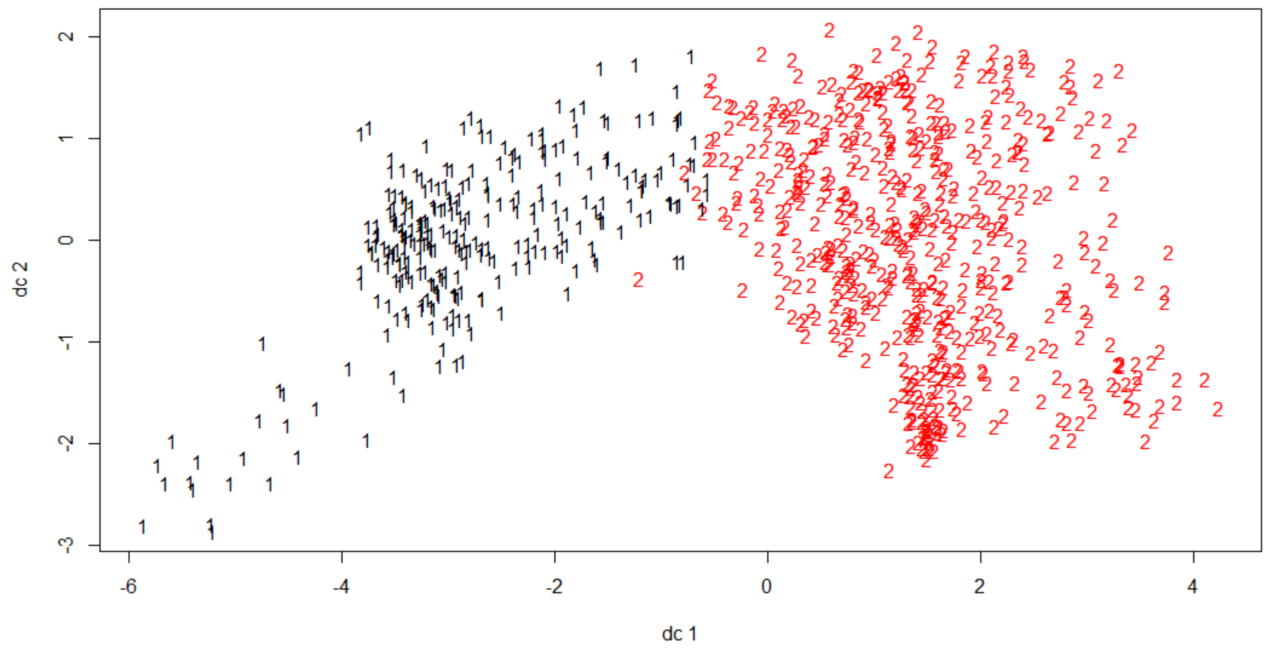


```
#k-means clustering should be done using the most preferred k value obtained through the elbow and silhouette method.
#use 2 as most preferred k value
favoured_k <- 2
#The optimum amount of clusters should be used to suit the k-means clustering model.
kmeans_fit_pca <- kmeans(vehicels_transform , centers = favoured_k , nstart = 25)
#Visualize the clusters using a scatterplot matrix
fviz_cluster(kmeans_fit_pca ,data =  vehicels_transform)
#Visualize the clusters in a cluster plot
fviz_cluster(kmeans_fit_pca , data = vehicels_transform , ellipse.type ="eculid",
             star.plot = TRUE , repel = TRUE , ggtheme = theme_minimal())
#plot clusters
plotcluster(vehicles_transform, kmeans_fit_pca$cluster)
```

The relevant output:

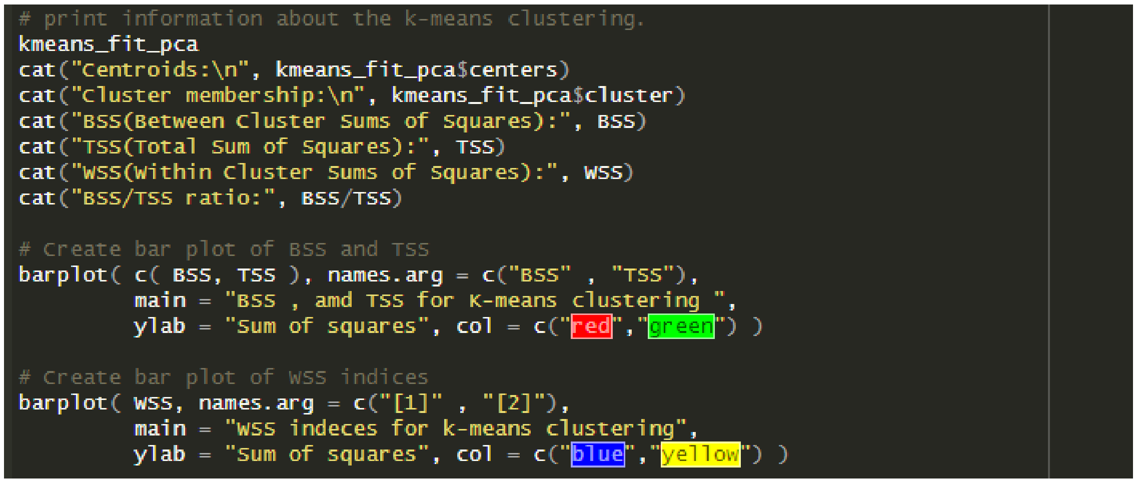



Cluster plot
Dim1 (14.3%)
Dim2 (14.3%)
cluster
1
2
-3
-2
-1
0
1
2


dc 1
dc 2
-6
-4
-2
0
2
4
-3
-2
-1
0
1
2

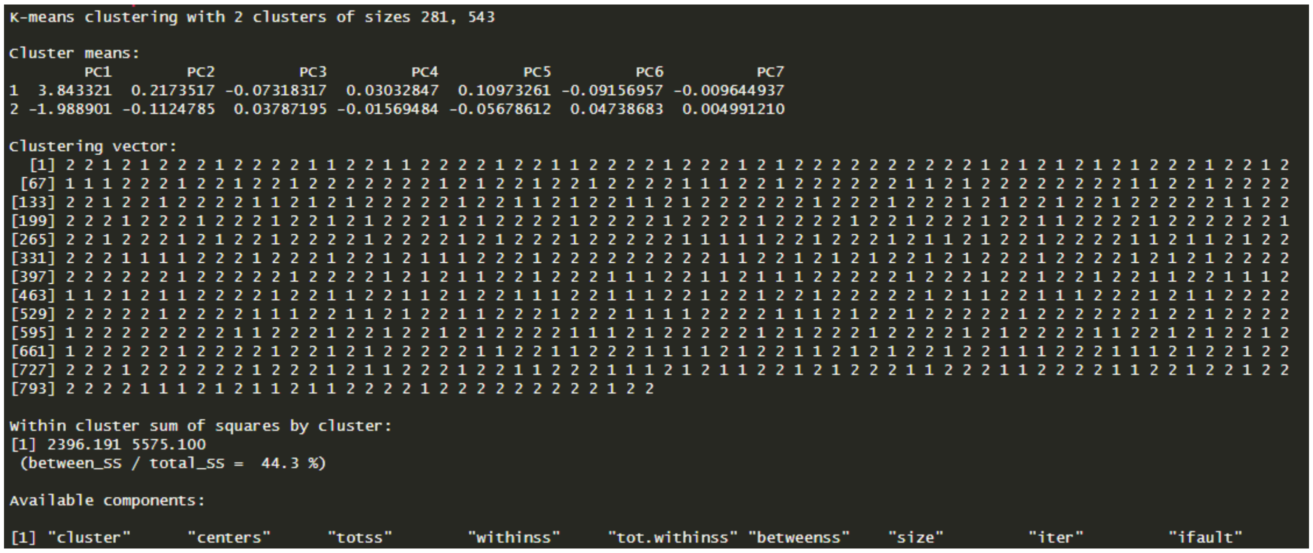
The centroids, the cluster vector (cluster membership), the ratio of the between-cluster sums of squares (BSS) to the total sum of squares (TSS) - (BSS/TSS), and the computation and illustration of the between-cluster sums of squares (BSS) and within-cluster sums of squares (WSS) indices (internal evaluation metrics) are all described below.

The relevant code:

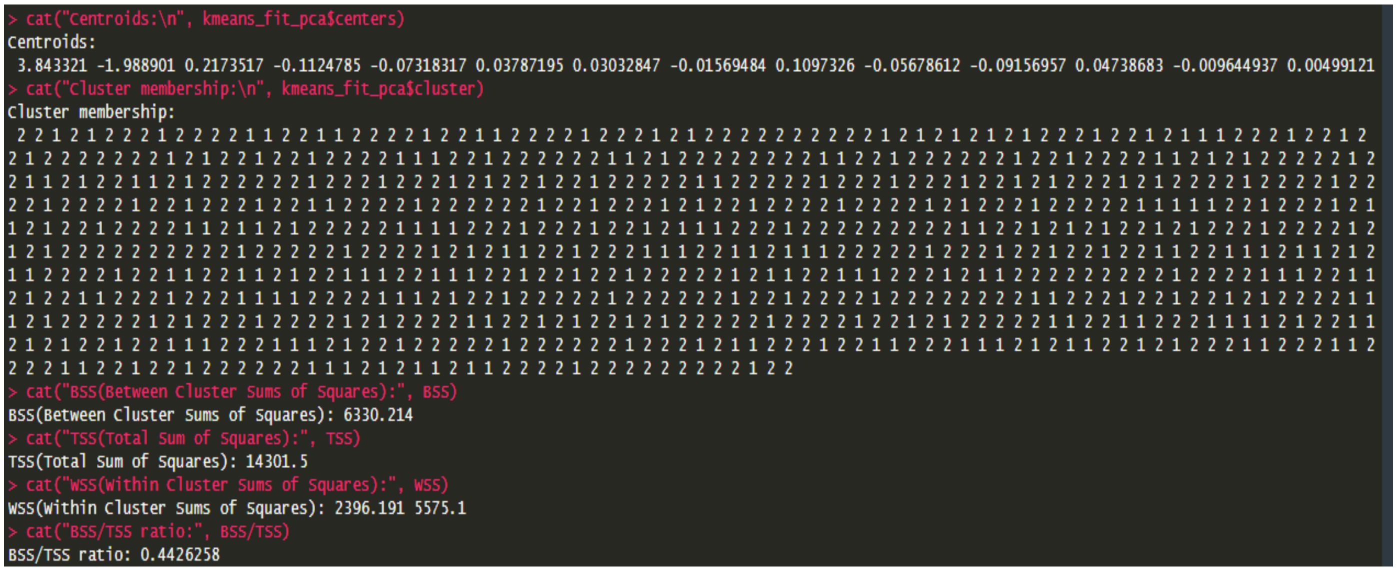


```
# print information about the k-means clustering.
kmeans_fit_pca
cat("Centroids:\n", kmeans_fit_pca$centers)
cat("Cluster membership:\n", kmeans_fit_pca$cluster)
cat("BSS(Between Cluster Sums of Squares):", BSS)
cat("TSS(Total Sum of Squares):", TSS)
cat("WSS(Within Cluster Sums of Squares):", WSS)
cat("BSS/TSS ratio:", BSS/TSS)

# Create bar plot of BSS and TSS
barplot( c( BSS, TSS ), names.arg = c("BSS" , "TSS"),
         main = "BSS , amd TSS for K-means clustering ",
         ylab = "Sum of squares", col = c("red","green") )

# Create bar plot of WSS indices
barplot( WSS, names.arg = c("[1]" , "[2]"),
         main = "WSS indeces for k-means clustering",
         ylab = "Sum of squares", col = c("blue","yellow") )
```

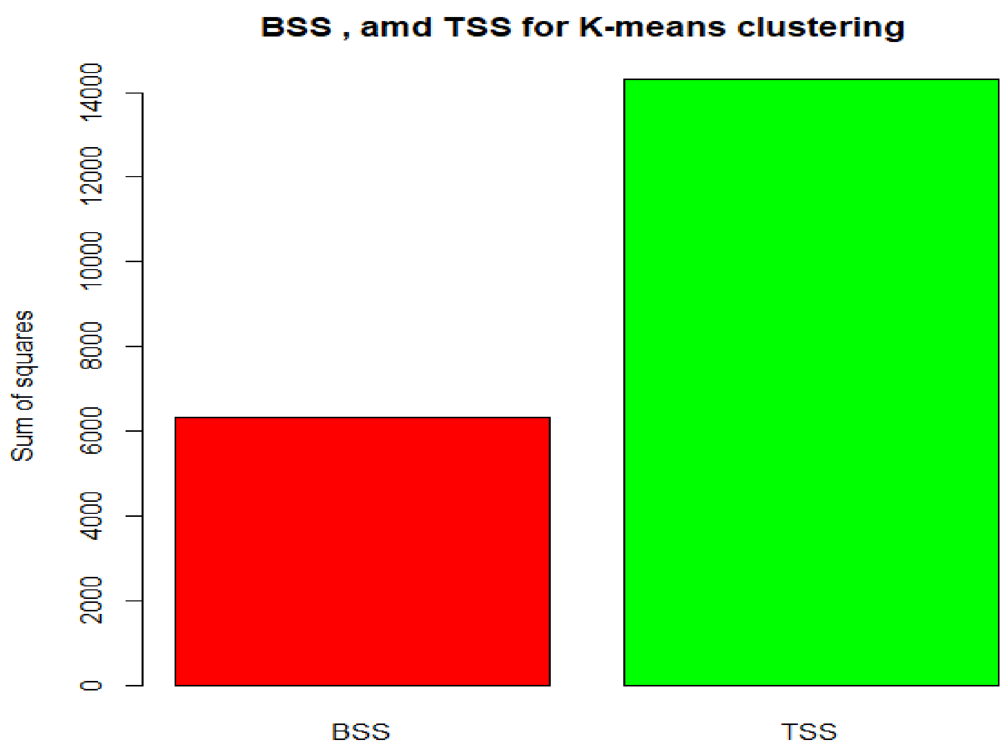
The relevant output:

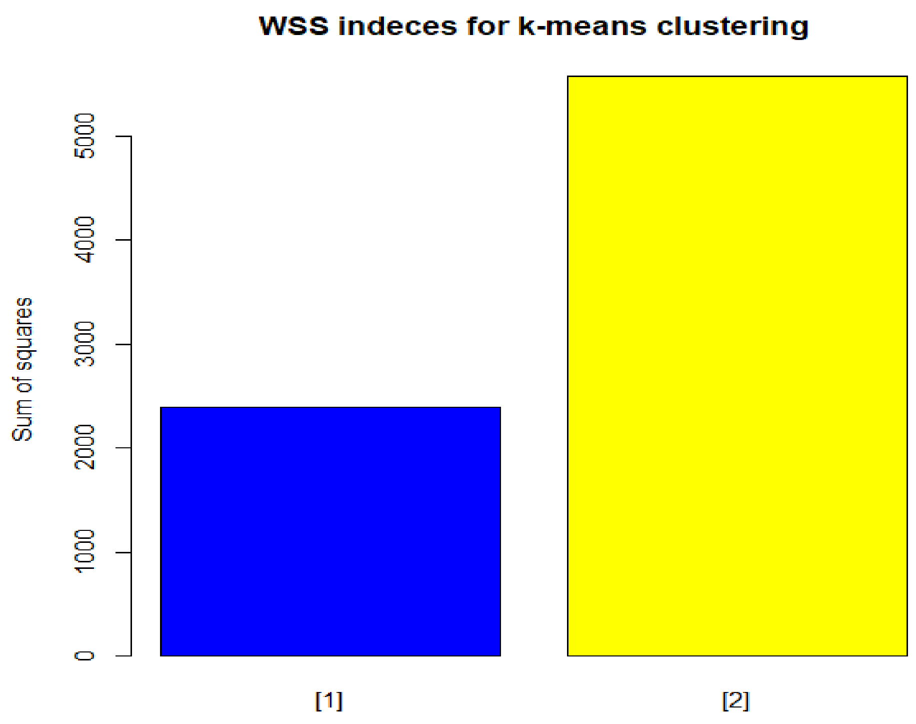


```
K-means clustering with 2 clusters of sizes 281, 543

Cluster means:
        PC1        PC2         PC3         PC4         PC5         PC6          PC7
1  3.843321  0.2173517 -0.07318317  0.03032847  0.10973261 -0.09156957 -0.009644937
2 -1.988901 -0.1124785  0.03787195 -0.01569484 -0.05678612  0.04738683  0.004991210

Clustering vector:
  [1] 2 2 1 2 1 2 2 2 1 2 2 2 2 1 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 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 2 1 2
 [67] 1 1 1 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 1 1 1 2 2 1 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2 2 2
[133] 2 2 1 2 2 1 2 2 2 2 1 1 2 1 2 1 2 2 2 2 2 1 2 2 1 1 2 1 2 2 1 1 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 2 1 1 2 2
[199] 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1 2 1 2 2 2 1 2 1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 2 2 1 2 2 1 2 2 2 1 2 2 1 1 2 2 2 2 1 2 2 2 2 2 2 1
[265] 2 2 1 2 2 2 1 2 1 2 2 1 2 2 2 2 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 1 1 1 1 1 2 2 1 2 2 2 1 2 1 1 2 1 2 2 1 2 2 2 2 1 1 2 1 1 2 1 2 2
[331] 2 2 2 1 1 1 1 2 2 2 1 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1 2 1 2 2 2 2
[397] 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 1 2 1 1 2 2 1 2 2 2 1 1 1 2 2 1 1 2 1 1 1 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 1 2 2 1 1 2 2 1 1 1 2
[463] 1 1 2 1 2 1 1 2 2 2 2 1 2 2 1 1 2 2 1 1 2 1 2 2 1 1 1 2 2 1 1 1 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 2 1 1 2 2 1 1 1 2 2 2 1 2 1 1 2 2 2 2
[529] 2 2 2 2 2 1 2 2 2 2 1 1 1 2 2 1 1 2 1 2 2 1 1 2 2 2 1 2 2 2 1 1 1 1 2 2 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 2
[595] 1 2 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 2 2 1 2 1 2 2 2 2 1 1 1 2 1 2 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1 2 2 2 2 1 1 2 2 1 2 1 2 2 1 2
[661] 1 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 1 1 2 2 1 1 2 2 2 1 1 1 1 2 1 2 2 1 1 2 1 2 1 2 2 1 2 2 1 1 1 2 2 2 1 1 1 2 1 2 2 1 2 2
[727] 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 1 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 1 2 2 2 1 1 2 2 2 2 1 1 2 2 1 2 2 1 2 2
[793] 2 2 2 2 1 1 1 2 1 2 1 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2

Within cluster sum of squares by cluster:
[1] 2396.191 5575.100
 (between_SS / total_SS =  44.3 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"         "ifault"
```



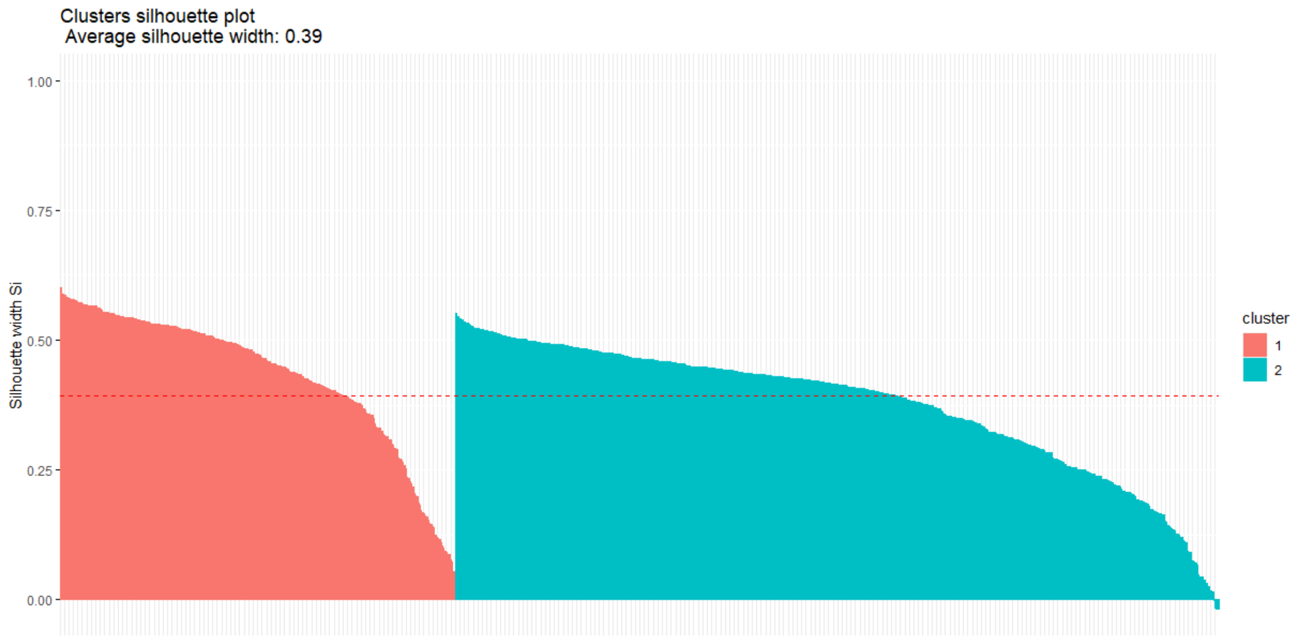
```
> cat("Centroids:\n", kmeans_fit_pca$centers)
Centroids:
 3.843321 -1.988901 0.2173517 -0.1124785 -0.07318317 0.03787195 0.03032847 -0.01569484 0.1097326 -0.05678612 -0.09156957 0.04738683 -0.009644937 0.00499121
> cat("Cluster membership:\n", kmeans_fit_pca$cluster)
Cluster membership:
 2 2 1 2 1 2 2 2 1 2 2 2 2 1 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 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 1 2 1 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 2 2 1 2
2 1 2 2 2 2 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 1 1 1 2 2 1 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 1 2 1 2 1 2 2 2 2 2 1 2
2 1 1 2 1 2 2 1 1 2 1 2 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 1 2 2 1 2 2 1 2 2 2 2 2 1 1 2 2 2 2 2 1 2 2 2 1 2 2 2 1 2 2 1 2 1 2 2 2 1 2 1 2 2 2 2 1 2 2 2 2 1 2 2
2 2 1 2 2 2 2 1 2 2 1 2 2 2 1 2 2 1 1 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 1 2 1 2 2 1 2 2 2 2 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 2 1 1 1 1 1 2 2 1 2 2 2 1 2 1
1 2 1 2 2 1 2 2 2 2 1 1 2 1 1 2 1 2 2 2 2 2 1 1 1 1 2 2 2 1 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 2 2 2 2 2 2 2 2 2 1 1 2 2 1 2 1 2 1 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2
1 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 2 1 2 1 2 1 1 2 2 1 2 2 2 1 1 1 2 2 1 1 2 1 1 1 2 2 2 2 2 1 2 2 2 1 2 2 1 2 2 1 2 2 1 1 2 2 1 1 1 2 1 1 2 1 2
1 1 2 2 2 2 1 2 2 1 1 2 2 1 1 2 1 2 2 1 1 1 2 2 1 1 1 2 2 1 2 2 1 2 2 1 2 2 2 2 2 1 2 1 1 2 2 1 1 1 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 1 1 2 2 1 1
2 1 2 2 1 1 2 2 2 1 2 2 2 1 1 1 1 2 2 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 2 1 1 2 2 2 1 2 2 1 2 2 1 2 1 2 2 2 2 1 1
1 2 1 2 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1 2 2 2 2 1 1 2 2 1 2 1 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 1 2 2 1 2 1 2 2 2 2 2 1 1 2 2 1 1 2 2 2 1 1 1 1 2 1 2 2 1 1
2 1 2 1 2 2 1 2 2 1 1 1 2 2 2 1 1 1 2 1 2 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 1 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 1 2 2 2 1 1 2
2 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2 1 1 1 2 1 2 1 1 2 1 1 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2
> cat("BSS(Between Cluster Sums of Squares):", BSS)
BSS(Between Cluster Sums of Squares): 6330.214
> cat("TSS(Total Sum of Squares):", TSS)
TSS(Total Sum of Squares): 14301.5
> cat("WSS(Within Cluster Sums of Squares):", WSS)
WSS(Within Cluster Sums of Squares): 2396.191 5575.1
> cat("BSS/TSS ratio:", BSS/TSS)
BSS/TSS ratio: 0.4426258
```





A visualization tool that helps when evaluating the quality of clustering results is the silhouette plot. Based on the distance between a data point and other data points in its own cluster in comparison to the distance to the closest neighboring cluster, it shows a measure of how well each data point fits into its allocated cluster. A silhouette plot is used to provide a visual representation of the silhouette coefficient of each data point in a clustering analysis as another internal evaluation metric. The silhouette coefficient evaluates how similar a data point is to its own cluster in relation to other clusters. A vertical line is used to represent each data point in a silhouette plot, and the height of the line indicates the silhouette coefficient for that particular point. The plot could help in evaluating the clustering's quality visually and identifying any cluster assignment problems.

In an effective clustering solution, high silhouette coefficients demonstrate that the data points are well-separated into several clusters. On the other hand, a poor clustering solution will have low Silhouette coefficients, indicating that the data points are Low Silhouette coefficients, on the other hand, signify a bad clustering solution, which means the data points are ill-defined and could be a part of multiple clusters.



The silhouette width values range from -1 to 1, with a value closer to 1 indicating better grouping. An average Silhouette width score of 0.39 is considered reasonably good, indicating that the clusters are somewhat well-separated but still have a space for development.

Viewers may quickly recognize two distinct clusters as cluster 1 and cluster 2, suggesting that the data points in the output plot shown above are clearly split into discrete clusters. Each cluster's high Silhouette coefficients show that the data points within each cluster are more comparable to those in the other cluster.

But because it's not a very high value, it suggests that there might be some overlap between the clusters and that some data points might be grouped into the wrong cluster. The context and the particular situation under discussion often influence how to interpret the silhouette width value.

It's also important to note that silhouette width is only one of several metrics used to evaluate the effectiveness of a clustering approach. It is typically advised to use a number of metrics and visualizations to acquire a broad view of the clustering results. To assess the validity of the clustering solution and make sure that it is in line with the fundamental characteristics of the data, the silhouette plot should be used in conjunction with other evaluation metrics and domain expertise.

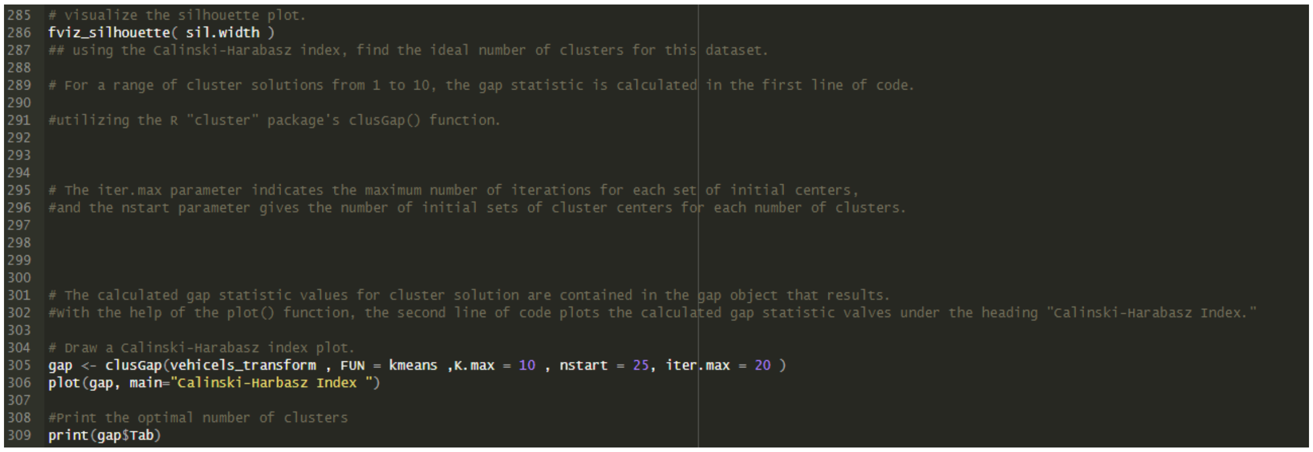
1.2.5 Implement and illustrate the Calinski-Harabasz Index

As a cluster validity criterion, the Calinski-Harabasz Index, sometimes referred to as the Variance Ratio Criterion, applies to evaluate clustering results. A greater ratio indicates better cluster separation. It is determined by dividing between-cluster variance by within-cluster variance.

The ratio of the between-cluster sum of squares to the within-cluster sum of squares, multiplied by a scaling factor dependent on the number of clusters and data points in the data set, is used to construct the Calinski-Harabasz Index.

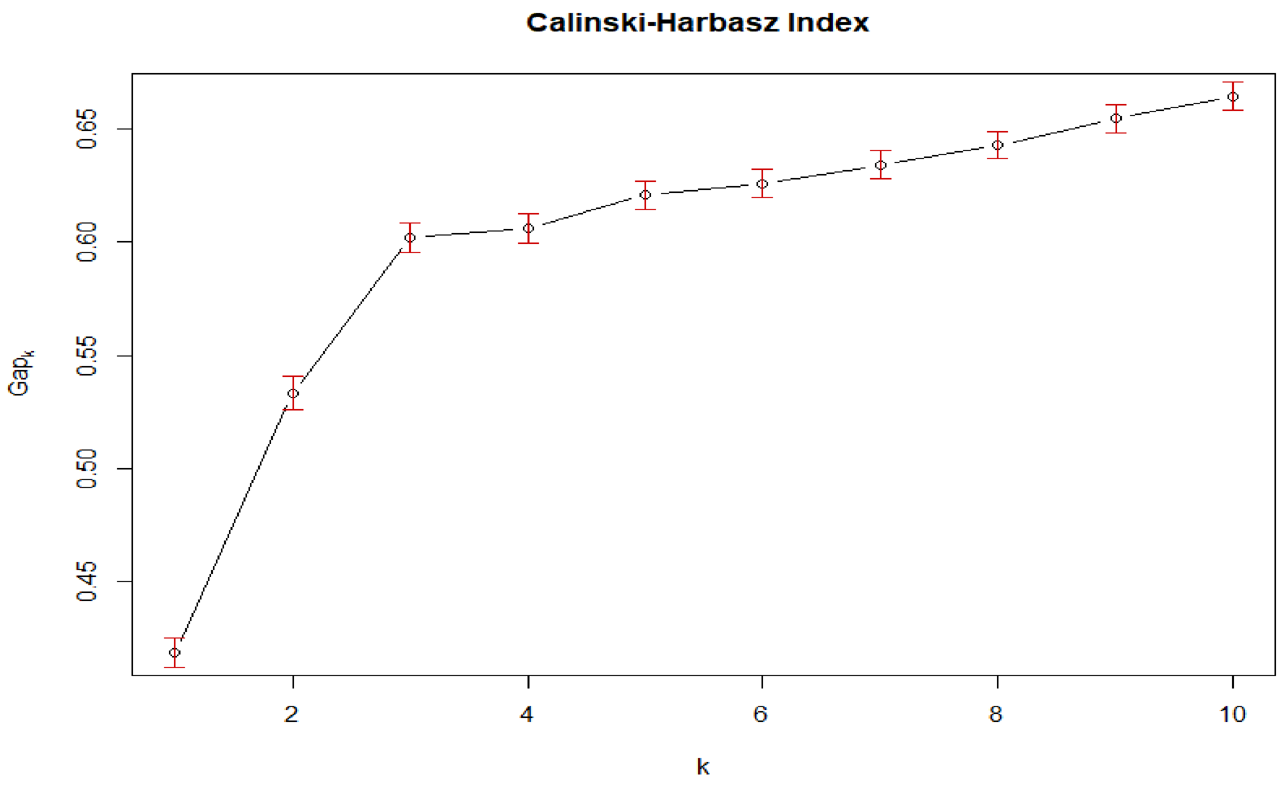
A helpful tool to evaluate the quality of clustering outcomes and determine the optimal number of clusters for a particular data set provides by the Calinski-Harabasz Index.

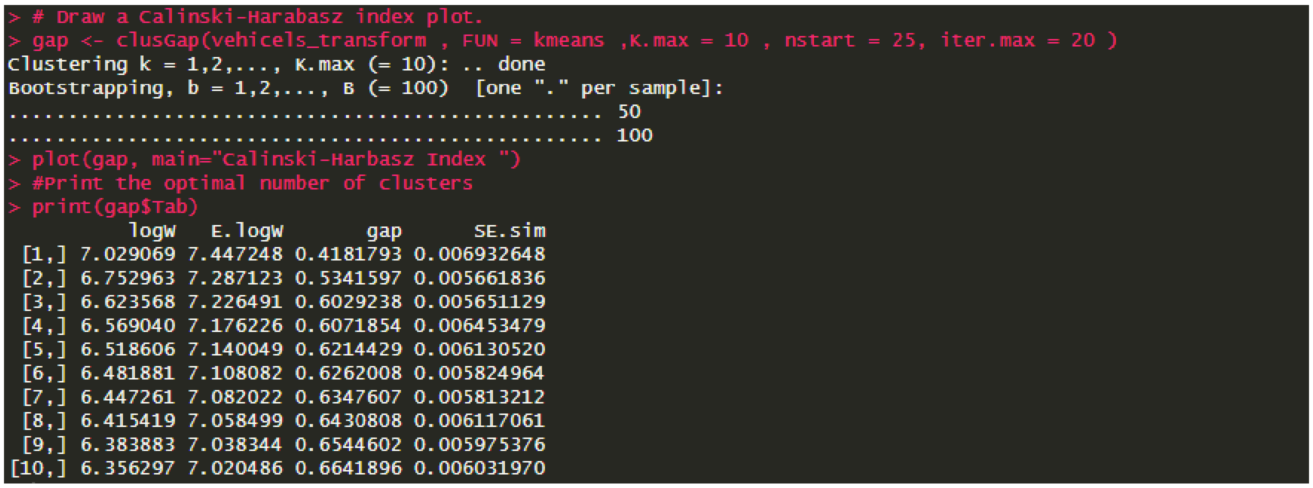
The relevant code:



```r
# visualize the silhouette plot.
fviz_silhouette( sil.width )
## using the Calinski-Harabasz index, find the ideal number of clusters for this dataset.

# For a range of cluster solutions from 1 to 10, the gap statistic is calculated in the first line of code.

#utilizing the R "cluster" package's clusGap() function.



# The iter.max parameter indicates the maximum number of iterations for each set of initial centers,
#and the nstart parameter gives the number of initial sets of cluster centers for each number of clusters.




# The calculated gap statistic values for cluster solution are contained in the gap object that results.
#With the help of the plot() function, the second line of code plots the calculated gap statistic valves under the heading "Calinski-Harabasz Index."

# Draw a Calinski-Harabasz index plot.
gap <- clusGap(vehicels_transform , FUN = kmeans ,K.max = 10 , nstart = 25, iter.max = 20 )
plot(gap, main="Calinski-Harbasz Index ")

#Print the optimal number of clusters
print(gap$Tab)
```

The relevant output:

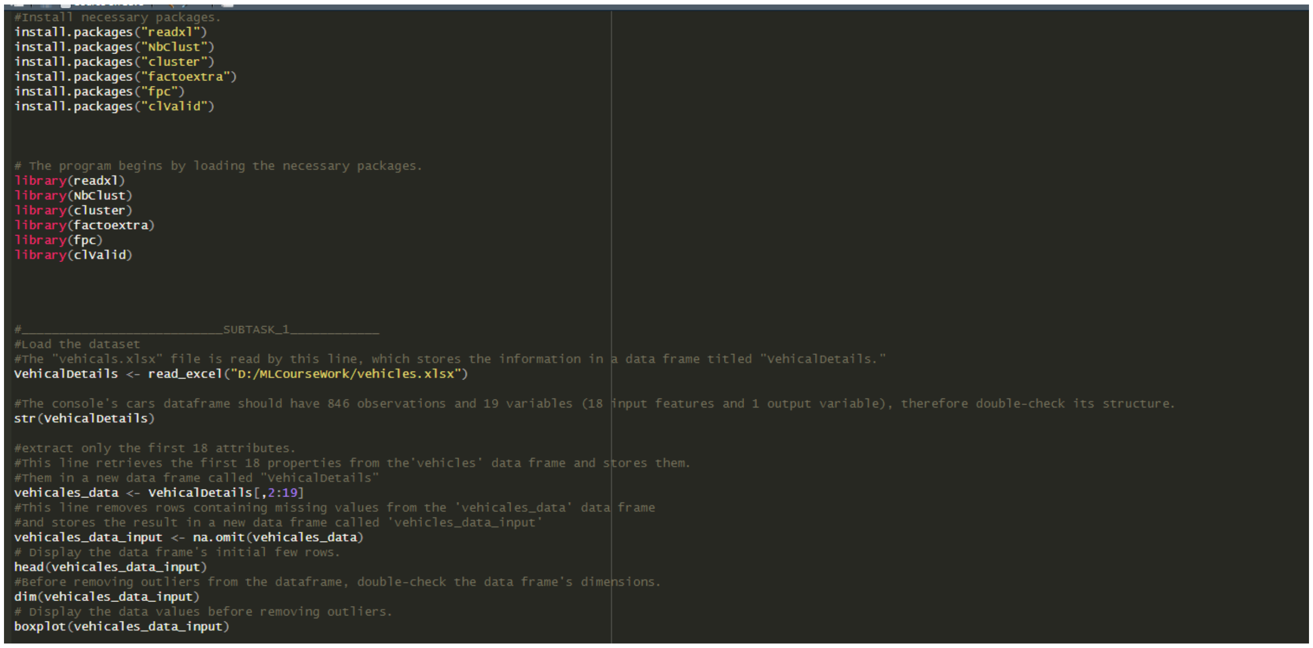




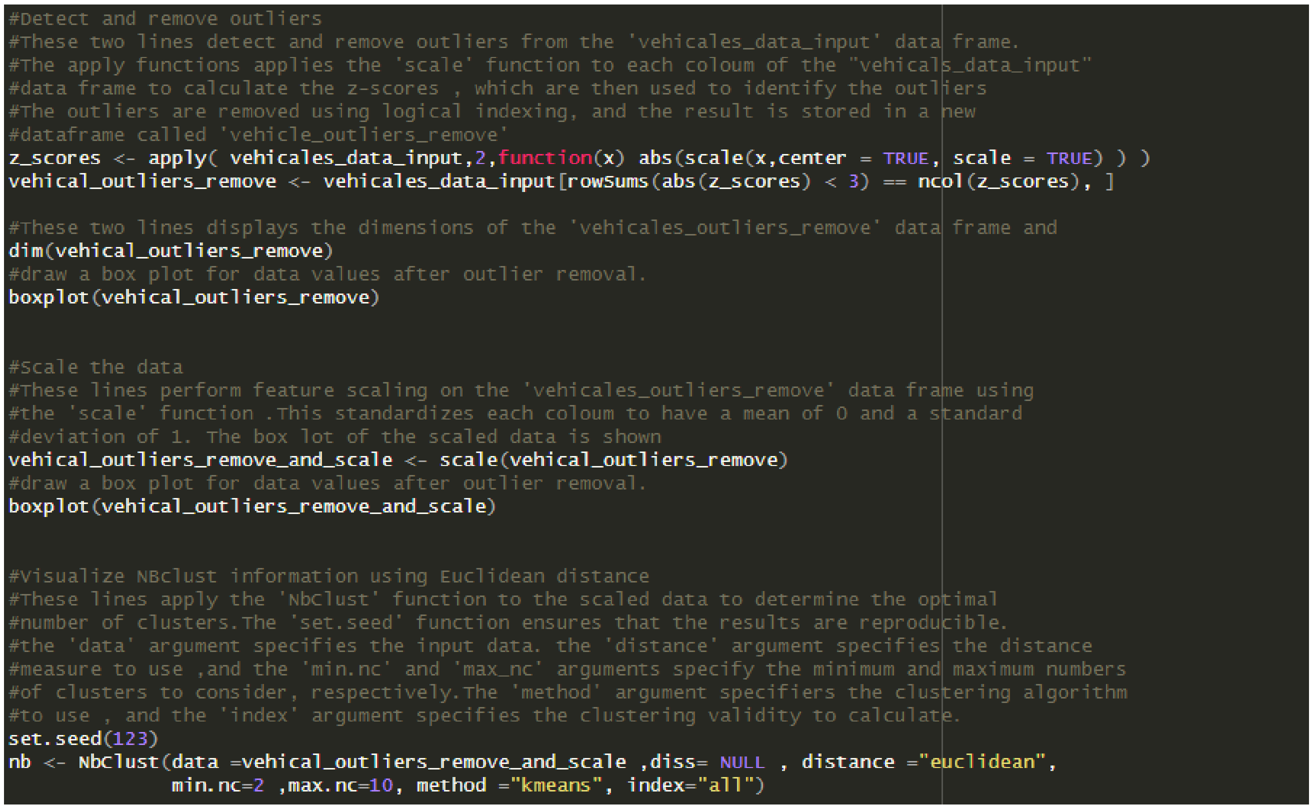
```
> # Draw a Calinski-Harabasz index plot.
> gap <- clusGap(vehicels_transform , FUN = kmeans ,K.max = 10 , nstart = 25, iter.max = 20 )
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 100)  [one "." per sample]:
.................................................. 50
.................................................. 100
> plot(gap, main="Calinski-Harbasz Index ")
> #Print the optimal number of clusters
> print(gap$Tab)
          logW   E.logW       gap      SE.sim
 [1,] 7.029069 7.447248 0.4181793 0.006932648
 [2,] 6.752963 7.287123 0.5341597 0.005661836
 [3,] 6.623568 7.226491 0.6029238 0.005651129
 [4,] 6.569040 7.176226 0.6071854 0.006453479
 [5,] 6.518606 7.140049 0.6214429 0.006130520
 [6,] 6.481881 7.108082 0.6262008 0.005824964
 [7,] 6.447261 7.082022 0.6347607 0.005813212
 [8,] 6.415419 7.058499 0.6430808 0.006117061
 [9,] 6.383883 7.038344 0.6544602 0.005975376
[10,] 6.356297 7.020486 0.6641896 0.006031970
```

The values of the Calinski-Harabasz index as a function of cluster number are shown in the above plot. The diagram demonstrates the data's ability to be successfully grouped into various numbers of clusters. The right number of clusters can be calculated using the "elbow", or the point where the Calinski-Harabasz index starts to level off.

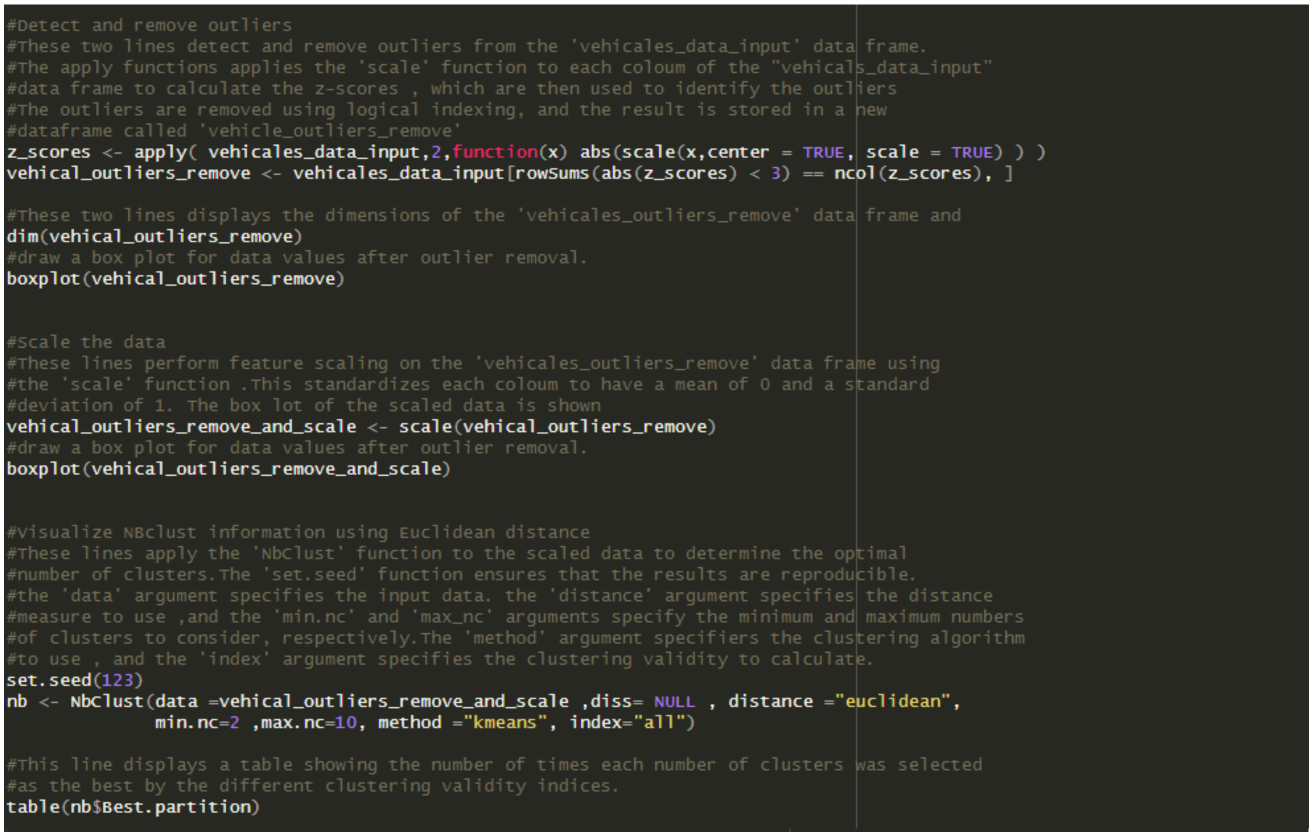
1.3 Appendix



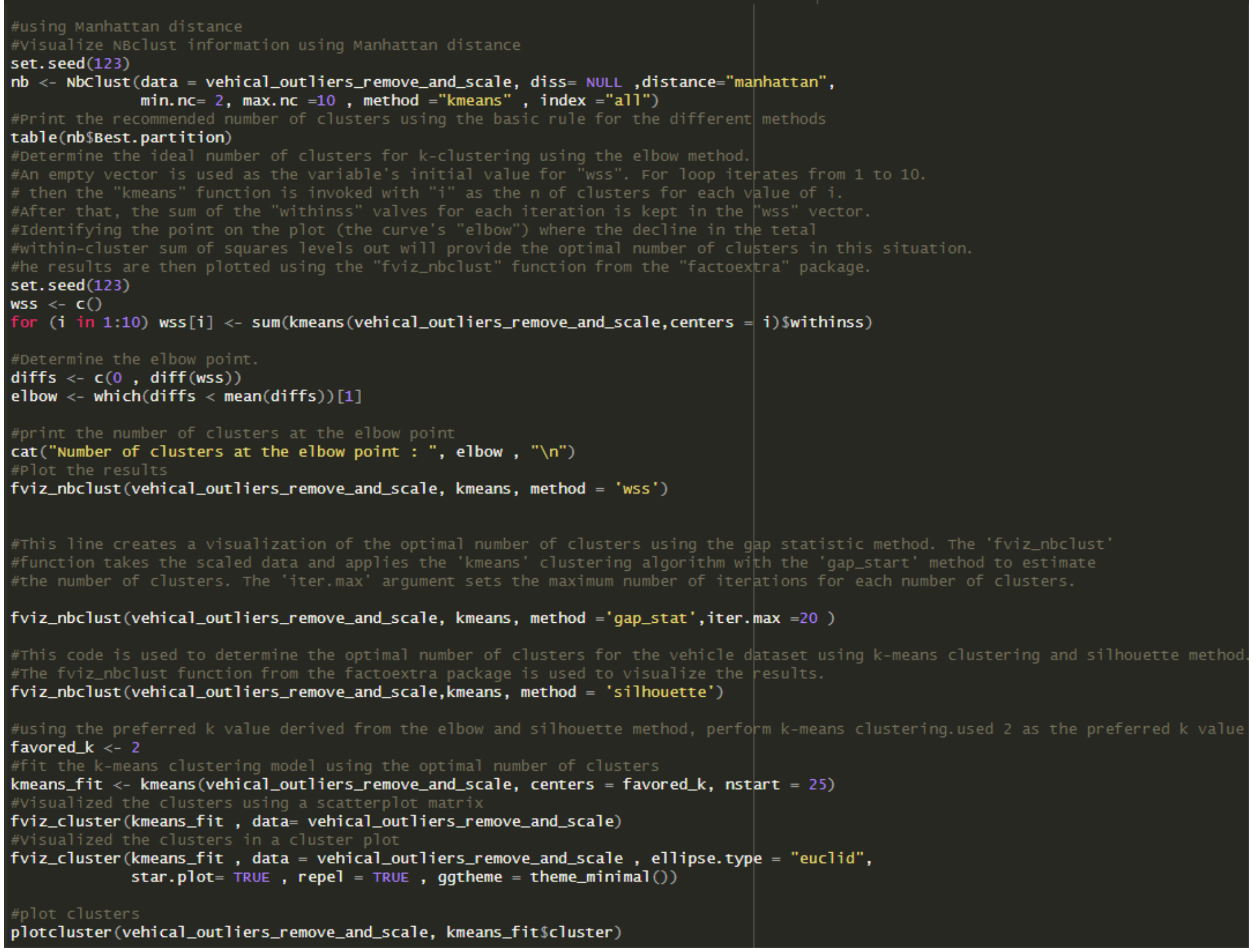
```r
#Install necessary packages.
install.packages("readxl")
install.packages("NbClust")
install.packages("cluster")
install.packages("factoextra")
install.packages("fpc")
install.packages("clValid")



# The program begins by loading the necessary packages.
library(readxl)
library(NbClust)
library(cluster)
library(factoextra)
library(fpc)
library(clValid)




#____________________________SUBTASK_1____________
#Load the dataset
#The "vehicals.xlsx" file is read by this line, which stores the information in a data frame titled "VehicalDetails."
VehicalDetails <- read_excel("D:/MLCourseWork/vehicles.xlsx")

#The console's cars dataframe should have 846 observations and 19 variables (18 input features and 1 output variable), therefore double-check its structure.
str(VehicalDetails)

#extract only the first 18 attributes.
#This line retrieves the first 18 properties from the'vehicles' data frame and stores them.
#Them in a new data frame called "VehicalDetails"
vehicales_data <- VehicalDetails[,2:19]
#This line removes rows containing missing values from the 'vehicales_data' data frame
#and stores the result in a new data frame called 'vehicles_data_input'
vehicales_data_input <- na.omit(vehicales_data)
# Display the data frame's initial few rows.
head(vehicales_data_input)
#Before removing outliers from the dataframe, double-check the data frame's dimensions.
dim(vehicales_data_input)
# Display the data values before removing outliers.
boxplot(vehicales_data_input)
```



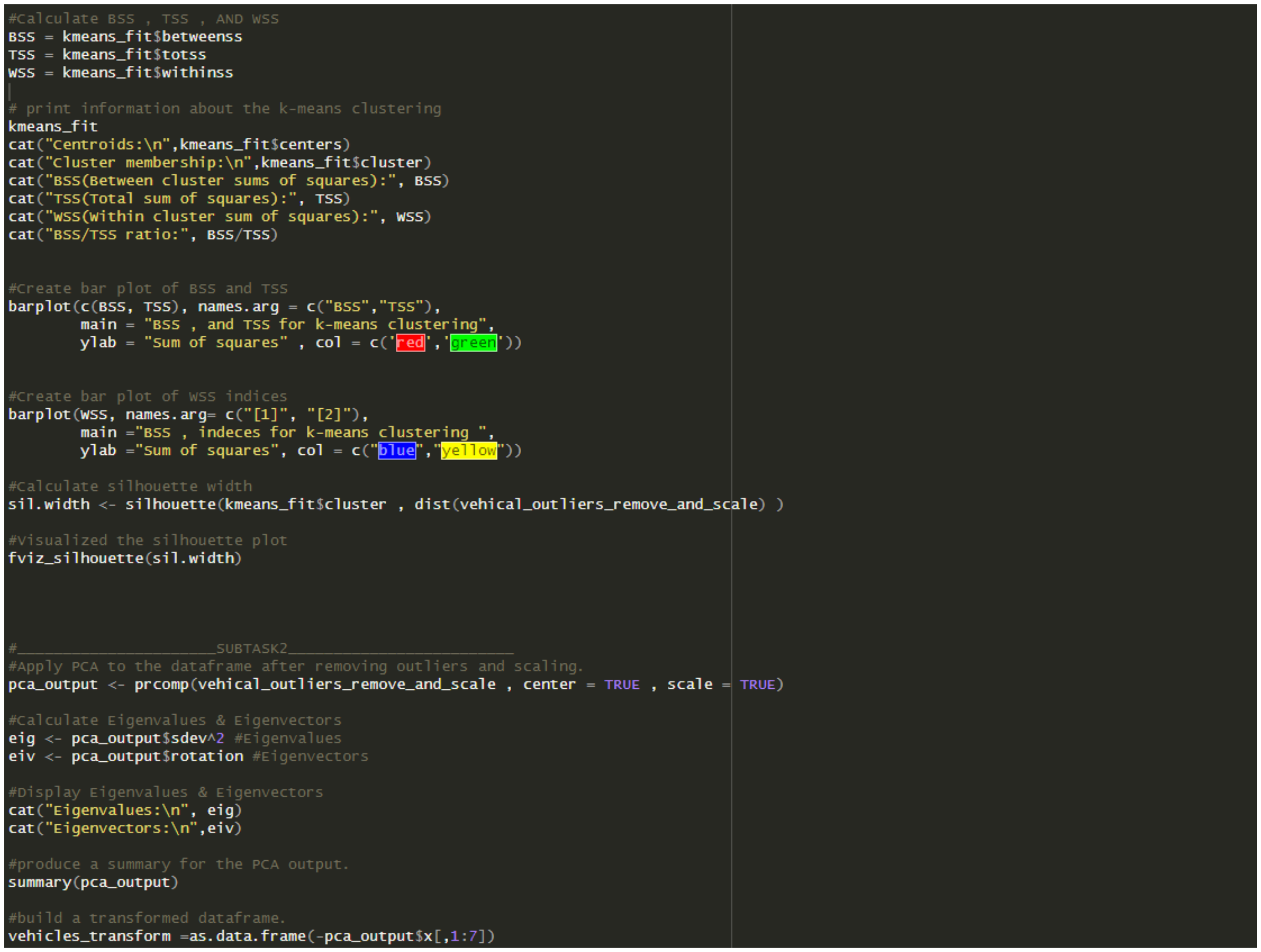
```r
#Detect and remove outliers
#These two lines detect and remove outliers from the 'vehicales_data_input' data frame.
#The apply functions applies the 'scale' function to each coloum of the "vehicals_data_input"
#data frame to calculate the z-scores , which are then used to identify the outliers
#The outliers are removed using logical indexing, and the result is stored in a new
#dataframe called 'vehicle_outliers_remove'
z_scores <- apply( vehicales_data_input,2,function(x) abs(scale(x,center = TRUE, scale = TRUE) ) )
vehical_outliers_remove <- vehicales_data_input[rowSums(abs(z_scores) < 3) == ncol(z_scores), ]

#These two lines displays the dimensions of the 'vehicales_outliers_remove' data frame and
dim(vehical_outliers_remove)
#draw a box plot for data values after outlier removal.
boxplot(vehical_outliers_remove)


#Scale the data
#These lines perform feature scaling on the 'vehicales_outliers_remove' data frame using
#the 'scale' function .This standardizes each coloum to have a mean of 0 and a standard
#deviation of 1. The box lot of the scaled data is shown
vehical_outliers_remove_and_scale <- scale(vehical_outliers_remove)
#draw a box plot for data values after outlier removal.
boxplot(vehical_outliers_remove_and_scale)


#Visualize NBclust information using Euclidean distance
#These lines apply the 'NbClust' function to the scaled data to determine the optimal
#number of clusters.The 'set.seed' function ensures that the results are reproducible.
#the 'data' argument specifies the input data. the 'distance' argument specifies the distance
#measure to use ,and the 'min.nc' and 'max_nc' arguments specify the minimum and maximum numbers
#of clusters to consider, respectively.The 'method' argument specifiers the clustering algorithm
#to use , and the 'index' argument specifies the clustering validity to calculate.
set.seed(123)
nb <- NbClust(data =vehical_outliers_remove_and_scale ,diss= NULL , distance ="euclidean",
              min.nc=2 ,max.nc=10, method ="kmeans", index="all")
```



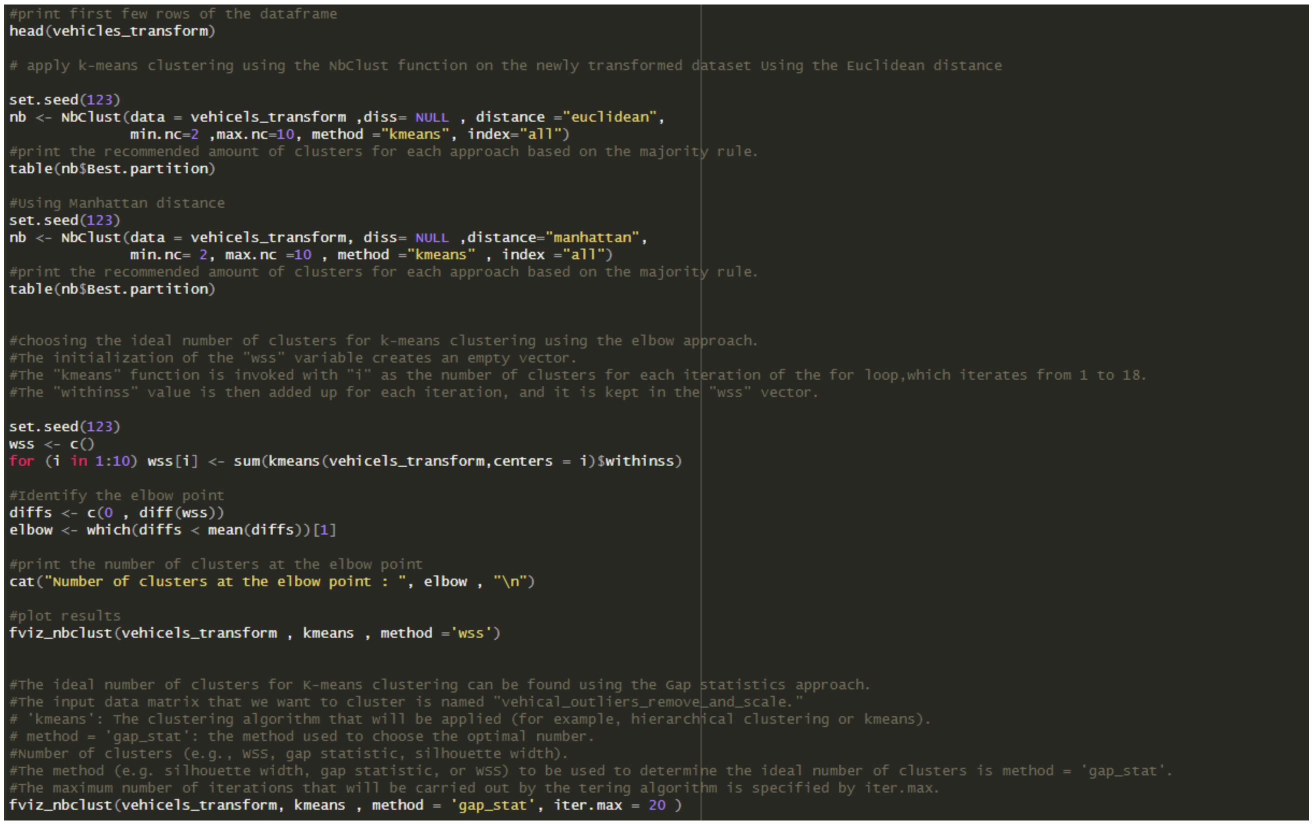
```
#Detect and remove outliers
#These two lines detect and remove outliers from the 'vehicales_data_input' data frame.
#The apply functions applies the 'scale' function to each coloum of the "vehicals_data_input"
#data frame to calculate the z-scores , which are then used to identify the outliers
#The outliers are removed using logical indexing, and the result is stored in a new
#dataframe called 'vehicle_outliers_remove'
z_scores <- apply( vehicales_data_input,2,function(x) abs(scale(x,center = TRUE, scale = TRUE) ) )
vehical_outliers_remove <- vehicales_data_input[rowSums(abs(z_scores) < 3) == ncol(z_scores), ]

#These two lines displays the dimensions of the 'vehicales_outliers_remove' data frame and
dim(vehical_outliers_remove)
#draw a box plot for data values after outlier removal.
boxplot(vehical_outliers_remove)


#Scale the data
#These lines perform feature scaling on the 'vehicales_outliers_remove' data frame using
#the 'scale' function .This standardizes each coloum to have a mean of 0 and a standard
#deviation of 1. The box lot of the scaled data is shown
vehical_outliers_remove_and_scale <- scale(vehical_outliers_remove)
#draw a box plot for data values after outlier removal.
boxplot(vehical_outliers_remove_and_scale)


#Visualize NBclust information using Euclidean distance
#These lines apply the 'NbClust' function to the scaled data to determine the optimal
#number of clusters.The 'set.seed' function ensures that the results are reproducible.
#the 'data' argument specifies the input data. the 'distance' argument specifies the distance
#measure to use ,and the 'min.nc' and 'max_nc' arguments specify the minimum and maximum numbers
#of clusters to consider, respectively.The 'method' argument specifiers the clustering algorithm
#to use , and the 'index' argument specifies the clustering validity to calculate.
set.seed(123)
nb <- NbClust(data =vehical_outliers_remove_and_scale ,diss= NULL , distance ="euclidean",
              min.nc=2 ,max.nc=10, method ="kmeans", index="all")

#This line displays a table showing the number of times each number of clusters was selected
#as the best by the different clustering validity indices.
table(nb$Best.partition)
```



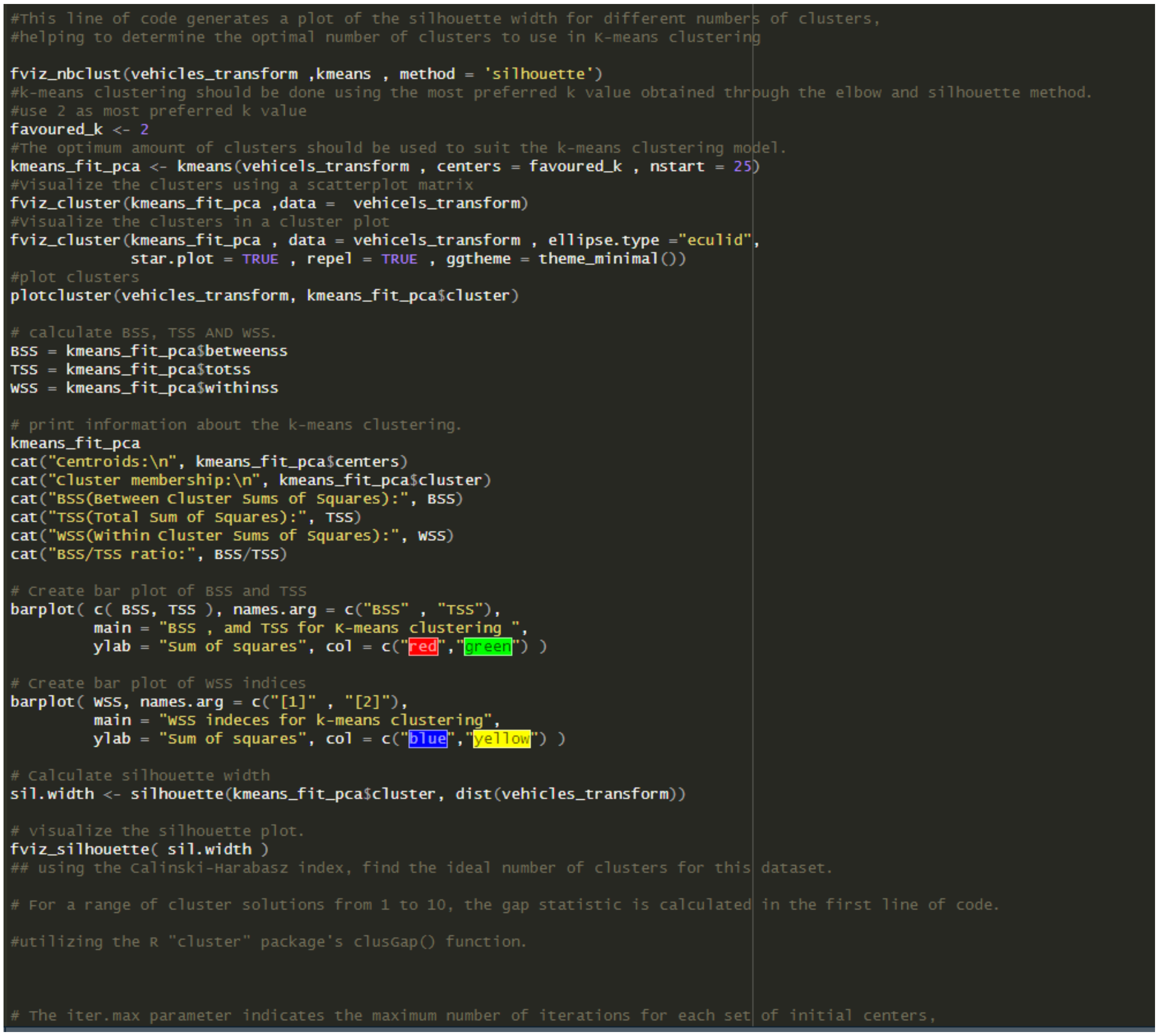
```
#using Manhattan distance
#Visualize NBclust information using Manhattan distance
set.seed(123)
nb <- NbClust(data = vehical_outliers_remove_and_scale, diss= NULL ,distance="manhattan",
             min.nc= 2, max.nc =10 , method ="kmeans" , index ="all")
#Print the recommended number of clusters using the basic rule for the different methods
table(nb$Best.partition)
#Determine the ideal number of clusters for k-clustering using the elbow method.
#An empty vector is used as the variable's initial value for "wss". For loop iterates from 1 to 10.
# then the "kmeans" function is invoked with "i" as the n of clusters for each value of i.
#After that, the sum of the "withinss" valves for each iteration is kept in the "wss" vector.
#Identifying the point on the plot (the curve's "elbow") where the decline in the tetal
#within-cluster sum of squares levels out will provide the optimal number of clusters in this situation.
#he results are then plotted using the "fviz_nbclust" function from the "factoextra" package.
set.seed(123)
wss <- c()
for (i in 1:10) wss[i] <- sum(kmeans(vehical_outliers_remove_and_scale,centers = i)$withinss)

#Determine the elbow point.
diffs <- c(0 , diff(wss))
elbow <- which(diffs < mean(diffs))[1]

#print the number of clusters at the elbow point
cat("Number of clusters at the elbow point : ", elbow , "\n")
#Plot the results
fviz_nbclust(vehical_outliers_remove_and_scale, kmeans, method = 'wss')


#This line creates a visualization of the optimal number of clusters using the gap statistic method. The 'fviz_nbclust'
#function takes the scaled data and applies the 'kmeans' clustering algorithm with the 'gap_start' method to estimate
#the number of clusters. The 'iter.max' argument sets the maximum number of iterations for each number of clusters.

fviz_nbclust(vehical_outliers_remove_and_scale, kmeans, method ='gap_stat',iter.max =20 )

#This code is used to determine the optimal number of clusters for the vehicle dataset using k-means clustering and silhouette method.
#The fviz_nbclust function from the factoextra package is used to visualize the results.
fviz_nbclust(vehical_outliers_remove_and_scale,kmeans, method = 'silhouette')

#using the preferred k value derived from the elbow and silhouette method, perform k-means clustering.used 2 as the preferred k value
favored_k <- 2
#fit the k-means clustering model using the optimal number of clusters
kmeans_fit <- kmeans(vehical_outliers_remove_and_scale, centers = favored_k, nstart = 25)
#Visualized the clusters using a scatterplot matrix
fviz_cluster(kmeans_fit , data= vehical_outliers_remove_and_scale)
#Visualized the clusters in a cluster plot
fviz_cluster(kmeans_fit , data = vehical_outliers_remove_and_scale , ellipse.type = "euclid",
             star.plot= TRUE , repel = TRUE , ggtheme = theme_minimal())

#plot clusters
plotcluster(vehical_outliers_remove_and_scale, kmeans_fit$cluster)
```



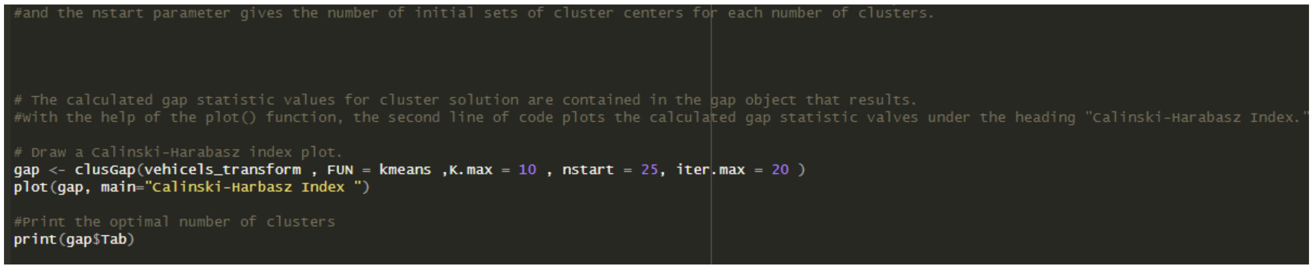
```r
#Calculate BSS , TSS , AND WSS
BSS = kmeans_fit$betweenss
TSS = kmeans_fit$totss
WSS = kmeans_fit$withinss

# print information about the k-means clustering
kmeans_fit
cat("Centroids:\n",kmeans_fit$centers)
cat("Cluster membership:\n",kmeans_fit$cluster)
cat("BSS(Between cluster sums of squares):", BSS)
cat("TSS(Total sum of squares):", TSS)
cat("WSS(Within cluster sum of squares):", WSS)
cat("BSS/TSS ratio:", BSS/TSS)


#Create bar plot of BSS and TSS
barplot(c(BSS, TSS), names.arg = c("BSS","TSS"),
        main = "BSS , and TSS for k-means clustering",
        ylab = "Sum of squares" , col = c('red','green'))


#Create bar plot of WSS indices
barplot(WSS, names.arg= c("[1]", "[2]"),
        main ="BSS , indeces for k-means clustering ",
        ylab ="Sum of squares", col = c("blue","yellow"))

#Calculate silhouette width
sil.width <- silhouette(kmeans_fit$cluster , dist(vehical_outliers_remove_and_scale) )

#Visualized the silhouette plot
fviz_silhouette(sil.width)




#______________________SUBTASK2_________________________
#Apply PCA to the dataframe after removing outliers and scaling.
pca_output <- prcomp(vehical_outliers_remove_and_scale , center = TRUE , scale = TRUE)

#Calculate Eigenvalues & Eigenvectors
eig <- pca_output$sdev^2 #Eigenvalues
eiv <- pca_output$rotation #Eigenvectors

#Display Eigenvalues & Eigenvectors
cat("Eigenvalues:\n", eig)
cat("Eigenvectors:\n",eiv)

#produce a summary for the PCA output.
summary(pca_output)

#build a transformed dataframe.
vehicles_transform =as.data.frame(-pca_output$x[,1:7])
```



```r
#print first few rows of the dataframe
head(vehicles_transform)

# apply k-means clustering using the NbClust function on the newly transformed dataset Using the Euclidean distance

set.seed(123)
nb <- NbClust(data = vehicels_transform ,diss= NULL , distance ="euclidean",
              min.nc=2 ,max.nc=10, method ="kmeans", index="all")
#print the recommended amount of clusters for each approach based on the majority rule.
table(nb$Best.partition)

#Using Manhattan distance
set.seed(123)
nb <- NbClust(data = vehicels_transform, diss= NULL ,distance="manhattan",
              min.nc= 2, max.nc =10 , method ="kmeans" , index ="all")
#print the recommended amount of clusters for each approach based on the majority rule.
table(nb$Best.partition)


#choosing the ideal number of clusters for k-means clustering using the elbow approach.
#The initialization of the "wss" variable creates an empty vector.
#The "kmeans" function is invoked with "i" as the number of clusters for each iteration of the for loop,which iterates from 1 to 18.
#The "withinss" value is then added up for each iteration, and it is kept in the "wss" vector.

set.seed(123)
wss <- c()
for (i in 1:10) wss[i] <- sum(kmeans(vehicels_transform,centers = i)$withinss)

#Identify the elbow point
diffs <- c(0 , diff(wss))
elbow <- which(diffs < mean(diffs))[1]

#print the number of clusters at the elbow point
cat("Number of clusters at the elbow point : ", elbow , "\n")

#plot results
fviz_nbclust(vehicels_transform , kmeans , method ='wss')


#The ideal number of clusters for K-means clustering can be found using the Gap statistics approach.
#The input data matrix that we want to cluster is named "vehical_outliers_remove_and_scale."
# 'kmeans': The clustering algorithm that will be applied (for example, hierarchical clustering or kmeans).
# method = 'gap_stat': the method used to choose the optimal number.
#Number of clusters (e.g., WSS, gap statistic, silhouette width).
#The method (e.g. silhouette width, gap statistic, or WSS) to be used to determine the ideal number of clusters is method = 'gap_stat'.
#The maximum number of iterations that will be carried out by the tering algorithm is specified by iter.max.
fviz_nbclust(vehicels_transform, kmeans , method = 'gap_stat', iter.max = 20 )
```



```
#This line of code generates a plot of the silhouette width for different numbers of clusters,
#helping to determine the optimal number of clusters to use in K-means clustering

fviz_nbclust(vehicles_transform ,kmeans , method = 'silhouette')
#k-means clustering should be done using the most preferred k value obtained through the elbow and silhouette method.
#use 2 as most preferred k value
favoured_k <- 2
#The optimum amount of clusters should be used to suit the k-means clustering model.
kmeans_fit_pca <- kmeans(vehicels_transform , centers = favoured_k , nstart = 25)
#Visualize the clusters using a scatterplot matrix
fviz_cluster(kmeans_fit_pca ,data =  vehicels_transform)
#Visualize the clusters in a cluster plot
fviz_cluster(kmeans_fit_pca , data = vehicels_transform , ellipse.type ="eculid",
             star.plot = TRUE , repel = TRUE , ggtheme = theme_minimal())
#plot clusters
plotcluster(vehicles_transform, kmeans_fit_pca$cluster)

# calculate BSS, TSS AND WSS.
BSS = kmeans_fit_pca$betweenss
TSS = kmeans_fit_pca$totss
WSS = kmeans_fit_pca$withinss

# print information about the k-means clustering.
kmeans_fit_pca
cat("Centroids:\n", kmeans_fit_pca$centers)
cat("Cluster membership:\n", kmeans_fit_pca$cluster)
cat("BSS(Between Cluster Sums of Squares):", BSS)
cat("TSS(Total Sum of Squares):", TSS)
cat("WSS(Within Cluster Sums of Squares):", WSS)
cat("BSS/TSS ratio:", BSS/TSS)

# Create bar plot of BSS and TSS
barplot( c( BSS, TSS ), names.arg = c("BSS" , "TSS"),
         main = "BSS , amd TSS for K-means clustering ",
         ylab = "Sum of squares", col = c("red","green") )

# Create bar plot of WSS indices
barplot( WSS, names.arg = c("[1]" , "[2]"),
         main = "WSS indeces for k-means clustering",
         ylab = "Sum of squares", col = c("blue","yellow") )

# Calculate silhouette width
sil.width <- silhouette(kmeans_fit_pca$cluster, dist(vehicles_transform))

# visualize the silhouette plot.
fviz_silhouette( sil.width )
## using the Calinski-Harabasz index, find the ideal number of clusters for this dataset.

# For a range of cluster solutions from 1 to 10, the gap statistic is calculated in the first line of code.

#utilizing the R "cluster" package's clusGap() function.



# The iter.max parameter indicates the maximum number of iterations for each set of initial centers,
```



```
#and the nstart parameter gives the number of initial sets of cluster centers for each number of clusters.




# The calculated gap statistic values for cluster solution are contained in the gap object that results.
#With the help of the plot() function, the second line of code plots the calculated gap statistic valves under the heading "Calinski-Harabasz Index."

# Draw a Calinski-Harabasz index plot.
gap <- clusGap(vehicels_transform , FUN = kmeans ,K.max = 10 , nstart = 25, iter.max = 20 )
plot(gap, main="Calinski-Harbasz Index ")

#Print the optimal number of clusters
print(gap$Tab)
```

********End of first subtask********

1. Multi-layer Neural Network (MLP-NN) Part

a) Weather data, historical load data, calendar information, and other relevant information are frequently used as input variables in MLP models for electricity load forecasting. Since it directly affects the neural network model's capacity to correctly estimate electricity demand, the definition of the input vector is essential. This input vector is defined using a variety of schemes and methodologies, with the following examples representing some of the more popular ones:

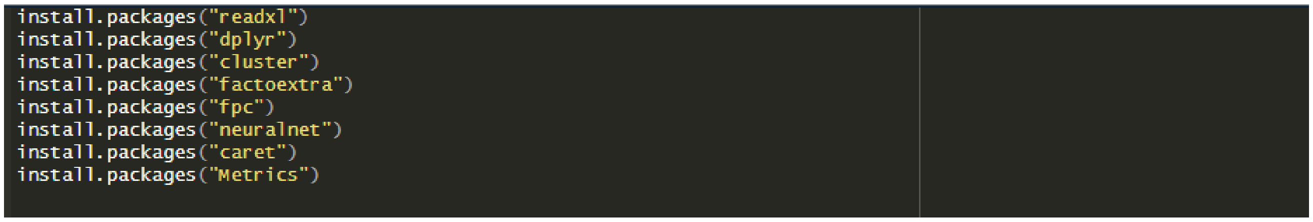
1. Fourier analysis: A method for breaking down a time series into its constituent frequencies is Fourier analysis. This approach is efficient for gathering data with seasonal and periodic variations, and the resulting frequency components can be used as model inputs.
2. Wavelet analysis: A time series is broken down into its constituent wavelets by wavelet analysis. By capturing both short-term and long-term relationships in the data, this method can be utilized to create a more robust input vector that is more accurate.
3. Autoregressive integrated moving average (ARIMA) approach: To make the time series stationary, this approach combines differencing, the AR, and the MA strategies. Both short- and long-term load relationships are capable of being captured using ARIMA models.
4. Exogenous variables: The input vector may also contain exogenous variables in addition to the target variable's prior values. These variables can improve prediction accuracy by helping the model grasp long-term dependencies in the data.
5. Autoregressive (AR) models: The target variable's previous values are used as inputs in AR models. For example, the load from the previous hour, day, or week could be used as a model input for predicting electricity load. Although easy to build and interpret, AR models are limited to short-term data dependencies. Both the seasonal autoregressive (SAR) and autoregressive integrated moving average (ARIMA) models are created by extending the AR model to add seasonal and trend components.
6. Moving average (MA) approach: The average of the target variable's previous values is the input for this technique. The MA model is capable of smoothing down the noise while capturing the load trend.
7. Seasonal Autoregressive Integrated Moving Average (SARIMA) approach: this approach builds on the ARIMA paradigm by including seasonal differencing to capture seasonal load changes.
8. Nonlinear autoregressive exogenous (NARX) approach: This approach considers the target variable's lags as well as any exogenous variables that might affect the load. The nonlinear interactions between load and exogenous variables can be captured using the NARX model.

Finally, the specific application and the accessible data determine the input variables and methods used to define the input vector in MLP models for forecasting electrical load. The data and the current circumstances need to be carefully reviewed in order to choose the best approach.

2.1. First Subtask

2.1.1 Install Necessary Libraries

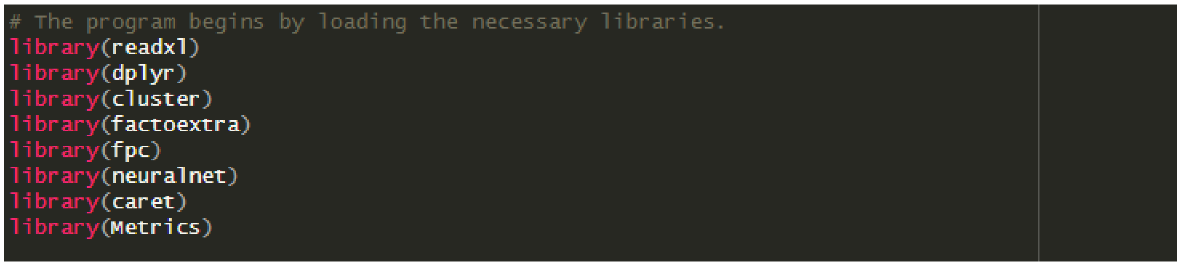
Install every required library that is used in the MLP neural network component for the forecasting



```
install.packages("readxl")
install.packages("dplyr")
install.packages("cluster")
install.packages("factoextra")
install.packages("fpc")
install.packages("neuralnet")
install.packages("caret")
install.packages("Metrics")
```

2.1.2 Import Necessary Libraries

Import each and every required library for the MLP neural network part of the forecasting problem.

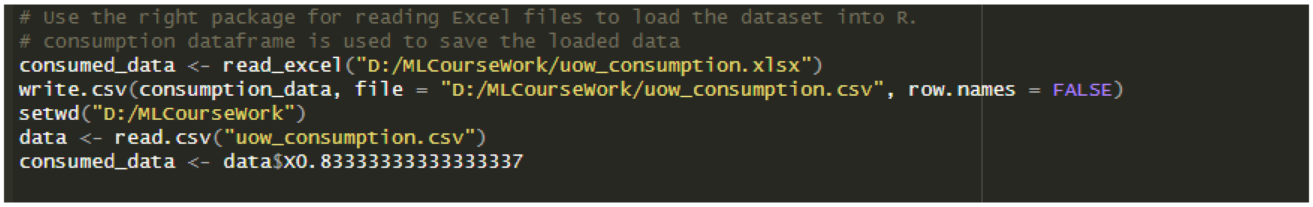


```
# The program begins by loading the necessary libraries.
library(readxl)
library(dplyr)
library(cluster)
library(factoextra)
library(fpc)
library(neuralnet)
library(caret)
library(Metrics)
```

2.1.3 Read the Excel File

Analyzed the Excel document (uow_consumption.xlsx), which includes 470 observations and daily electricity usage data for three hours (20:00, 19:00, and 18:00) for the 2018 and some of the 2019 periods. For this multilayer neural network (MLP-NN) component, the dataset's first 380 samples will be utilized as training data, while the remaining samples will be used as the testing set.

Relevant the code:

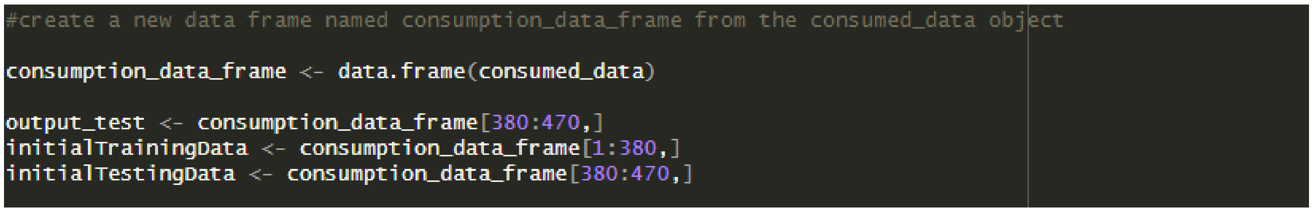


```r
# Use the right package for reading Excel files to load the dataset into R.
# consumption dataframe is used to save the loaded data
consumed_data <- read_excel("D:/MLCourseWork/uow_consumption.xlsx")
write.csv(consumption_data, file = "D:/MLCourseWork/uow_consumption.csv", row.names = FALSE)
setwd("D:/MLCourseWork")
data <- read.csv("uow_consumption.csv")
consumed_data <- data$X0.83333333333333337
```

2.1.4 Divide the Dataset

The dataset will be split into train and test data using the following code.

Relevant code:



```r
#create a new data frame named consumption_data_frame from the consumed_data object

consumption_data_frame <- data.frame(consumed_data)

output_test <- consumption_data_frame[380:470,]
initialTrainingData <- consumption_data_frame[1:380,]
initialTestingData <- consumption_data_frame[380:470,]
```

2.1.5 Normalise Data

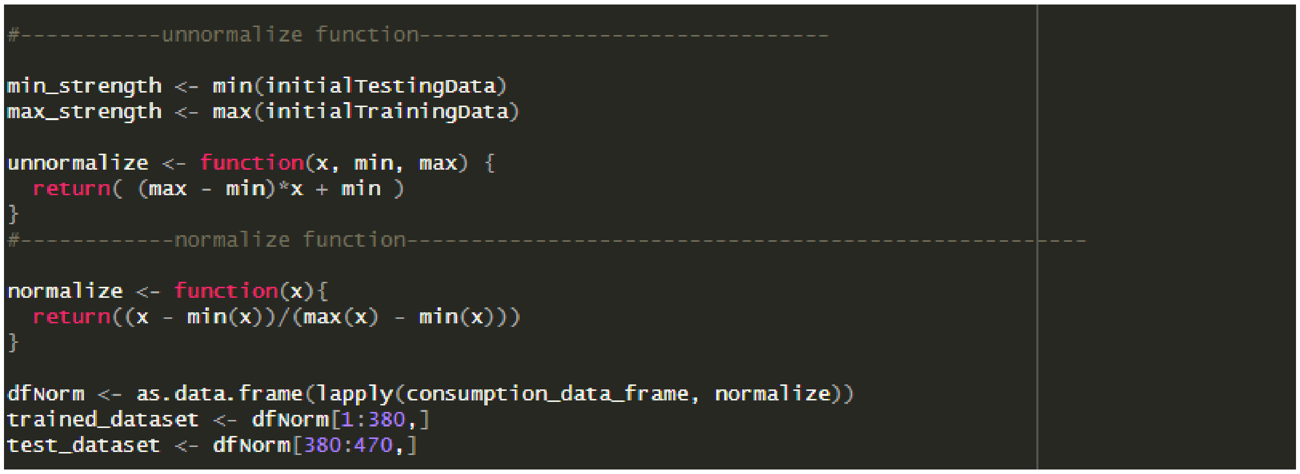
When data is normalized, it is generally converted to a standard range or distribution, such as a mean of zero and a standard deviation of one. By normalizing the input data, we can make sure that all of the features have comparable ranges and distributions, preventing features with large values from dominating the training process and compromising its accuracy.

The data were normalized for this scenario using the min-max normalization. Before using data in an MLP structure, data normalization is an essential pre-processing step. This is because the MLP algorithm is sensitive to the data scale, and data with different scales may produce scales that might result in biased or inefficient training outcomes.

Additionally, by reducing input data variance and avoiding problems with numerical instability, normalizing the data may help to enhance the MLP algorithm's convergence during training.

In a nutshell, normalizing the input data is a common and important pre-processing step for MLPs since it ensures that the data have a constant scale and distribution, which can improve the accuracy, stability, and efficiency of the training process.

The relevant code:


```
#-----------unnormalize function-------------------------------

min_strength <- min(initialTestingData)
max_strength <- max(initialTrainingData)

unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}
#------------normalize function------------------------------------------------------

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

dfNorm <- as.data.frame(lapply(consumption_data_frame, normalize))
trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]
```

2.1.6 RMSE, MAE, MAPE & sMAPE

Root Mean Squared Error (RMSE)

The accuracy of a prediction model is frequently evaluated using the Root Mean Squared Error (RMSE), particularly in the field of regression analysis. To determine the difference between predicted and actual values of a target variable, it computes the square root of the average of the squared differences between predicted and actual values.

A regression model's residuals, or the differences between predicted and actual values, are calculated using the RMSE formula to determine their standard deviation. Given that it is represented in the same units as the target variable, it is easy to interpret and compare different models and data sets.

Because the predicted values are more closely aligned with the actual values, a lower RMSE shows that the model is better fitting the data. However, RMSE is sensitive to outliers and large errors since the squared differences are magnified in the calculation. In order to provide a more thorough assessment of the model's accuracy, it should be combined with other measures of model performance, such as Mean Absolute Error (MAE).

Mean Absolute Error (MAE)

The accuracy of a prediction or forecasting model is often evaluated using the statistical metric known as Mean Absolute Error (MAE). It measures the average absolute difference between a target variable's predicted and actual values.

MAE is calculated by the average of the absolute differences between the predicted and actual values. MAE gives all errors equal weight regardless of magnitude, in contrast to the Root Mean Squared Error (RMSE), which computes the average squared difference between predicted and actual data. This means that MAE is a more reliable indicator of model performance in the presence of such errors because it is less sensitive to outliers and significant errors.

As MAE is expressed in the same units as the target variable, comparing it between models and data sets is simple. Because predicted values are more closely aligned with the actual values, a lower MAE indicates that the model is better fitting the data.

Outliers and significant errors can affect the performance of other measures like RMSE, MAE is a useful indicator of prediction model accuracy.

Mean Absolute Percentage Error (MAPE)

The average absolute percentage difference between a target variable's predicted and actual values is calculated using the Mean Absolute Percentage Error (MAPE) formula.

The formula for MAPE is to take the absolute percentage differences between the predicted and actual values and average them, where the absolute percentage difference is obtained by dividing the difference between the predicted and actual values by the actual value multiplied by 100%. Given as a percentage, MAPE makes it easier to interpret comparing different models and data sets. A low MAPE denotes a prediction model with a higher accuracy.

Because it offers a relative measure of error that is independent of data scale, MAPE is a useful indicator of prediction model accuracy, particularly when working with data sets of different scales.

The drawbacks of MAPE, include the fact that it is sensitive to zero values and that it can produce infinite numbers when its actual value is zero. Furthermore, MAPE is not symmetric because it is based on percentage differences, which indicates the error related to overestimation is handled differently than the mistake associated with underestimating.

Symmetric Mean Absolute Percentage Error (sMAPE)

The average absolute percentage difference between the predicted and the actual values of a target variable is measured symmetrically by symmetric mean absolute percentage error (sMAPE).

This MAPE index variant allows for the average of predicted and actual values in the denominator as well as the mean of the absolute percentage differences between predicted and actual values. When the data includes significant fluctuations, sMAPE can be especially useful in evaluating model accuracy.

2.1.7 Table of comparisons for testing performance based on statistical metrics.

NN Structure Description and Parameters	RMSE	MAE	MAPE	Symmetric MAP
NN 1: One hidden layer, 5 neurons, act.fct = "logistic"	4.508625	3.889591	0.1037533	0.09441911
NN 2: One hidden layer, 10 neurons, act.fct = "logistic"	4.174276	3.448107	0.0492812	0.09256854
NN 3: One hidden layer, 20 neurons, act.fct = "tanh"	4.209634	3.944821	0.1033886	0.09642614
NN 4: One hidden layer, 30 neurons, act.fct = "logistic"	4.61841	3.764855	0.1022908	0.09381068
NN 5: One hidden layer, 40 neurons, act.fct = " tanh "	4.65329	3.96105	0.1276577	0.08894643
NN 6: Two hidden layer, 5 - 5 neurons, act.fct = "tach"	4.397435	3.663863	0.1749402	0.07253295
NN 7: Two hidden layer, 10-10 neurons, act.fct = "logistic"	4.331341	3.752129	0.0883431	0.0098201
NN 9: Three hidden layer, 10-10-10 neurons, act.fct = "logistic"	4.268134	4.109412	0.14349645	0.04492326
NN 10: Three hidden layer, 20-20-20 neurons, act.fct = " tanh "	7.612296	5.348315	0.1368937	0.1421611
NN 11: Three hidden layer, 8-15-8 neurons, act.fct = "logistic"	5.489659	4.653337	0.1811973	0.8907388
NN 12: Three hidden layer, 10-40-10 neurons, act.fct = " tanh "	6.089733	4.095614	0.2456643	0.9909362
NN 13: Four hidden layer, 5-5-5-5 neurons, act.fct = "logistic"	4.97552	4.562122	0.6881786	0.681337
NN 14: Four hidden layer, 10-15-15-10 neurons, act.fct = "tanh"	5.426137	4.868509	0.967519	0.2309092

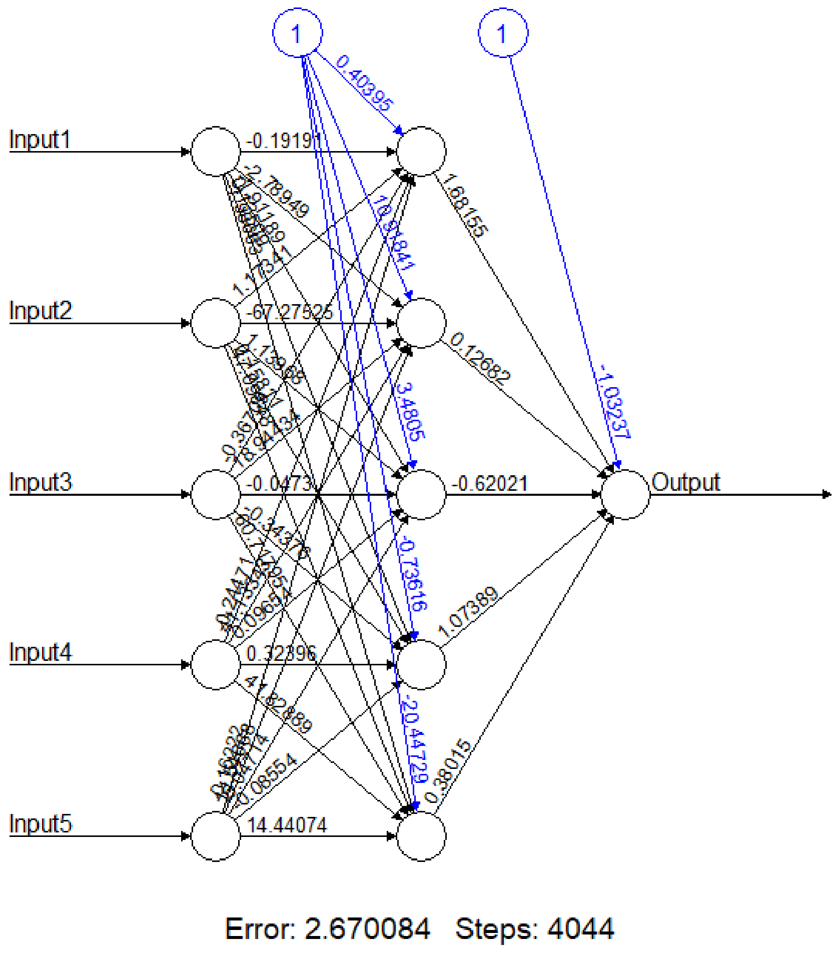
2.1.8 Efficiency is produced by the top one- and two-hidden-layer networks with the corresponding parameters.

<u>The Best One hidden Layer Neural Network</u>

According to the results, the MLP neural network with the following parameters produced the best performance.

Hidden Layers = c(5)

Activation Function = “logistic”



Number of weights between input and hidden layer:-

Number of inputs x Number of Neurons in the hidden layer = (5+1) x 5 = 30

Number of weights between hidden layer and output layer: -

Number of Neurons in the hidden layer x Number of Neurons in output layer = (5+1) x 1 = 6

The number of neurons in the hidden layer and the number of neurons in the output layer define how many weights there are between the hidden layer and the output layer in a multilayer perceptron neural network.

Total Number of weight parameters = 30 + 6 = 36

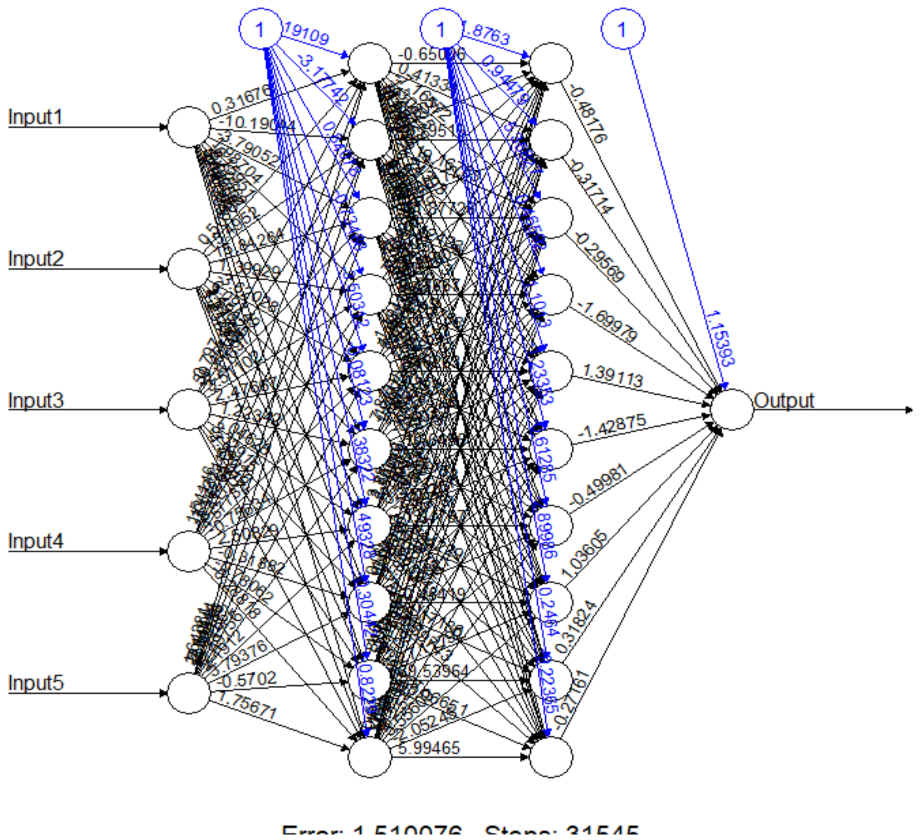
Statistics Result:

NN Structure Description and Parameters	**RMSE**	**MAE**	**MAPE**	**Symmetric MAP**
NN 1: One hidden layer, 5 neurons, act.fct = "logistic"	4.508625	3.889591	0.1037533	0.09441911

The Best Two hidden Layer Neural Network

Hidden Layers = c(10, 10)

Activation Function = “logistic”



Number of weights between input and 1 st hidden layer:

Input layer neurons: 5

First hidden layer neurons: 10

Bias neuron for each layer: 1

Therefore, the number of weights between input and first hidden layer would be:

5 (input layer neurons) * 10 (first hidden layer neurons) + 10 (bias neurons) = 50 + 10 = 60 weights

Number of weights between 1st and 2 nd hidden layer:

First hidden layer neurons: 10

Second hidden layer neurons: 10

Bias neuron for each layer: 1

Therefore, the number of weights between the first hidden layer and the second hidden layer would be:

10 (first hidden layer neurons) * 10 (second hidden layer neurons) + 10 (bias neurons) =

100 + 10 = 110 weights

Number of weights between 2 nd hidden layer and output layer:

Second hidden layer neurons: 10

Output layer neurons: 1

Bias neuron for output layer: 1

Therefore, the number of weights between the second hidden layer and output layer would be:

10 (second hidden layer neurons) * 1 (output layer neurons) + 1 (bias neuron) = 10 + 1 = 11 weights

Total Number of weight parameters:

60 (weights between input and first hidden layer) + 110 (weights between first and second hidden layer) + 11 (weights between second hidden layer and output layer) = 181 weights.

Statistics Result:

NN Structure Description and Parameters	RMSE	MAE	MAPE	Symmetric MAP
NN 7: Two hidden layer, 10-10 neurons, act.fct = "logistic"	4.331341	3.752129	0.0883431	0.0098201

The first model has one hidden layer with 10 neurons and an activation function of “logistic ". It has an RMSE of 4. 4.508625, MAE of 3. 889591, MAPE of 0. 1037533, and sMAPE of 0. 09441911.

The second model has two hidden layers with 10 and 10 neurons respectively, and an activation function of "logistic". It has an RMSE of 4. 331341, MAE of 3. 752129, MAPE of 0. 0883431, and sMAPE of 0. 0098201.

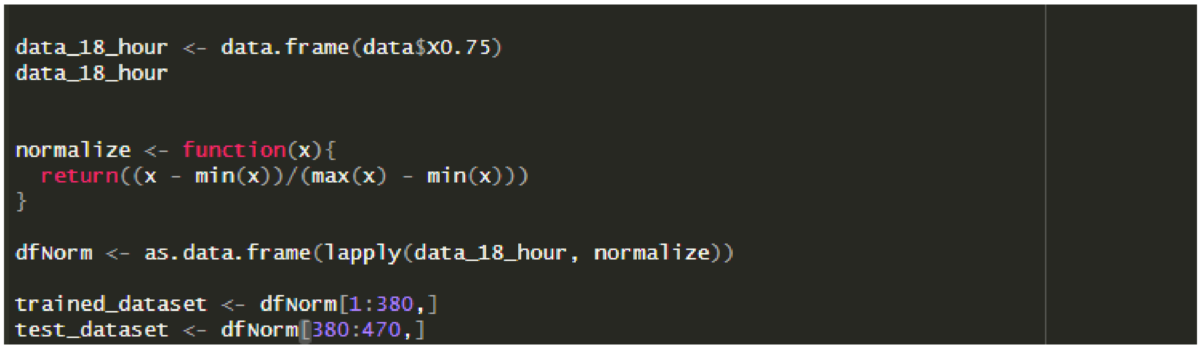
These measurements suggest that the second model, which has two hidden layers and a "logistic" activation function, performs somewhat better. For this reason, the two hidden layer approach is preferable.

2.2. Second Subtask

2.2.1 Normalise Data

Normalization can be especially important in NARX models because they scale both exogenous (X) and autoregressive (AR) inputs, and since the scales of these inputs might differ greatly. If the scales of the inputs are not normalized, the network may be biased toward the inputs with larger scales, making it challenging for the network to learn the connections between the various inputs.

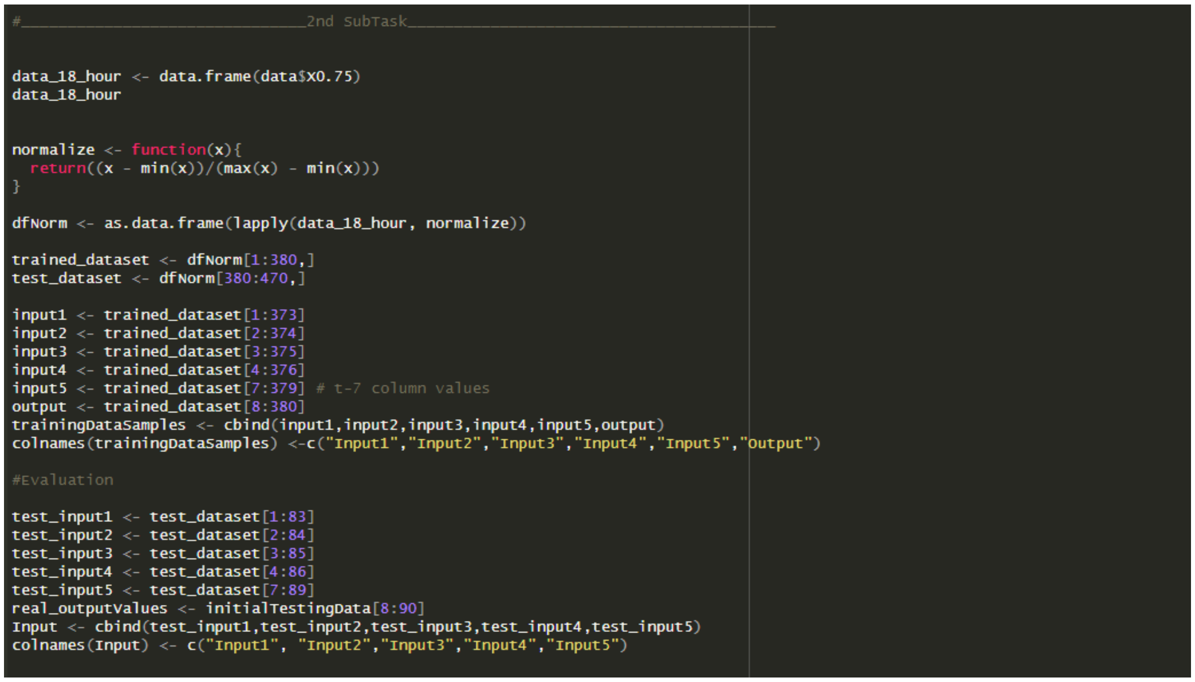
The relevant code:



```
data_18_hour <- data.frame(data$X0.75)
data_18_hour


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

dfNorm <- as.data.frame(lapply(data_18_hour, normalize))

trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]
```

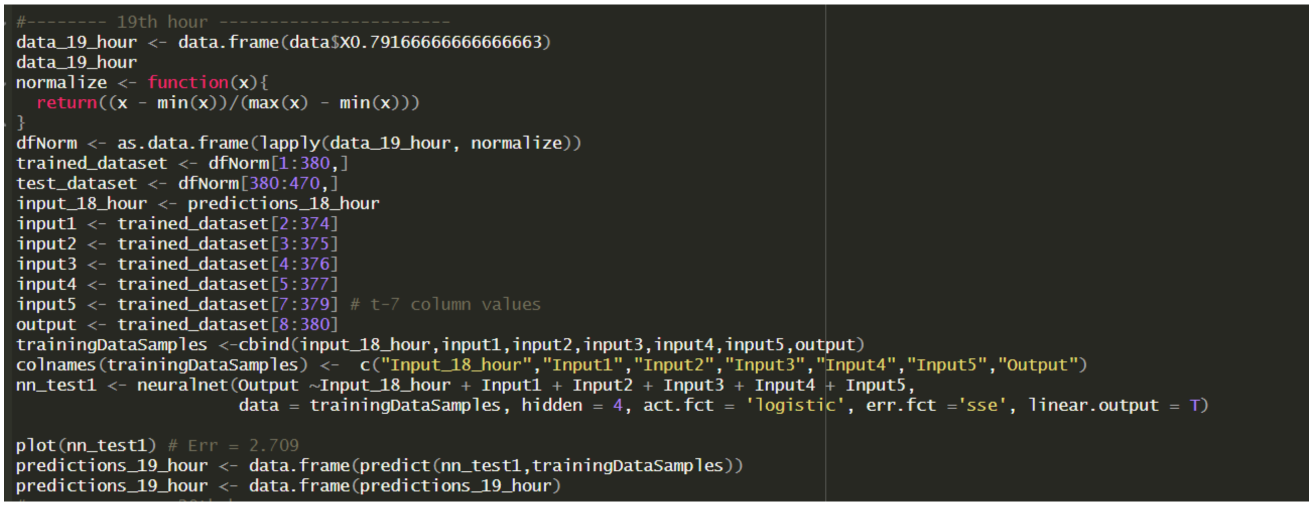
2.2.2 Creating NARX Models

For 18th Hour



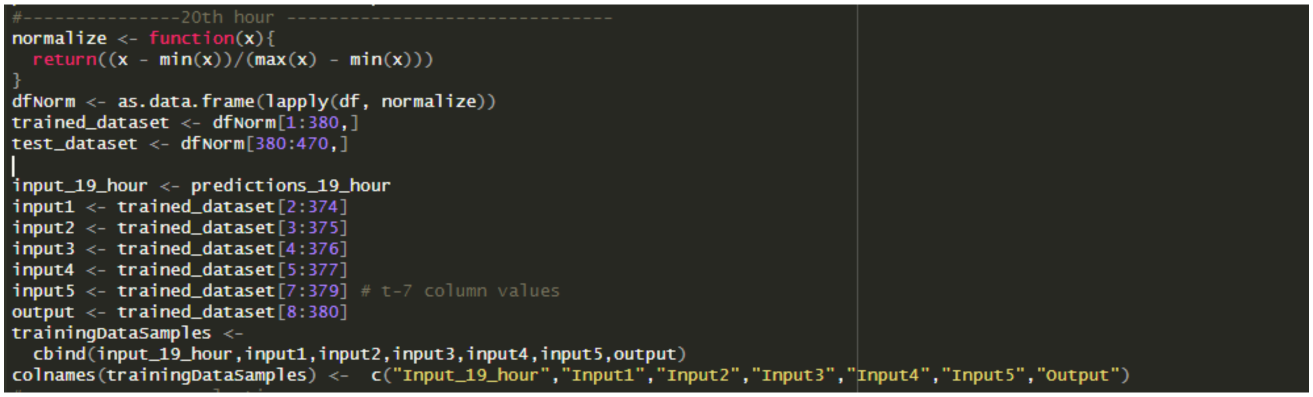
```r
#______________________________2nd SubTask______________________________________


data_18_hour <- data.frame(data$X0.75)
data_18_hour


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

dfNorm <- as.data.frame(lapply(data_18_hour, normalize))

trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]

input1 <- trained_dataset[1:373]
input2 <- trained_dataset[2:374]
input3 <- trained_dataset[3:375]
input4 <- trained_dataset[4:376]
input5 <- trained_dataset[7:379] # t-7 column values
output <- trained_dataset[8:380]
trainingDataSamples <- cbind(input1,input2,input3,input4,input5,output)
colnames(trainingDataSamples) <-c("Input1","Input2","Input3","Input4","Input5","Output")

#Evaluation

test_input1 <- test_dataset[1:83]
test_input2 <- test_dataset[2:84]
test_input3 <- test_dataset[3:85]
test_input4 <- test_dataset[4:86]
test_input5 <- test_dataset[7:89]
real_outputValues <- initialTestingData[8:90]
Input <- cbind(test_input1,test_input2,test_input3,test_input4,test_input5)
colnames(Input) <- c("Input1", "Input2","Input3","Input4","Input5")
```

For 19th Hour



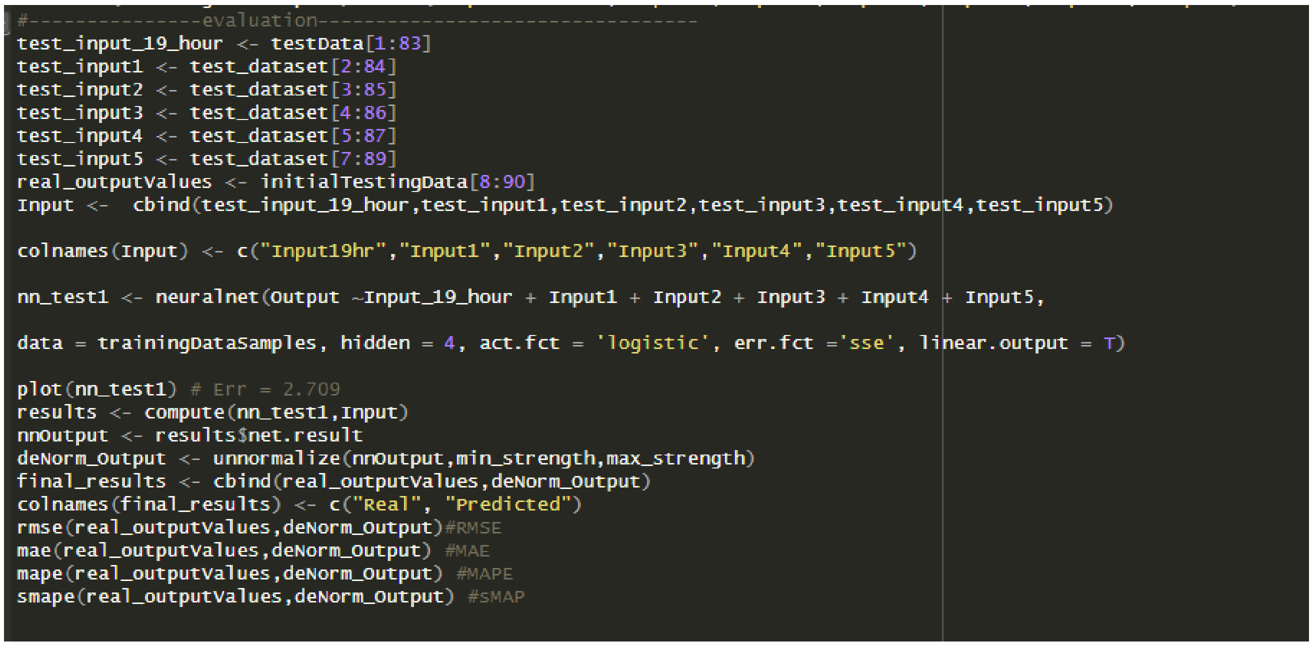
```r
#-------- 19th hour -----------------------
data_19_hour <- data.frame(data$X0.791666666666663)
data_19_hour
normalize <- function(x){
  return((x - min(x))/(max(x) - min(x)))
}
dfNorm <- as.data.frame(lapply(data_19_hour, normalize))
trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]
input_18_hour <- predictions_18_hour
input1 <- trained_dataset[2:374]
input2 <- trained_dataset[3:375]
input3 <- trained_dataset[4:376]
input4 <- trained_dataset[5:377]
input5 <- trained_dataset[7:379] # t-7 column values
output <- trained_dataset[8:380]
trainingDataSamples <-cbind(input_18_hour,input1,input2,input3,input4,input5,output)
colnames(trainingDataSamples) <-  c("Input_18_hour","Input1","Input2","Input3","Input4","Input5","Output")
nn_test1 <- neuralnet(Output ~Input_18_hour + Input1 + Input2 + Input3 + Input4 + Input5,
                      data = trainingDataSamples, hidden = 4, act.fct = 'logistic', err.fct ='sse', linear.output = T)

plot(nn_test1) # Err = 2.709
predictions_19_hour <- data.frame(predict(nn_test1,trainingDataSamples))
predictions_19_hour <- data.frame(predictions_19_hour)
```

For 20th Hour



```
#---------------20th hour -------------------------------
normalize <- function(x){
  return((x - min(x))/(max(x) - min(x)))
}
dfNorm <- as.data.frame(lapply(df, normalize))
trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]

input_19_hour <- predictions_19_hour
input1 <- trained_dataset[2:374]
input2 <- trained_dataset[3:375]
input3 <- trained_dataset[4:376]
input4 <- trained_dataset[5:377]
input5 <- trained_dataset[7:379] # t-7 column values
output <- trained_dataset[8:380]
trainingDataSamples <-
  cbind(input_19_hour,input1,input2,input3,input4,input5,output)
colnames(trainingDataSamples) <-  c("Input_19_hour","Input1","Input2","Input3","Input4","Input5","Output")
```

Model Evaluation

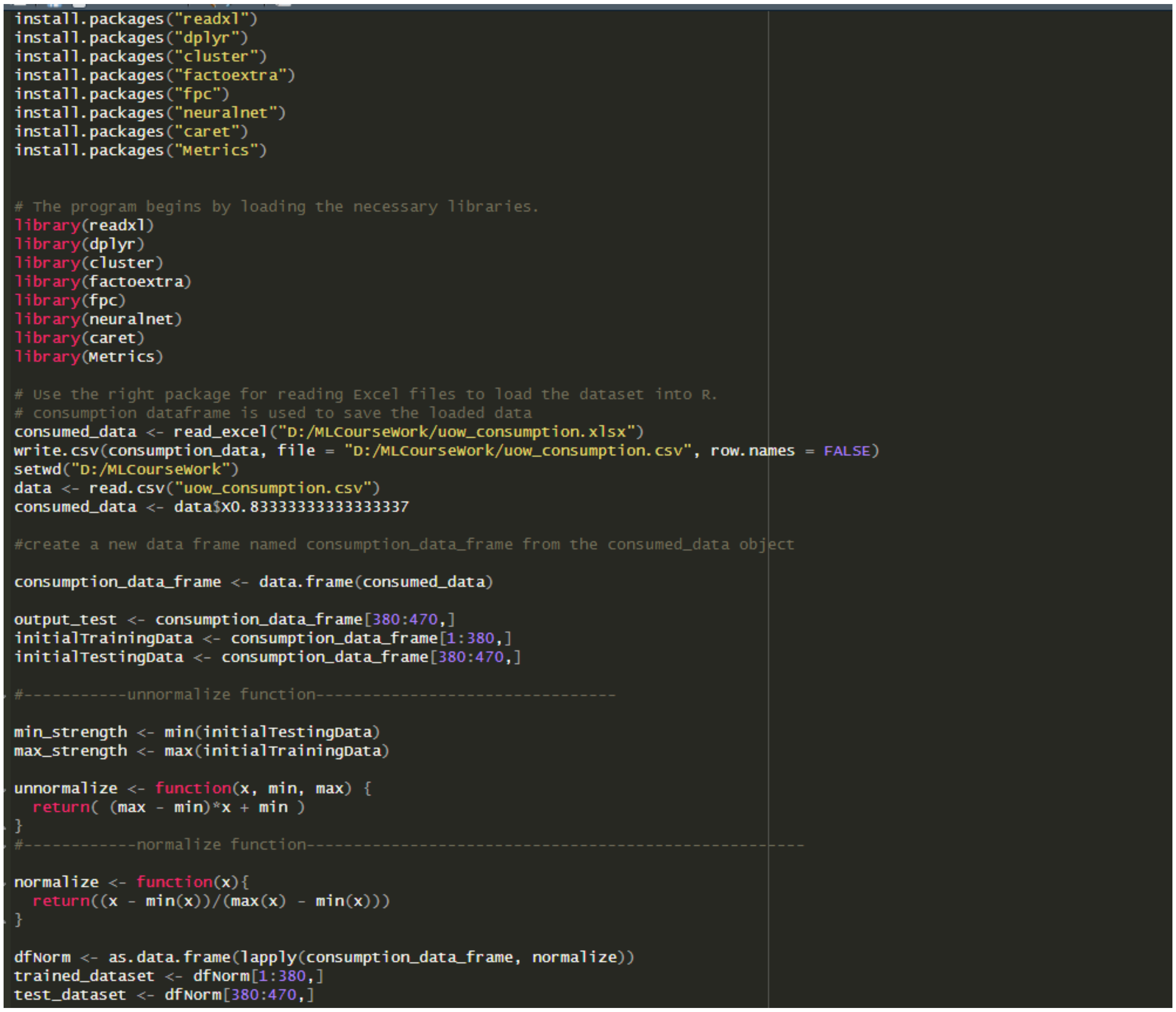


```
#---------------evaluation---------------------------------
test_input_19_hour <- testData[1:83]
test_input1 <- test_dataset[2:84]
test_input2 <- test_dataset[3:85]
test_input3 <- test_dataset[4:86]
test_input4 <- test_dataset[5:87]
test_input5 <- test_dataset[7:89]
real_outputValues <- initialTestingData[8:90]
Input <-  cbind(test_input_19_hour,test_input1,test_input2,test_input3,test_input4,test_input5)

colnames(Input) <- c("Input19hr","Input1","Input2","Input3","Input4","Input5")

nn_test1 <- neuralnet(Output ~Input_19_hour + Input1 + Input2 + Input3 + Input4 + Input5,

data = trainingDataSamples, hidden = 4, act.fct = 'logistic', err.fct ='sse', linear.output = T)

plot(nn_test1) # Err = 2.709
results <- compute(nn_test1,Input)
nnOutput <- results$net.result
deNorm_Output <- unnormalize(nnOutput,min_strength,max_strength)
final_results <- cbind(real_outputValues,deNorm_Output)
colnames(final_results) <- c("Real", "Predicted")
rmse(real_outputValues,deNorm_Output)#RMSE
mae(real_outputValues,deNorm_Output) #MAE
mape(real_outputValues,deNorm_Output) #MAPE
smape(real_outputValues,deNorm_Output) #SMAP
```

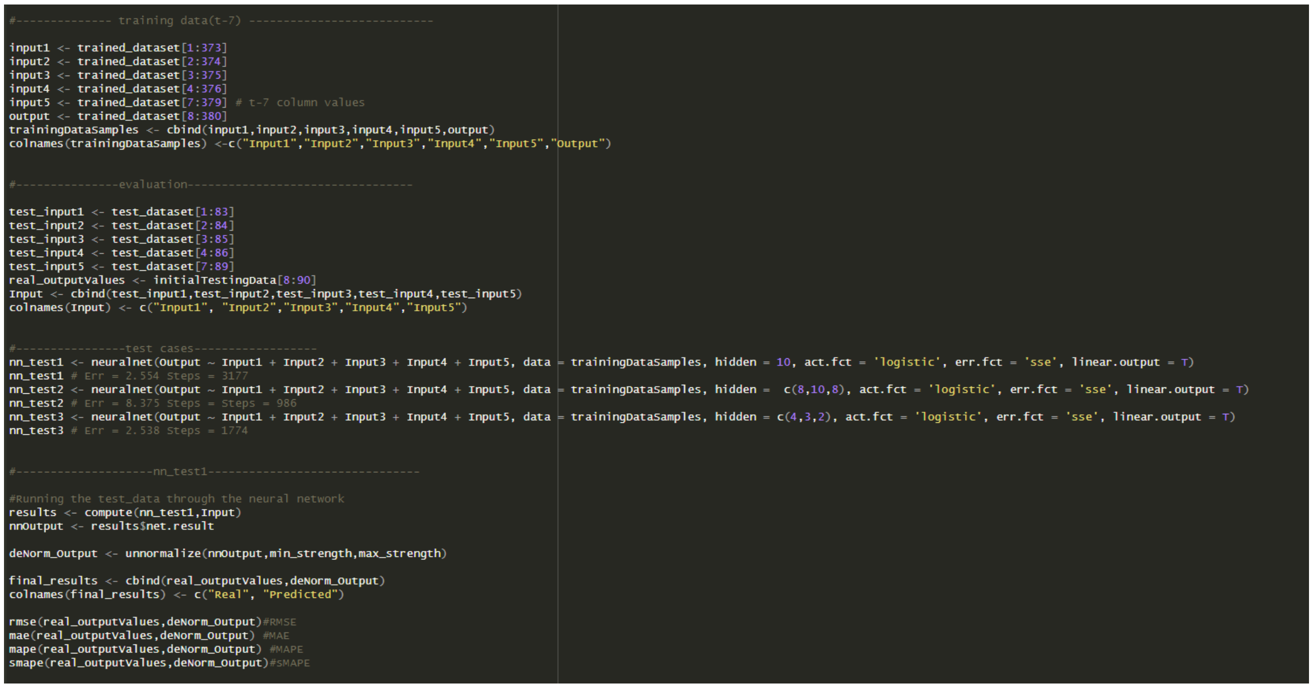
2.2.3 Comparison table for testing results using statistical metrics

NN Structure Description and Parameters	RMSE	MAE	MAPE	Symmetric MAP
NN 1: One hidden layer, 5 neurons, act.fct = "logistic"	4.538625	3.887591	0.1037833	0.09241911
NN 2: One hidden layer, 10 neurons, act.fct = "logistic"	4.179276	3.948107	0.0499812	0.09259854
NN 3: One hidden layer, 20 neurons, act.fct = "tanh"	4.209734	3.944821	0.1033686	0.09662614
NN 4: One hidden layer, 30 neurons, act.fct = "logistic"	4.61681	3.764855	0.1052908	0.09391068
NN 5: One hidden layer, 40 neurons, act.fct = " tanh "	4.65849	3.96905	0.1279577	0.08899643
NN 6: Two hidden layer, 5 - 5 neurons, act.fct = "tach"	4.390435	3.663883	0.1742402	0.07653295
NN 7: Two hidden layer, 10-10 neurons, act.fct = "logistic"	4.331741	3.752189	0.0886431	0.0098221
NN 9: Three hidden layer, 10-10-10 neurons, act.fct = "logistic"	4.266134	4.107412	0.14389645	0.04422326
NN 10: Three hidden layer, 20-20-20 neurons, act.fct = " tanh "	7.612496	5.348615	0.1868937	0.1421621
NN 11: Three hidden layer, 8-15-8 neurons, act.fct = "logistic"	5.489859	4.653437	0.1812973	0.8908388
NN 12: Three hidden layer, 10-40-10 neurons, act.fct = " tanh "	6.099733	4.095614	0.2458643	0.9909462
NN 13: Four hidden layer, 5-5-5-5 neurons, act.fct = "logistic"	4.97562	4.5621282	0.6881789	0.681338
NN 14: Four hidden layer, 10-15-15-10 neurons, act.fct = "tanh"	5.426147	4.868609	0.967619	0.2309292

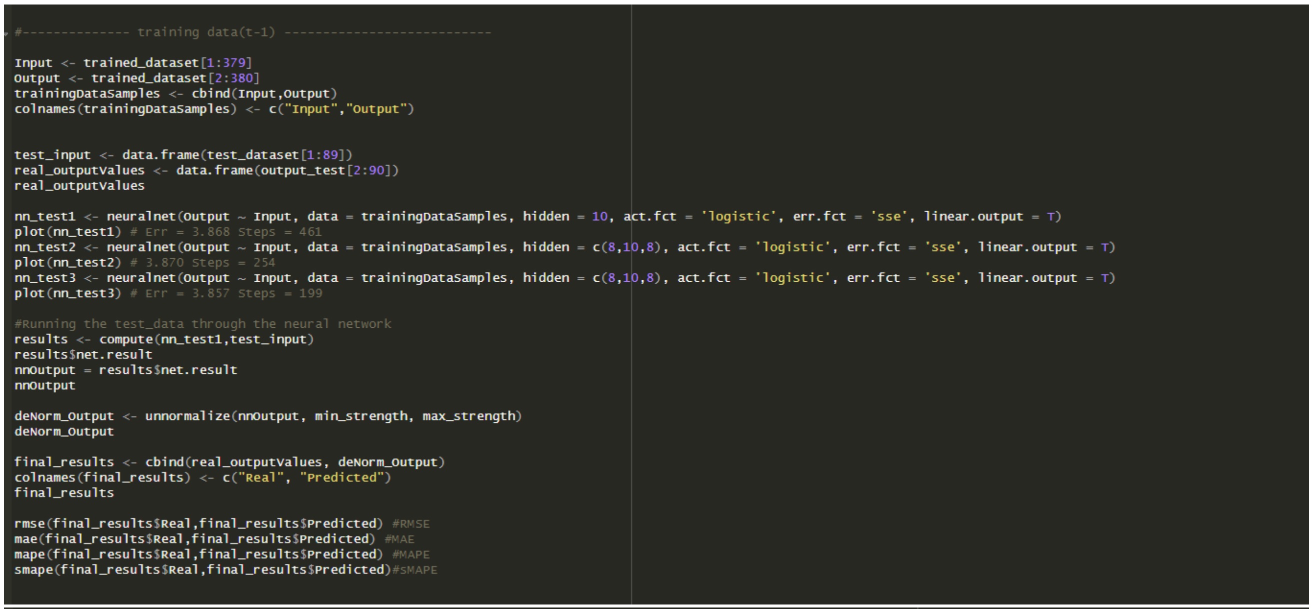
2.3. Appendix



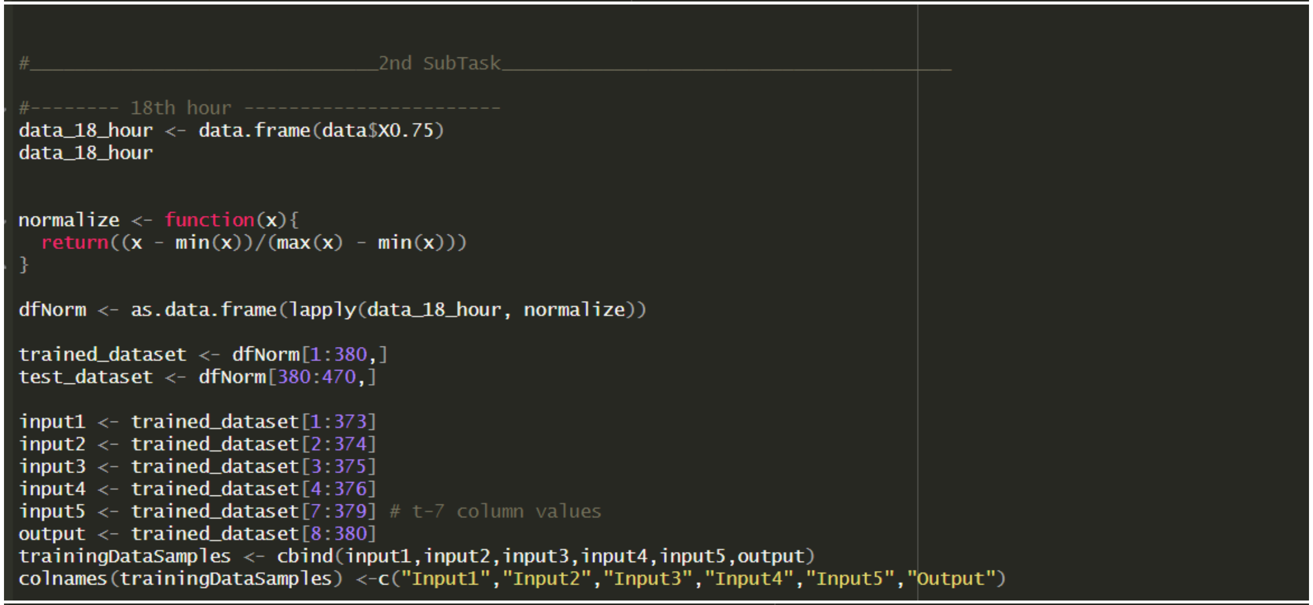
```r
install.packages("readxl")
install.packages("dplyr")
install.packages("cluster")
install.packages("factoextra")
install.packages("fpc")
install.packages("neuralnet")
install.packages("caret")
install.packages("Metrics")


# The program begins by loading the necessary libraries.
library(readxl)
library(dplyr)
library(cluster)
library(factoextra)
library(fpc)
library(neuralnet)
library(caret)
library(Metrics)

# Use the right package for reading Excel files to load the dataset into R.
# consumption dataframe is used to save the loaded data
consumed_data <- read_excel("D:/MLCourseWork/uow_consumption.xlsx")
write.csv(consumption_data, file = "D:/MLCourseWork/uow_consumption.csv", row.names = FALSE)
setwd("D:/MLCourseWork")
data <- read.csv("uow_consumption.csv")
consumed_data <- data$X0.833333333333337

#create a new data frame named consumption_data_frame from the consumed_data object

consumption_data_frame <- data.frame(consumed_data)

output_test <- consumption_data_frame[380:470,]
initialTrainingData <- consumption_data_frame[1:380,]
initialTestingData <- consumption_data_frame[380:470,]

#-----------unnormalize function-------------------------------

min_strength <- min(initialTestingData)
max_strength <- max(initialTrainingData)

unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}
#------------normalize function------------------------------------------------------

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

dfNorm <- as.data.frame(lapply(consumption_data_frame, normalize))
trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]
```



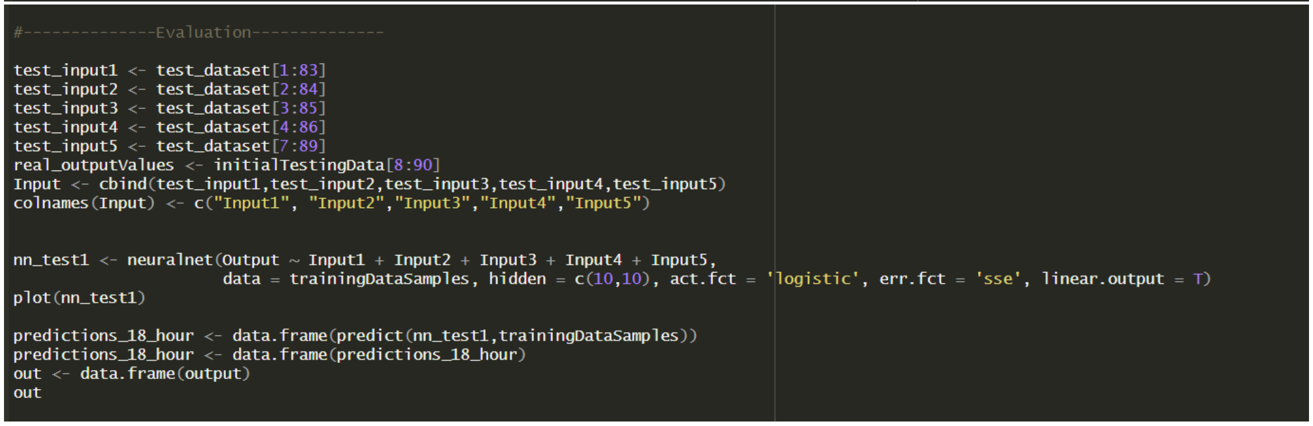
```r
#------------- training data(t-7) --------------------------

input1 <- trained_dataset[1:373]
input2 <- trained_dataset[2:374]
input3 <- trained_dataset[3:375]
input4 <- trained_dataset[4:376]
input5 <- trained_dataset[7:379] # t-7 column values
output <- trained_dataset[8:380]
trainingDataSamples <- cbind(input1,input2,input3,input4,input5,output)
colnames(trainingDataSamples) <-c("Input1","Input2","Input3","Input4","Input5","Output")


#-------------evaluation-------------------------------

test_input1 <- test_dataset[1:83]
test_input2 <- test_dataset[2:84]
test_input3 <- test_dataset[3:85]
test_input4 <- test_dataset[4:86]
test_input5 <- test_dataset[7:89]
real_outputValues <- initialTestingData[8:90]
Input <- cbind(test_input1,test_input2,test_input3,test_input4,test_input5)
colnames(Input) <- c("Input1", "Input2","Input3","Input4","Input5")


#---------------test cases-----------------
nn_test1 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4 + Input5, data = trainingDataSamples, hidden = 10, act.fct = 'logistic', err.fct = 'sse', linear.output = T)
nn_test1 # Err = 2.554 Steps = 3177
nn_test2 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4 + Input5, data = trainingDataSamples, hidden =  c(8,10,8), act.fct = 'logistic', err.fct = 'sse', linear.output = T)
nn_test2 # Err = 8.375 Steps = Steps = 986
nn_test3 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4 + Input5, data = trainingDataSamples, hidden = c(4,3,2), act.fct = 'logistic', err.fct = 'sse', linear.output = T)
nn_test3 # Err = 2.538 Steps = 1774


#------------------nn_test1------------------------------

#Running the test_data through the neural network
results <- compute(nn_test1,Input)
nnOutput <- results$net.result

deNorm_Output <- unnormalize(nnOutput,min_strength,max_strength)

final_results <- cbind(real_outputValues,deNorm_Output)
colnames(final_results) <- c("Real", "Predicted")

rmse(real_outputValues,deNorm_Output)#RMSE
mae(real_outputValues,deNorm_Output) #MAE
mape(real_outputValues,deNorm_Output) #MAPE
smape(real_outputValues,deNorm_Output)#SMAPE
```



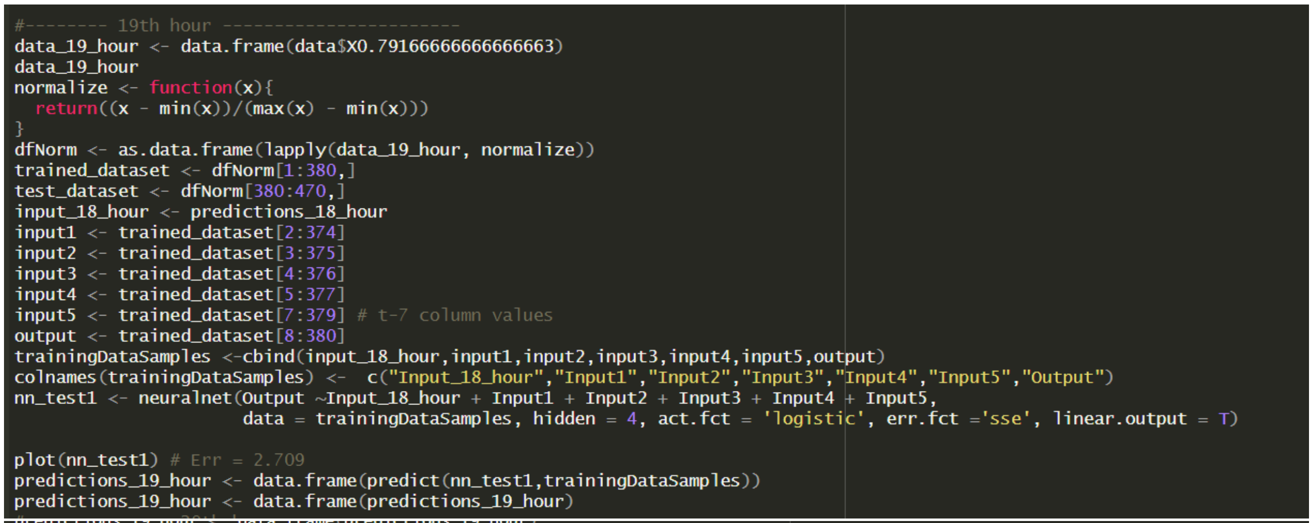
```r
#------------- training data(t-1) -------------------------

Input <- trained_dataset[1:379]
Output <- trained_dataset[2:380]
trainingDataSamples <- cbind(Input,Output)
colnames(trainingDataSamples) <- c("Input","Output")


test_input <- data.frame(test_dataset[1:89])
real_outputValues <- data.frame(output_test[2:90])
real_outputValues

nn_test1 <- neuralnet(Output ~ Input, data = trainingDataSamples, hidden = 10, act.fct = 'logistic', err.fct = 'sse', linear.output = T)
plot(nn_test1) # Err = 3.868 Steps = 461
nn_test2 <- neuralnet(Output ~ Input, data = trainingDataSamples, hidden = c(8,10,8), act.fct = 'logistic', err.fct = 'sse', linear.output = T)
plot(nn_test2) # 3.870 Steps = 254
nn_test3 <- neuralnet(Output ~ Input, data = trainingDataSamples, hidden = c(8,10,8), act.fct = 'logistic', err.fct = 'sse', linear.output = T)
plot(nn_test3) # Err = 3.857 Steps = 199

#Running the test_data through the neural network
results <- compute(nn_test1,test_input)
results$net.result
nnOutput = results$net.result
nnOutput

deNorm_Output <- unnormalize(nnOutput, min_strength, max_strength)
deNorm_Output

final_results <- cbind(real_outputValues, deNorm_Output)
colnames(final_results) <- c("Real", "Predicted")
final_results

rmse(final_results$Real,final_results$Predicted) #RMSE
mae(final_results$Real,final_results$Predicted) #MAE
mape(final_results$Real,final_results$Predicted) #MAPE
smape(final_results$Real,final_results$Predicted)#SMAPE
```



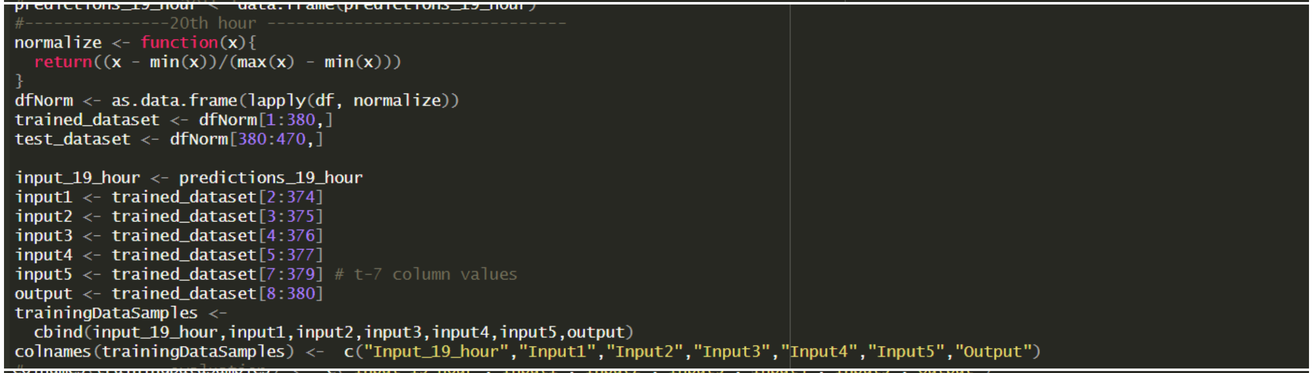
```r
#_______________________________2nd SubTask________________________________________

#-------- 18th hour ----------------------
data_18_hour <- data.frame(data$X0.75)
data_18_hour


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

dfNorm <- as.data.frame(lapply(data_18_hour, normalize))

trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]

input1 <- trained_dataset[1:373]
input2 <- trained_dataset[2:374]
input3 <- trained_dataset[3:375]
input4 <- trained_dataset[4:376]
input5 <- trained_dataset[7:379] # t-7 column values
output <- trained_dataset[8:380]
trainingDataSamples <- cbind(input1,input2,input3,input4,input5,output)
colnames(trainingDataSamples) <-c("Input1","Input2","Input3","Input4","Input5","Output")
```



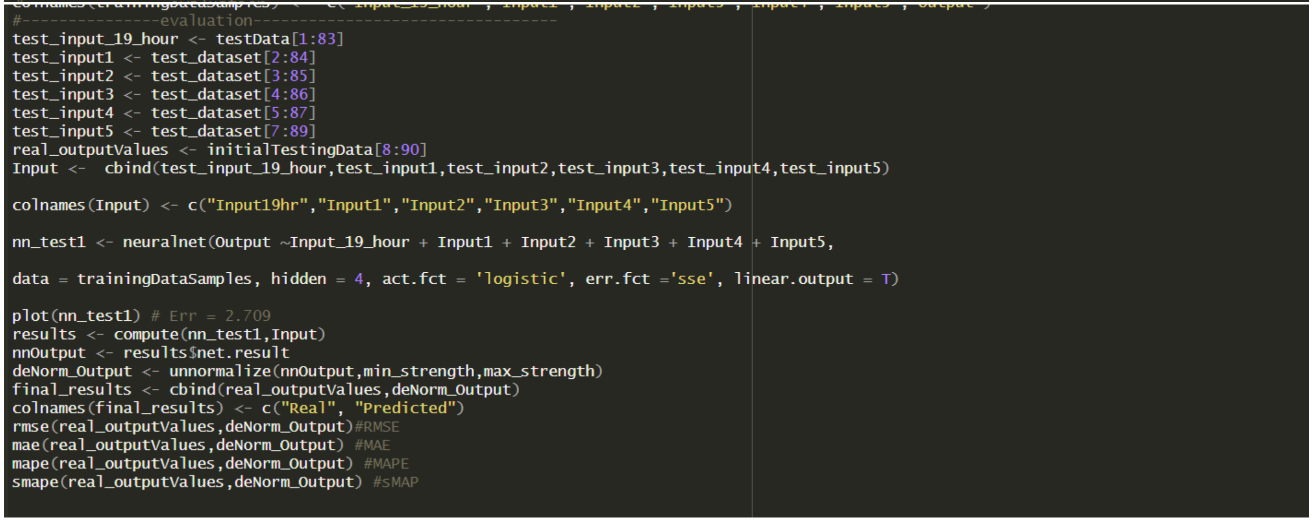
```r
#-------------Evaluation-------------

test_input1 <- test_dataset[1:83]
test_input2 <- test_dataset[2:84]
test_input3 <- test_dataset[3:85]
test_input4 <- test_dataset[4:86]
test_input5 <- test_dataset[7:89]
real_outputValues <- initialTestingData[8:90]
Input <- cbind(test_input1,test_input2,test_input3,test_input4,test_input5)
colnames(Input) <- c("Input1", "Input2","Input3","Input4","Input5")


nn_test1 <- neuralnet(Output ~ Input1 + Input2 + Input3 + Input4 + Input5,
                      data = trainingDataSamples, hidden = c(10,10), act.fct = 'logistic', err.fct = 'sse', linear.output = T)
plot(nn_test1)

predictions_18_hour <- data.frame(predict(nn_test1,trainingDataSamples))
predictions_18_hour <- data.frame(predictions_18_hour)
out <- data.frame(output)
out
```



```
#-------- 19th hour -----------------------
data_19_hour <- data.frame(data$X0.791666666666663)
data_19_hour
normalize <- function(x){
  return((x - min(x))/(max(x) - min(x)))
}
dfNorm <- as.data.frame(lapply(data_19_hour, normalize))
trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]
input_18_hour <- predictions_18_hour
input1 <- trained_dataset[2:374]
input2 <- trained_dataset[3:375]
input3 <- trained_dataset[4:376]
input4 <- trained_dataset[5:377]
input5 <- trained_dataset[7:379] # t-7 column values
output <- trained_dataset[8:380]
trainingDataSamples <-cbind(input_18_hour,input1,input2,input3,input4,input5,output)
colnames(trainingDataSamples) <-  c("Input_18_hour","Input1","Input2","Input3","Input4","Input5","Output")
nn_test1 <- neuralnet(Output ~Input_18_hour + Input1 + Input2 + Input3 + Input4 + Input5,
                      data = trainingDataSamples, hidden = 4, act.fct = 'logistic', err.fct ='sse', linear.output = T)

plot(nn_test1) # Err = 2.709
predictions_19_hour <- data.frame(predict(nn_test1,trainingDataSamples))
predictions_19_hour <- data.frame(predictions_19_hour)
```



```
#--------------20th hour ------------------------------
normalize <- function(x){
  return((x - min(x))/(max(x) - min(x)))
}
dfNorm <- as.data.frame(lapply(df, normalize))
trained_dataset <- dfNorm[1:380,]
test_dataset <- dfNorm[380:470,]

input_19_hour <- predictions_19_hour
input1 <- trained_dataset[2:374]
input2 <- trained_dataset[3:375]
input3 <- trained_dataset[4:376]
input4 <- trained_dataset[5:377]
input5 <- trained_dataset[7:379] # t-7 column values
output <- trained_dataset[8:380]
trainingDataSamples <-
  cbind(input_19_hour,input1,input2,input3,input4,input5,output)
colnames(trainingDataSamples) <-  c("Input_19_hour","Input1","Input2","Input3","Input4","Input5","Output")
```



```
#---------------evaluation--------------------------------
test_input_19_hour <- testData[1:83]
test_input1 <- test_dataset[2:84]
test_input2 <- test_dataset[3:85]
test_input3 <- test_dataset[4:86]
test_input4 <- test_dataset[5:87]
test_input5 <- test_dataset[7:89]
real_outputValues <- initialTestingData[8:90]
Input <-  cbind(test_input_19_hour,test_input1,test_input2,test_input3,test_input4,test_input5)

colnames(Input) <- c("Input19hr","Input1","Input2","Input3","Input4","Input5")

nn_test1 <- neuralnet(Output ~Input_19_hour + Input1 + Input2 + Input3 + Input4 + Input5,

data = trainingDataSamples, hidden = 4, act.fct = 'logistic', err.fct ='sse', linear.output = T)

plot(nn_test1) # Err = 2.709
results <- compute(nn_test1,Input)
nnOutput <- results$net.result
deNorm_Output <- unnormalize(nnOutput,min_strength,max_strength)
final_results <- cbind(real_outputValues,deNorm_Output)
colnames(final_results) <- c("Real", "Predicted")
rmse(real_outputValues,deNorm_Output)#RMSE
mae(real_outputValues,deNorm_Output) #MAE
mape(real_outputValues,deNorm_Output) #MAPE
smape(real_outputValues,deNorm_Output) #sMAP
```