Machine Learning and Data Mining

5DATA001C.2

Course work Report

Module Leader's Name- Mr. Nipuna Senanayaka

Name: K.K. Isuru Kuruppu

UOW ID: w187045

IIT ID: 20210172

Partitioning Clustering Part

Objectives/Deliverables (partitioning clustering)

1st Subtask Objectives:

(a)

Scaling: - Scaling is the process of converting a dataset's numerical features to a standard scale. Due of the sensitivity of many machine learning algorithms to the magnitude of the input features, this is done. Results may be skewed if the features are not weighted equally; when this happens, some features will predominate over others. This bias can be avoided and the machine learning algorithm's performance can be enhanced by scaling the features. Scaling can also hasten the optimization process during training, hastening convergence and improving outcomes.

Outliers Detection: - Data points known as outliers differ significantly from other data points in the dataset. They may appear for a number of reasons, including measurement errors, data corruption, or just the basic nature of the data itself. Outliers can significantly affect how well machine learning algorithm's function since they can distort the findings and produce false models. In order to improve data collecting and processing procedures, outliers can be used to discover data quality problems and abnormalities.

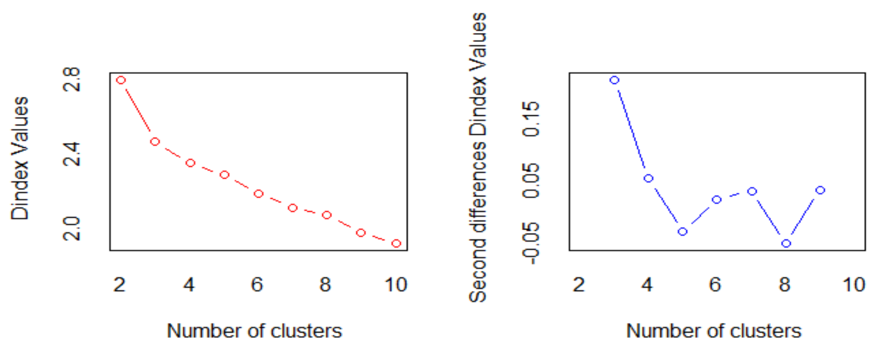
Removal: - To increase the model's accuracy and resilience, outliers can be eliminated. However, it's crucial to remember that eliminating too many outliers can also result in the loss of important data, so a careful balance must be struck. Removing outliers can aid in discovering and fixing problems with data collection and processing, which can assist to improve data quality and reliability. Additionally, eliminating outliers can aid in enhancing the model's interpretability by assisting in the discovery of data patterns and relationships that may have been hidden by the presence of outliers.

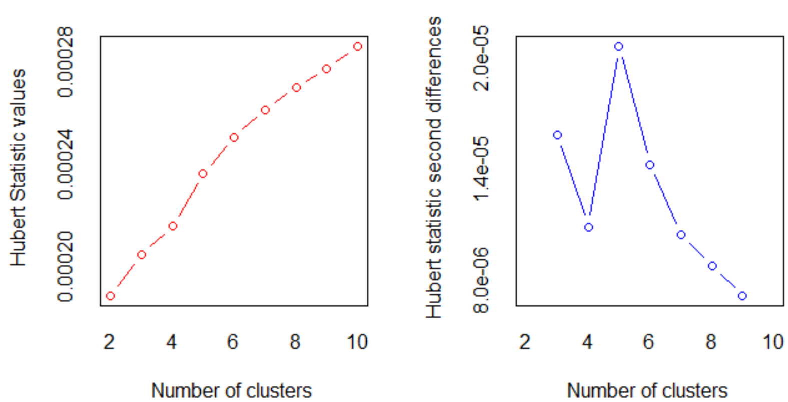
(b)

1. NbClust methods.

The R package NbClust is used to determine the ideal number of clusters to include in a dataset. It offers a number of indices and techniques, including the Silhouette coefficient, Calinski-Harabasz index, Dunn index, and Gap statistic, to assess clustering solutions. In order to facilitate the selecting process, it also offers graphical representations of the indexes.

```r
#NbClust method
install.packages("NbClust")
library(NbClust)
nb <- NbClust(vehicles_cleaned, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
print(nb)
```





```
> nb <- NbClust(vehicles_cleaned, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

*******************************************************************
* Among all indices:
* 7 proposed 2 as the best number of clusters
* 14 proposed 3 as the best number of clusters
* 2 proposed 9 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


*******************************************************************
> print(nb)
$All.index
       KL       CH Hartigan     CCC    Scott      Marriot    TrCovW   TraceW Friedman  Rubin Cindex     DB Silhouette   Duda
2  3.5544 613.4316 221.0492  1.6559 1715.763 9.691366e+32 416530.21 6433.965 2473.208 1.8165 0.4007 1.0353     0.3946 0.7425
3  3.3396 506.4872  83.6306  3.8244 2621.103 6.583886e+32 198623.19 4975.349 2578.279 2.3491 0.3722 1.2997     0.3016 1.9629
4  1.7836 402.5012  53.2393  2.6011 3164.673 5.702936e+32 154577.93 4478.007 2763.442 2.6100 0.3885 1.5349     0.2392 1.6375
5  0.6373 336.1101  73.0006  1.4613 3991.802 2.983757e+32 131393.69 4181.938 2904.615 2.7947 0.4448 1.4713     0.2317 1.0650
6  1.5649 309.2130  51.4709  2.3405 4719.828 1.640218e+32 110299.99 3811.448 3079.918 3.0664 0.4484 1.4725     0.2168 1.7268
7  2.1001 283.5613  29.9185  2.4751 5331.845 9.936042e+31  93033.32 3566.675 3183.041 3.2768 0.4450 1.6214     0.2051 1.5176
8  0.3629 256.6921  65.0312 -1.7152 5446.125 1.115700e+32  85111.06 3429.677 3132.448 3.4077 0.4692 1.6756     0.1837 0.8929
9  1.6831 251.9244  42.9683 -1.1747 5954.440 7.208491e+31  70740.98 3155.351 3332.948 3.7040 0.4487 1.6319     0.1821 3.9550
10 1.5085 241.2646  31.2382 -5.1568 6447.939 4.632999e+31  64901.20 2983.724 3486.566 3.9170 0.4160 1.5749     0.1837 2.1828
    Pseudot2   Beale Ratkowsky      Ball Ptbiserial   Frey McClain   Dunn Hubert SDindex Dindex   SDbw
2   180.6750  4.3229    0.4014 3216.9825     0.6584 1.2881  0.4844 0.1214  2e-04  1.0509 2.7989 0.6204
3  -206.0288 -6.0999    0.4066 1658.4496     0.5939 1.2143  1.1299 0.0939  2e-04  1.1138 2.4683 0.4848
4  -151.8335 -4.8412    0.3663 1119.5017     0.5386 0.3197  1.6607 0.1035  2e-04  1.2477 2.3505 0.4598
5   -15.6920 -0.7588    0.3380  836.3876     0.5389 0.5574  1.7803 0.0931  2e-04  1.3146 2.2861 0.4423
6  -114.0613 -5.2272    0.3165  635.2414     0.5197 0.7410  2.1426 0.1001  3e-04  1.2635 2.1901 0.4259
7   -56.6137 -4.2181    0.3021  509.5249     0.4863 1.0920  2.6742 0.1037  3e-04  1.3882 2.1145 0.4093
8    19.5444  1.4886    0.2866  428.7097     0.4624 0.5733  3.0586 0.1110  3e-04  1.5407 2.0721 0.3885
9  -119.5448 -9.2029    0.2750  350.5946     0.4339 0.3441  3.7096 0.1216  3e-04  1.4777 1.9788 0.3596
10  -80.7378 -6.6702    0.2643  298.3724     0.4219 0.2591  4.0945 0.1164  3e-04  1.5062 1.9207 0.3518

$All.CriticalValues
   CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
2          0.9180            46.5283       0.0000
3          0.8951            49.2000       1.0000
4          0.8953            45.6108       1.0000
5          0.8904            31.6453       1.0000
6          0.8861            34.8482       1.0000
7          0.8619            26.5924       1.0000
```

Console | Terminal | Background Jobs

R 4.2.2 · D:/2nd Year/ML_CourseWork/

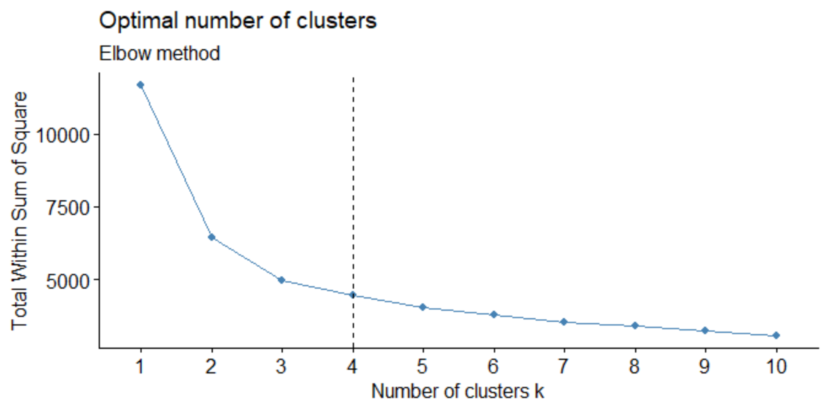

```
3  3.3396 506.4872  83.6306  3.8244 2621.103 6.583886e+32 198623.19 4975.349 2578.279 2.3491 0.3722 1.2997     0.3016 1.9629
4  1.7836 402.5012  53.2393  2.6011 3164.673 5.702936e+32 154577.93 4478.007 2763.442 2.6100 0.3885 1.5349     0.2392 1.6375
5  0.6373 336.1101  73.0006  1.4613 3991.802 2.983757e+32 131393.69 4181.938 2904.615 2.7947 0.4448 1.4713     0.2317 1.0650
6  1.5649 309.2130  51.4709  2.3405 4719.828 1.640218e+32 110299.99 3811.448 3079.918 3.0664 0.4484 1.4725     0.2168 1.7268
7  2.1001 283.5613  29.9185  2.4751 5331.845 9.936042e+31  93033.32 3566.675 3183.041 3.2768 0.4450 1.6214     0.2051 1.5176
8  0.3629 256.6921  65.0312 -1.7152 5446.125 1.115700e+32  85111.06 3429.677 3132.448 3.4077 0.4692 1.6756     0.1837 0.8929
9  1.6831 251.9244  42.9683 -1.1747 5954.440 7.208491e+31  70740.98 3155.351 3332.948 3.7040 0.4487 1.6319     0.1821 3.9550
10 1.5085 241.2646  31.2382 -5.1568 6447.939 4.632999e+31  64901.20 2983.724 3486.566 3.9170 0.4160 1.5749     0.1837 2.1828
    Pseudot2   Beale Ratkowsky      Ball Ptbiserial   Frey McClain   Dunn Hubert SDindex Dindex   SDbw
2   180.6750  4.3229    0.4014 3216.9825     0.6584 1.2881  0.4844 0.1214  2e-04  1.0509 2.7989 0.6204
3  -206.0288 -6.0999    0.4066 1658.4496     0.5939 1.2143  1.1299 0.0939  2e-04  1.1138 2.4683 0.4848
4  -151.8335 -4.8412    0.3663 1119.5017     0.5386 0.3197  1.6607 0.1035  2e-04  1.2477 2.3505 0.4598
5   -15.6920 -0.7588    0.3380  836.3876     0.5389 0.5574  1.7803 0.0931  2e-04  1.3146 2.2861 0.4423
6  -114.0613 -5.2272    0.3165  635.2414     0.5197 0.7410  2.1426 0.1001  3e-04  1.2635 2.1901 0.4259
7   -56.6137 -4.2181    0.3021  509.5249     0.4863 1.0920  2.6742 0.1037  3e-04  1.3882 2.1145 0.4093
8    19.5444  1.4886    0.2866  428.7097     0.4624 0.5733  3.0586 0.1110  3e-04  1.5407 2.0721 0.3885
9  -119.5448 -9.2029    0.2750  350.5946     0.4339 0.3441  3.7096 0.1216  3e-04  1.4777 1.9788 0.3596
10  -80.7378 -6.6702    0.2643  298.3724     0.4219 0.2591  4.0945 0.1164  3e-04  1.5062 1.9207 0.3518

$All.CriticalValues
   CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
2          0.9180            46.5283       0.0000
3          0.8951            49.2000       1.0000
4          0.8953            45.6108       1.0000
5          0.8904            31.6453       1.0000
6          0.8861            34.8482       1.0000
7          0.8619            26.5924       1.0000
```


```
8          0.8835            21.4969       0.0839
9          0.8426            29.8813       1.0000
10         0.8400            28.3715       1.0000

$Best.nc
                    KL       CH Hartigan    CCC    Scott      Marriot TrCovW   TraceW Friedman   Rubin Cindex     DB Silhouette
Number_clusters 2.0000   2.0000   3.0000 3.0000   3.0000 3.000000e+00      3   3.0000   9.0000  3.0000 3.0000 2.0000     2.0000
Value_Index     3.5544 613.4316 137.4185 3.8244 905.3405 2.226529e+32 217907 961.2743 200.4994 -0.2717 0.3722 1.0353     0.3946
                  Duda  PseudoT2   Beale Ratkowsky     Ball PtBiserial   Frey McClain   Dunn Hubert SDindex Dindex    SDbw
Number_clusters 3.0000    3.0000  3.0000    3.0000    3.000     2.0000 3.0000  2.0000 9.0000      0  2.0000      0 10.0000
Value_Index     1.9629 -206.0288 -6.0999    0.4066 1558.533     0.6584 1.2143  0.4844 0.1216      0  1.0509      0  0.3518

$Best.partition
  [1] 3 3 2 3 3 3 3 3 3 3 3 3 3 2 1 3 2 2 1 1 3 3 2 3 1 2 2 1 3 3 3 2 3 3 1 2 1 2 1 3 2 1 1 1 1 3 1 3 2 3 2 3 3 1 2 2 1 1 1 3 1 1 3
 [64] 2 2 2 3 1 3 2 3 2 1 1 2 3 1 3 3 1 3 1 2 3 2 3 1 1 1 2 1 3 3 2 2 2 1 1 2 3 3 1 3 2 2 1 3 1 1 3 1 1 1 3 2 2 3 1 2 1 3 1 3 3 1 1
[127] 3 2 3 3 2 3 3 2 3 2 3 1 3 3 1 2 3 3 2 2 1 1 2 2 3 2 3 3 3 3 1 2 1 3 1 2 3 3 2 3 2 3 3 3 1 2 1 1 1 3 3 3 3 3 1 1 2 3 3 3 2 1 3
[190] 1 2 1 3 1 2 1 1 3 2 3 2 1 1 1 1 2 3 1 3 1 3 3 1 2 1 1 3 3 2 1 1 2 1 3 3 2 3 2 2 1 3 3 3 2 1 1 3 3 1 1 3 3 3 1 1 2 3 3 1 1 1 3
[253] 3 1 2 1 3 3 2 3 2 1 3 3 2 3 3 3 1 3 2 2 2 2 2 1 3 2 1 1 1 3 1 2 2 1 2 3 1 2 3 3 3 3 2 2 1 2 2 1 2 3 3 3 1 1 2 2 2 3 3 2 1 3 1
[316] 2 3 3 2 3 2 2 2 3 3 1 1 1 1 3 3 3 3 3 1 2 2 1 1 2 1 2 1 3 1 3 3 2 1 3 3 2 3 2 3 3 3 2 3 2 3 1 1 3 3 3 1 1 3 1 2 3 3 1 1 2 3 1
[379] 3 3 2 2 3 2 2 1 1 2 3 3 2 2 1 3 2 2 1 2 2 2 3 3 3 3 3 2 1 3 2 3 3 2 3 2 1 1 2 2 3 1 2 1 2 2 1 3 1 2 2 3 3 1 1 2 3 1 2 2 1 2 2
[442] 3 2 1 1 2 2 1 1 2 2 3 3 1 3 2 1 1 2 1 3 3 1 3 2 3 2 3 1 3 2 2 1 1 3 2 3 2 2 3 3 3 3 1 1 3 3 2 1 1 3 1 2 3 2 1 1 2 2 3 3 3 2 3
[505] 1 3 2 3 3 1 2 2 2 2 3 1 1 2 2 2 3 2 1 3 1 1 1 3 1 2 3 3 3 3 3 2 3 3 2 3 3 1 2 1 1 3 1 3 3 1 2 2 1 3 1 2 3 3 2 3 1 2 1 2 1 3 1
[568] 3 2 2 1 2 3 3 1 3 1 2 3 2 1 3 1 1 1 3 2 3 1 3 3 3 3 2 3 1 2 2 3 3 2 1 2 1 3 3 3 1 2 3 1 3 2 1 3 3 2 1 3 1 3 3 1 3 2 2 3 3 2 2
[631] 1 3 2 2 2 2 3 2 3 3 2 2 3 2 3 2 3 2 3 1 2 2 2 3 1 1 2 2 2 3 2 3 3 2 3 1 3 1 3 2 3 3 3 3 1 2 1 1 1 2 1 2 2 1 3 3 2 1 2 2 1 3 3
[694] 2 2 2 3 2 2 1 1 2 1 2 3 1 3 2 2 3 1 2 3 3 1 3 3 2 1 3 2 1 1 2 1 3 1 1 3 2 2 3 1 2 3 2 2 1 3 2 1 1 3 3 1 1 1 3 3 3 3 3 3 2 3 1
> 
```

2.Elbow methods.

Based on the idea that as the number of clusters increases, the within-cluster sum of squares (WCSS) drops and the between-cluster sum of squares (BCSS) increases, the Elbow approach is a heuristic methodology used to determine the ideal number of clusters in a clustering algorithm. Plotting the WCSS versus the number of clusters and choosing the number of clusters where the rate of WCSS decline begins to level off are both required. The Elbow method can be combined with other techniques to conduct a more thorough study.

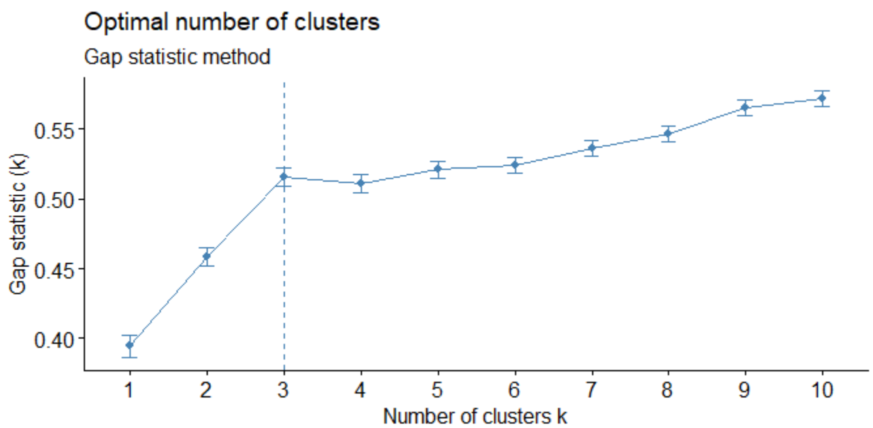
```
#Elbow method
install.packages("factoextra")
library(factoextra)
x11() # creates an X11 graphics device
graphics.off() # reset the graphics device
#plot(x, y) # try plotting again
fviz_nbclust(vehicles_cleaned, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")
```



3. Gap statistics methods.

The ideal number of clusters in a dataset can be found using the statistical method known as gap statistics. It compares the overall within-cluster variation with what would be predicted under a null reference distribution for various values of k (number of clusters). The amount of k that maximizes the gap statistic, or the amount of clustering that is most noticeably improved above the null reference distribution, is the optimal number of clusters.

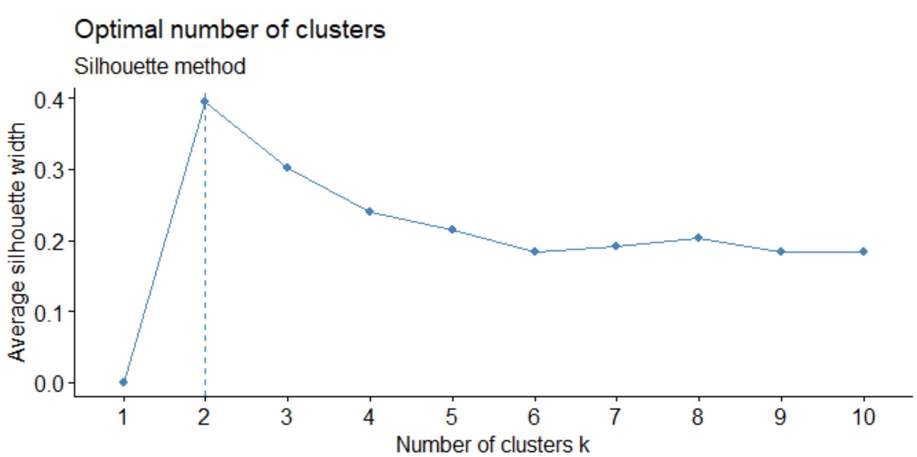
```
# Gap statistic method
set.seed(123)
fviz_nbclust(vehicles_cleaned, kmeans, nstart = 25,  method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")
```



4.silhouette methods.

When comparing an observation to other clusters, the silhouette method, a clustering evaluation technique, determines how similar the observation is to its own cluster. For each observation, it generates a silhouette coefficient that spans from -1 to 1. The overall average is generated for the entire dataset and includes the average silhouette coefficient for each observation in each cluster.

```
# Silhouette method
install.packages("cluster")
library(cluster)
library(factoextra)
fviz_nbclust(vehicles_cleaned, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")
```



(c)

Elbow methods are useful for identifying the optimal number of clusters but have limitations, silhouette methods are useful for determining the quality of the clusters, NbClust methods are useful for determining the optimal number of clusters and the best clustering method, and gap statistics methods are useful for identifying the optimal number of clusters but can be computationally intensive. The particular requirements of the investigation and the features of the dataset will determine which clustering validation technique is used.

```r
#compute k-means clustering with k=3
set.seed(123)
final_stat <- kmeans(vehicles_cleaned,3,nstart = 25)
print(final_stat)

#BSS
BSS <- sum(final_stat$size-(colMeans(vehicles_cleaned)-final_stat$centers)^2)
cat("BSS:",BSS,"\n")

#TSS
TSS <- sum((vehicles_cleaned-colMeans(vehicles_cleaned))^2)
cat("TSS:",TSS,"\n")

#WSS
WSS <- sum(final_stat$withinss)
cat("WSS:",WSS,"\n")

#BSS to TSS
ratio_BSS_to_TSS <- BSS/TSS
cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS,"\n")


```

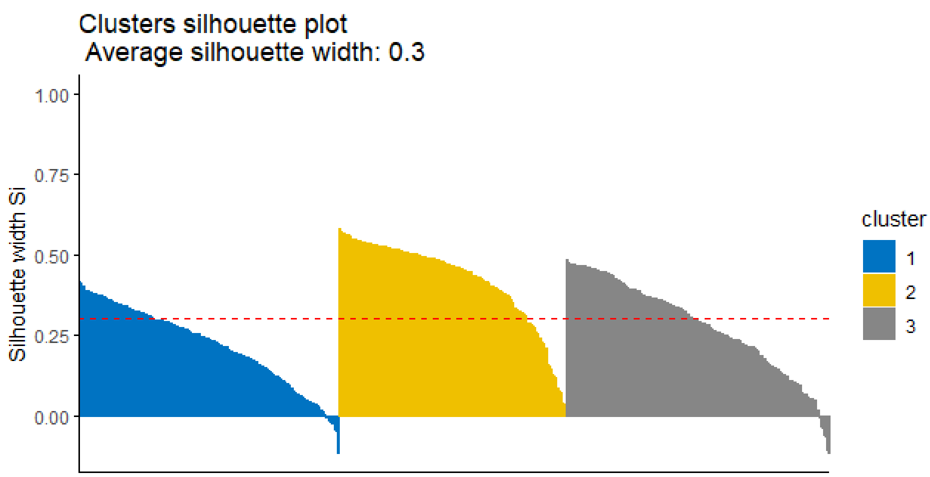
```
> set.seed(123)
> final_stat <- kmeans(vehicles_cleaned,3,nstart = 25)
> print(final_stat)
K-means clustering with 3 clusters of sizes 225, 298, 233

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 Sc.Var.maxis     Ra.Gyr
1 -0.9389574 -0.5120825 -0.8859695 -1.09632688 -0.5719431 -0.3213252 -0.7644035  0.8465480   -0.7323289 -0.4696032   -0.7898951   -0.7723910 -0.3731066
2 -0.2238050 -0.5356122 -0.2794557 -0.03372406  0.2055694 -0.1743615 -0.4489849  0.3091812   -0.4785472 -0.5055057   -0.4150789   -0.4554336 -0.5680657
3  1.0715510  1.1555704  1.1767668  0.98650497  0.1498029  0.2031571  1.2472704 -1.1876186    1.2471203  1.0650855    1.1588940    1.2498769  1.0663029
  Skew.Maxis  Skew.maxis  Kurt.maxis  Kurt.Maxis    Holl.Ra
1  0.7919002 -0.17425926 -0.27461182 -1.06702184 -1.1036953
2 -0.6210913 -0.07257192 -0.06071316  0.77364655  0.6723991
3 -0.1047237  0.05859024  0.24717465  0.01371576  0.1832716

Clustering vector:
  [1] 2 2 3 2 2 2 2 2 2 2 2 2 2 3 1 2 3 3 1 1 2 2 3 2 1 3 3 1 2 2 2 3 2 2 1 3 1 3 1 2 3 1 1 1 1 2 1 2 3 2 3 2 2 1 3 3 1 1 1 2 1 1 2 3 3 3 2 1 2 3 2 3 1 1 3 2 1
 [78] 2 2 1 2 1 3 2 3 2 1 1 1 3 1 2 2 3 3 3 1 1 3 2 2 1 2 3 3 1 2 1 1 2 1 1 1 2 3 3 2 1 3 1 2 1 2 2 1 1 2 3 2 2 3 2 2 3 2 3 2 1 2 2 1 3 2 2 3 3 1 1 3 3 2 3 2 2
[155] 2 2 1 3 1 2 1 3 2 2 3 2 3 2 2 2 1 3 1 1 1 2 2 2 2 2 1 1 3 2 2 2 3 1 2 1 3 1 2 1 3 1 1 2 3 2 3 1 1 1 1 3 2 1 2 1 2 2 1 3 1 1 2 2 3 1 1 3 1 2 2 3 2 3 3 1 2
[232] 2 2 3 1 1 2 2 1 1 2 2 2 1 1 3 2 2 1 1 1 2 2 1 3 1 2 2 3 2 3 1 2 2 3 2 2 2 1 2 3 3 3 3 3 1 2 3 1 1 1 2 1 3 3 1 3 2 1 3 2 2 2 2 3 3 1 3 3 1 3 2 2 2 1 1 3 3
[309] 3 2 2 3 1 2 1 3 2 2 3 2 3 3 3 2 2 1 1 1 1 2 2 2 2 2 1 3 3 1 1 3 1 3 1 2 1 2 2 3 1 2 2 3 2 3 2 2 2 3 2 3 2 1 1 2 2 2 1 1 2 1 3 2 2 1 1 3 2 1 2 2 3 3 2 3 3
[386] 1 1 3 2 2 3 3 1 2 3 3 1 3 3 3 2 2 2 2 2 3 1 2 3 2 2 3 2 3 1 1 3 3 2 1 3 1 3 3 1 2 1 3 3 2 2 1 1 3 2 1 3 3 1 3 3 2 3 1 1 3 3 1 1 3 3 2 2 1 2 3 1 1 3 1 2 2
[463] 1 2 3 2 3 2 1 2 3 3 1 1 2 3 2 3 3 2 2 2 2 1 1 2 2 3 1 1 2 1 3 2 3 1 1 3 3 2 2 2 3 2 1 2 3 2 2 1 3 3 3 3 2 1 1 3 3 3 2 3 1 2 1 1 1 2 1 3 2 2 2 2 2 3 2 2 3
[540] 2 2 1 3 1 1 2 1 2 2 1 3 3 1 2 1 3 2 2 3 2 1 3 1 3 1 2 1 2 3 3 1 3 2 2 1 2 1 3 2 3 1 2 1 1 1 2 3 2 1 2 2 2 2 3 2 1 3 3 2 2 3 1 3 1 2 2 2 1 3 2 1 2 3 1 2 2
[617] 3 1 2 1 2 2 1 2 3 3 2 2 3 3 1 2 3 3 3 3 2 3 2 2 3 3 2 3 2 3 2 3 2 1 3 3 3 2 1 1 3 3 3 2 3 2 2 3 2 1 2 1 2 3 2 2 2 2 1 3 1 1 1 3 1 3 3 1 2 2 3 1 3 3 1 2 2
[694] 3 3 3 2 3 3 1 1 3 1 3 2 1 2 3 3 2 1 3 2 2 1 2 2 3 1 2 3 1 1 3 1 2 1 1 2 3 3 2 1 3 2 3 3 1 2 3 1 1 2 2 1 1 1 2 2 2 2 2 2 3 2 1

Within cluster sum of squares by cluster:
[1] 1293.510 2032.588 1649.251
 (between_SS / total_SS =  57.4 %)

Available components:

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

```
> #BSS
> BSS <- sum(final_stat$size-(colMeans(vehicles_cleaned)-final_stat$centers)^2)
> cat("BSS:",BSS,"\n")
BSS: 13579.7
>
> #TSS
> TSS <- sum((vehicles_cleaned-colMeans(vehicles_cleaned))^2)
> cat("TSS:",TSS,"\n")
TSS: 11694.95
>
> #WSS
> WSS <- sum(final_stat$withinss)
> cat("WSS:",WSS,"\n")
WSS: 4975.349
>
> #BSS to TSS
> ratio_BSS_to_TSS <- BSS/TSS
> cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS,"\n")
ratio_BSS_to_TSS: 1.161159
>
```

(d)

ytruyruru

```

#Silhouette plot
pam.res2 <- pam(vehicles_cleaned,3,metric="euclidean",stand = FALSE)
fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())

```



```
> #Silhouette plot
> pam.res2 <- pam(vehicles_cleaned,3,metric="euclidean",stand = FALSE)
> fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())
  cluster size ave.sil.width
1       1  262          0.22
2       2  229          0.42
3       3  265          0.28
>
```

```
#average Silhouette width score
sil <- silhouette(final_stat$cluster, dist(vehicles_cleaned))
avg_Sil_width <-mean(sil[,3])
cat("average Silhouette width score:",avg_Sil_width,"\n")
```

```
> #average Silhouette width score
> sil <- silhouette(final_stat$cluster, dist(vehicles_cleaned))
> avg_Sil_width <-mean(sil[,3])
> cat("average Silhouette width score:",avg_Sil_width,"\n")
average Silhouette width score: 0.3015821
>
```

2nd Subtask Objectives:

(e) The amount of variance explained by the main components and the complexity of the model must be balanced when deciding how many principal components to keep. In this instance, we decided to keep the eight major components out of the original 18 features that provided a total score of at least 92%. This decision strikes a compromise between the complexity of the modified dataset and the amount of variance that the model can explain.

```
# Show the eigenvalues and eigenvectors
summary(pca_data)
```

```
> # Show the eigenvalues and eigenvectors
> summary(pca_data)

Call:
PCA(X = vehicles_cleaned, graph = FALSE)


Eigenvalues
                       Dim.1   Dim.2   Dim.3   Dim.4   Dim.5   Dim.6   Dim.7   Dim.8   Dim.9  Dim.10  Dim.11  Dim.12  Dim.13  Dim.14  Dim.15  Dim.16  Dim.17
Variance               9.871   3.318   1.206   1.138   0.910   0.643   0.315   0.225   0.113   0.078   0.059   0.042   0.027   0.021   0.016   0.012   0.006
% of var.             54.840  18.434   6.700   6.321   5.057   3.572   1.748   1.249   0.626   0.433   0.330   0.234   0.149   0.117   0.087   0.067   0.035
Cumulative % of var.  54.840  73.274  79.974  86.296  91.353  94.924  96.672  97.921  98.547  98.980  99.309  99.544  99.692  99.809  99.896  99.963  99.998
                      Dim.18
Variance               0.000
% of var.              0.002
Cumulative % of var. 100.000

Individuals (the 10 first)
                Dist    Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3    ctr   cos2  
1           |  2.518 |  0.665  0.006  0.070 |  0.630  0.016  0.063 |  0.511  0.029  0.041 |
2           |  2.189 | -1.497  0.030  0.468 |  0.408  0.007  0.035 |  0.303  0.010  0.019 |
3           |  4.542 |  4.070  0.222  0.803 | -0.336  0.005  0.005 |  1.199  0.158  0.070 |
4           |  3.648 | -1.453  0.028  0.159 |  3.120  0.388  0.731 |  0.461  0.023  0.016 |
5           |  3.572 | -0.712  0.007  0.040 |  2.318  0.214  0.421 |  1.929  0.408  0.292 |
6           |  3.173 | -1.924  0.050  0.368 |  1.617  0.104  0.260 |  1.079  0.128  0.116 |
7           |  5.710 | -4.357  0.254  0.582 |  3.466  0.479  0.369 | -0.568  0.035  0.010 |
8           |  3.185 |  1.617  0.035  0.258 |  2.121  0.179  0.444 | -0.576  0.036  0.033 |
9           |  4.185 | -3.300  0.146  0.622 |  2.170  0.188  0.269 | -0.279  0.009  0.004 |
10          |  5.615 | -4.437  0.264  0.624 |  2.511  0.251  0.200 | -1.522  0.254  0.073 |

Variables (the 10 first)
               Dim.1    ctr   cos2    Dim.2    ctr   cos2    Dim.3    ctr   cos2  
Comp        |  0.857  7.441  0.735 |  0.159  0.763  0.025 |  0.031  0.080  0.001 |
Circ        |  0.893  8.072  0.797 | -0.270  2.195  0.073 |  0.234  4.527  0.055 |
D.Circ      |  0.944  9.032  0.892 |  0.073  0.162  0.005 | -0.076  0.483  0.006 |
Rad.Ra      |  0.865  7.584  0.749 |  0.350  3.689  0.122 | -0.069  0.398  0.005 |
Pr.Axis.Ra  |  0.351  1.249  0.123 |  0.463  6.466  0.215 |  0.083  0.569  0.007 |
Max.L.Ra    |  0.608  3.741  0.369 |  0.149  0.667  0.022 |  0.170  2.405  0.029 |
Scat.Ra     |  0.970  9.531  0.941 | -0.146  0.640  0.021 | -0.126  1.321  0.016 |
Elong       | -0.964  9.418  0.930 |  0.035  0.037  0.001 |  0.106  0.929  0.011 |
Pr.Axis.Rect |  0.960  9.340  0.922 | -0.169  0.861  0.029 | -0.124  1.275  0.015 |
Max.L.Rect  |  0.855  7.401  0.731 | -0.248  1.850  0.061 |  0.257  5.458  0.066 |
> 
```

cumulative score per principals

```r
# Show the cumulative percentage of variance explained
eig_val <- get_eigenvalue(pca_data)
eig_val

cumulative_variances <- cumsum(eig_val/sum(eig_val)*100)
cumulative_variances

barplot(cumulative_variances, main = "Cumulative Percentage of Variance Explained", xlab = "Number of Components", ylab = "Cumulative %")

# Choose the PCs that provide at least cumulative score > 92%
num_components <- length(cumulative_variances[cumulative_variances > 92])
print(paste("Number of components needed to explain at least 92% of the variance:", num_components))

# Create a transformed data set
pca_result <- PCA(vehicles_cleaned, ncp = num_components, graph = FALSE)$ind$coord

```

```
> # Show the cumulative percentage of variance explained
> eig_val <- get_eigenvalue(pca_data)
> eig_val
         eigenvalue variance.percent cumulative.variance.percent
Dim.1  9.8712495125      54.840275070                    54.84028
Dim.2  3.3180897921      18.433832178                    73.27411
Dim.3  1.2060436568       6.700242538                    79.97435
Dim.4  1.1378385229       6.321325127                    86.29567
Dim.5  0.9102549455       5.056971920                    91.35265
Dim.6  0.6428806461       3.571559145                    94.92421
Dim.7  0.3146665491       1.748147495                    96.67235
Dim.8  0.2247308345       1.248504636                    97.92086
Dim.9  0.1127178635       0.626210353                    98.54707
Dim.10 0.0778662792       0.432590440                    98.97966
Dim.11 0.0593239686       0.329577603                    99.30924
Dim.12 0.0421707175       0.234281764                    99.54352
Dim.13 0.0267363587       0.148535326                    99.69205
Dim.14 0.0210310931       0.116839406                    99.80889
Dim.15 0.0156955629       0.087197572                    99.89609
Dim.16 0.0120530603       0.066961446                    99.96305
Dim.17 0.0062928513       0.034960285                    99.99801
Dim.18 0.0003577853       0.001987696                   100.00000
> 
> cumulative_variances <- cumsum(eig_val/sum(eig_val)*100)
> cumulative_variances
 [1]   0.5517772   0.7372497   0.8046644   0.8682667   0.9191475   0.9550829   0.9726719   0.9852338   0.9915344   0.9958870   0.9992030   1.0015602
[13]   1.0030547   1.0042303   1.0051077   1.0057814   1.0061332   1.0061532   4.0715818   5.1019851   5.4765112   5.8298568   6.1125284   6.3121692
[25]   6.4098861   6.4796743   6.5146778   6.5388585   6.5572810   6.5703768   6.5786795   6.5852105   6.5900846   6.5938276   6.5957818   6.5958929
[37]   9.6613216  13.7571535  18.2275115  23.0512151  28.1575903  33.4636064  38.8673394  44.3408605  49.8493852  55.3820905  60.9332184  66.4974420
[49]  72.0699683  77.6490257  83.2329572  88.8206316  94.4102603 100.0000000
> 
> barplot(cumulative_variances, main = "Cumulative Percentage of Variance Explained", xlab = "Number of Components", ylab = "Cumulative %")
> 
> # Choose the PCs that provide at least cumulative score > 92%
> num_components <- length(cumulative_variances[cumulative_variances > 92])
> print(paste("Number of components needed to explain at least 92% of the variance:", num_components))
[1] "Number of components needed to explain at least 92% of the variance: 2"
> 
> # Create a transformed data set
> pca_result <- PCA(vehicles_cleaned, ncp = num_components, graph = FALSE)$ind$coord
> 
```

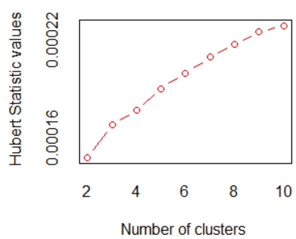
(f)

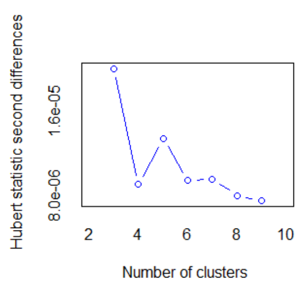
1.NbClust methods.

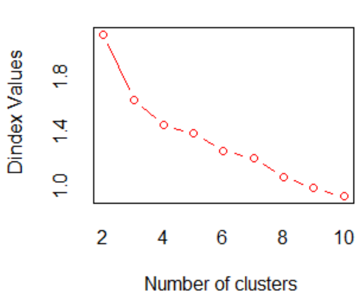
```r
#NbClust method
library(NbClust)
nb <- NbClust(pca_result, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
print(nb)
```

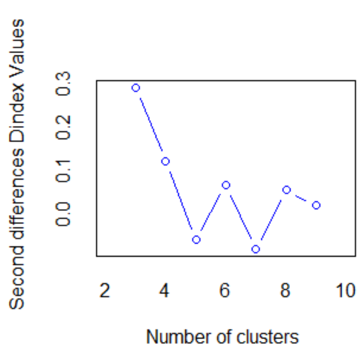
```
> nb <- NbClust(pca_result, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

*******************************************************************
* Among all indices:
* 6 proposed 2 as the best number of clusters
* 10 proposed 3 as the best number of clusters
* 2 proposed 4 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 2 proposed 6 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


*******************************************************************
Warning message:
did not converge in 10 iterations
> print(nb)
$All.index
       KL        CH Hartigan     CCC    Scott  Marriot     TrCovW    TraceW Friedman   Rubin Cindex     DB Silhouette   Duda  Pseudot2   Beale Ratkowsky
2  1.7939 1026.7720 503.8277  1.2932 1126.730 16869357 3404373.7 4221.8991   3.3945  2.3618 0.3109 0.7788     0.5159 0.5500  428.7110  0.8165    0.3384
3  2.7089 1106.8778 211.0906  5.1869 1901.026 13629387 2388541.1 2530.8012   5.8992  3.9399 0.2539 0.8741     0.4646 1.6603 -159.0779 -0.3958    0.4722
4  0.4762 1013.7967 115.5794  3.0168 2238.307 15509508  949178.7 1976.6746   8.5981  5.0444 0.2451 0.9531     0.4182 1.5443 -131.4722 -0.3506    0.4235
5  5.2637  904.8999 163.5866 -0.0344 2454.258 18212182  622167.1 1713.3408   9.5297  5.8197 0.3077 0.9188     0.4017 0.8684   35.5976  0.1507    0.3929
6  0.1811  913.1075  58.6729  0.3563 2758.200 17543696  394364.7 1406.8860  12.7778  7.0874 0.3623 0.9201     0.3884 1.3379  -56.5676 -0.2508    0.3661
7  0.8802  829.1222 187.6605 -2.2779 2871.427 20557470  393940.7 1304.8099  14.3575  7.6418 0.3597 1.0806     0.3466 0.7966   52.0788  0.2538    0.3408
8  2.1899  914.3210 131.3368  0.5076 3203.141 17313947  315969.2 1043.3905  17.2437  9.5565 0.3379 0.9793     0.3703 2.9648 -131.8793 -0.6549    0.3278
9  3.1578  955.6413  94.9891  1.7695 3446.987 15871586  186681.0  887.5508  21.3095 11.2344 0.3106 0.9275     0.3677 2.8925 -100.1042 -0.6468    0.3121
10 0.2448  966.7352 126.0526  2.0964 3649.562 14988686  143151.2  787.4217  26.4407 12.6630 0.2906 0.9254     0.3706 2.3506 -103.9981 -0.5678    0.2974
        Ball Ptbiserial   Frey McClain   Dunn Hubert SDindex Dindex   SDbw
2  2110.9496     0.6616 0.9159  0.3884 0.0166  1e-04  0.8477 2.1139 0.8785
3   843.6004     0.6428 1.1878  0.7790 0.0090  2e-04  0.8656 1.6358 0.7715
4   494.1686     0.5784 0.2506  1.1316 0.0070  2e-04  1.0481 1.4501 0.8115
5   342.6682     0.5813 0.5548  1.1634 0.0152  2e-04  0.9829 1.3856 0.5531
6   234.4810     0.5627 2.4585  1.3089 0.0328  2e-04  0.9652 1.2620 0.5049
7   186.4014     0.5210 0.4230  1.5682 0.0241  2e-04  1.5915 1.2057 0.4181
8   130.4238     0.4992 0.4823  1.7549 0.0200  2e-04  1.5269 1.0691 0.3674
9    98.6168     0.4794 0.4378  1.9118 0.0190  2e-04  1.5050 0.9880 0.2753
10   78.7422     0.4649 0.4277  2.0245 0.0200  2e-04  1.5098 0.9277 0.2532

$All.CriticalValues
  CritValue_Duda CritValue_PseudoT2 Fvalue_Beale
2         0.5713           393.1612       0.4423
3         0.5134           379.1353       1.0000
4         0.5045           366.3245       1.0000
5         0.5012           233.8476       0.8602
```

```
5          0.5012          233.8476       0.8602
6          0.4760          246.5460       1.0000
7          0.4913          211.2269       0.7760
8          0.4156          279.8445       1.0000
9          0.4186          212.4822       1.0000
10         0.4156          254.5320       1.0000

$Best.nc
                   KL       CH Hartigan    CCC    Scott Marriot  TrCovW   TraceW Friedman   Rubin Cindex     DB Silhouette   Duda  PseudoT2  Beale Ratkowsky
Number_clusters 5.0000    3.000   3.0000 3.0000   3.0000       3       4    3.000  10.0000  6.0000 4.0000 2.0000     2.0000 3.0000    3.0000 2.0000    3.0000
Value_Index     5.2637 1106.878 292.7371 5.1869 774.2957 5120090 1439362 1136.971   5.1311 -0.7132 0.2451 0.7788     0.5159 1.6603 -159.0779 0.8165    0.4722
                    Ball PtBiserial Frey McClain   Dunn Hubert SDindex Dindex    SDbw
Number_clusters    3.000     2.0000    1  2.0000 6.0000      0  2.0000      0 10.0000
Value_Index     1267.349     0.6616   NA  0.3884 0.0328      0  0.8477      0  0.2532

$Best.partition
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39
  3   3   2   3   3   3   3   3   3   3   3   3   3   2   1   3   2   2   1   1   3   3   2   3   1   2   2   1   3   3   3   2   3   3   1   2   1   2   1
 40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78
  3   2   1   1   1   3   3   1   3   2   3   2   3   3   1   2   2   1   1   1   3   1   1   3   2   2   2   3   1   3   2   3   2   1   1   2   3   1   3
 79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
  3   1   3   1   2   3   2   3   1   1   1   2   1   3   3   2   2   2   1   1   3   3   3   1   3   2   2   1   3   1   1   3   3   3   1   3   2   2   3
118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156
  1   2   1   3   1   3   3   1   1   3   2   3   3   2   3   3   2   3   2   3   1   3   3   1   2   3   1   2   2   1   1   2   2   3   2   3   3   3   3
157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195
  1   2   1   3   1   2   3   3   2   3   2   3   3   3   1   2   1   1   1   3   3   3   3   3   1   1   2   3   3   3   2   1   3   1   2   1   3   1   2
196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
  1   1   3   2   3   2   1   1   1   1   2   3   1   3   1   3   3   1   2   1   1   3   3   2   1   1   2   1   3   3   2   3   2   2   1   3   3   3   2
235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273
  1   1   3   3   1   1   3   3   3   1   1   2   3   3   1   1   1   3   3   1   2   1   3   3   2   3   2   1   3   3   2   3   3   3   1   3   2   2   2
274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312
  2   2   3   3   2   1   1   1   3   1   2   2   1   2   3   1   2   3   3   3   3   2   2   1   2   2   1   2   3   3   3   1   1   2   2   2   3   3   2
313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351
  1   3   1   2   3   3   2   3   2   2   2   3   3   1   3   1   1   3   3   3   3   3   1   2   2   1   1   2   1   2   1   3   3   3   3   2   1   3   3
352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390
  3   3   2   3   3   3   2   3   2   3   1   1   3   3   3   1   1   3   1   2   3   3   1   1   2   3   1   3   3   2   2   3   2   2   1   1   2   3   3
391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429
  2   2   1   1   2   2   1   2   2   2   3   3   3   3   3   2   1   3   2   3   3   2   3   2   1   1   2   2   3   3   2   1   2   2   3   3   1   2   2
430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468
  3   3   1   1   2   3   1   2   2   1   2   2   3   2   3   1   2   2   1   1   2   2   3   3   1   3   2   1   1   2   1   3   3   1   3   2   3   2   3
469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507
  1   3   2   2   1   1   3   2   3   2   2   3   3   3   3   1   1   3   3   2   1   1   3   1   2   3   2   1   1   2   2   3   3   3   2   3   1   3   2
508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546
  3   3   1   2   2   2   2   3   1   1   2   2   2   3   2   1   3   1   1   1   3   1   3   3   3   3   3   3   2   3   3   2   3   3   1   2   1   1   3
547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585
  1   3   3   1   2   2   1   3   1   2   3   3   2   3   1   2   1   2   1   3   1   3   2   2   3   2   3   3   1   3   1   2   3   2   1   3   1   3   3
586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624
  3   2   3   1   3   3   3   3   2   3   1   2   2   3   3   2   1   2   1   3   3   3   1   2   3   1   3   2   1   3   3   2   1   3   1   3   3   1   3
625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663
  2   2   3   3   2   2   1   3   2   2   2   2   3   2   3   3   2   2   3   2   3   2   3   2   3   1   2   2   2   3   1   1   2   2   2   3   2   3   3
664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702
  2   3   1   3   1   3   2   3   3   3   3   1   2   1   1   1   2   1   2   2   1   3   3   2   1   2   2   1   3   3   2   2   2   3   2   2   1   1   2
703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741
  1   2   3   1   3   2   2   3   1   2   3   3   1   3   3   2   1   3   2   1   1   2   1   3   1   1   3   2   2   3   1   2   3   2   2   1   3   2   1
742 743 744 745 746 747 748 749 750 751 752 753 754 755 756
  1   3   3   1   1   1   3   3   3   3   3   3   2   3   1
```



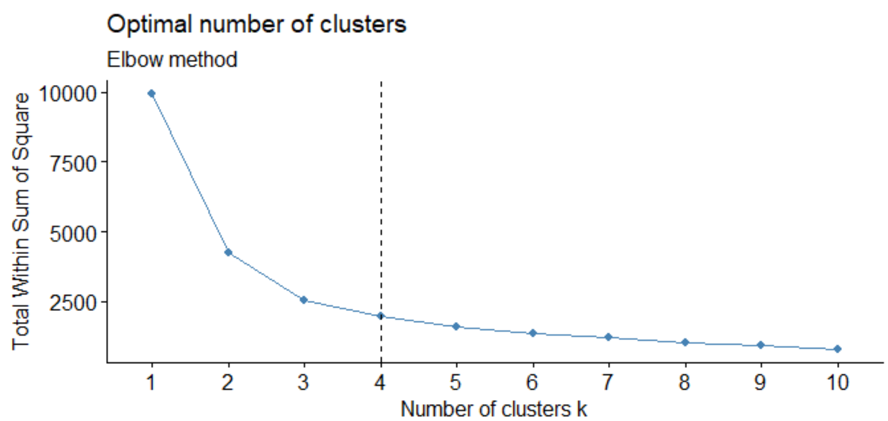






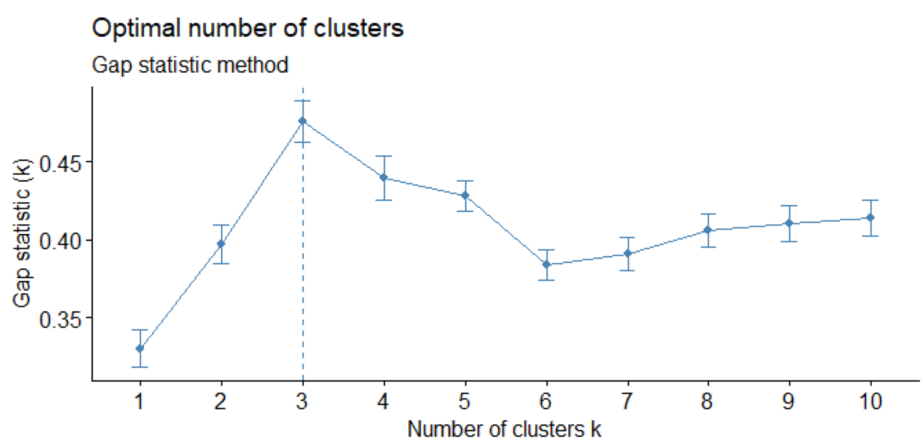
2.Elbow methods.

```r
#Elbow method
x11() # creates an X11 graphics device
graphics.off() # reset the graphics device
#plot(x, y) # try plotting again
fviz_nbclust(pca_result, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")

```



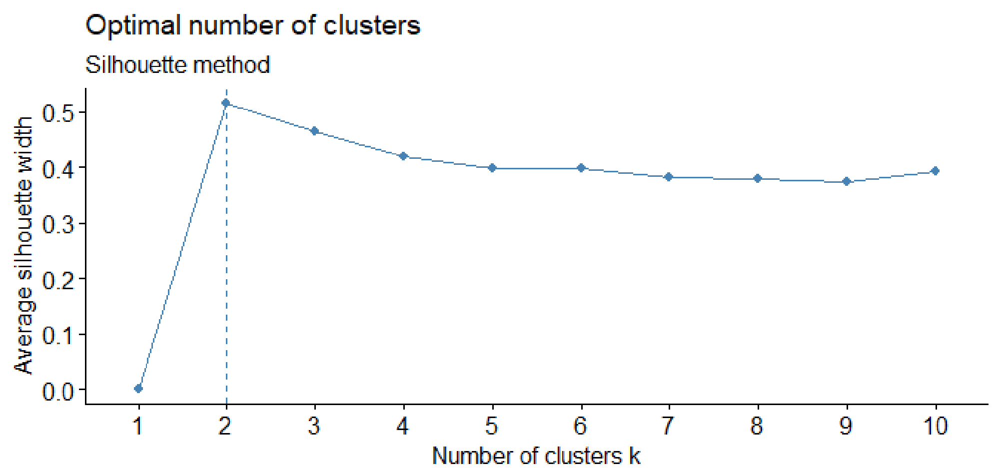
3.Gap statistics methods

```r
# Gap statistic method
set.seed(123)
fviz_nbclust(pca_result, kmeans, nstart = 25,  method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")

```



4.silhouette methods.

```r
# Silhouette method
library(cluster)
library(factoextra)
fviz_nbclust(pca_result, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")
```



(g)

```r
#compute k-means clustering with k=3
set.seed(123)
final_stat_pca <- kmeans(pca_result,3,nstart = 25)
print(final_stat_pca)

#BSS
BSS_pca <- sum(final_stat_pca$size-(colMeans(pca_result)-final_stat_pca$centers)^2)
cat("BSS:",BSS_pca,"\n")

#TSS
TSS_pca <- sum((pca_result-colMeans(pca_result))^2)
cat("TSS:",TSS_pca,"\n")

#WSS
WSS_pca <- sum(final_stat_pca$withinss)
cat("WSS:",WSS_pca,"\n")

#BSS to TSS
ratio_BSS_to_TSS_pca <- BSS_pca/TSS_pca
cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS_pca,"\n")
```

```
> #compute k-means clustering with k=3
> set.seed(123)
> final_stat_pca <- kmeans(pca_result,3,nstart = 25)
> print(final_stat_pca)
K-means clustering with 3 clusters of sizes 215, 311, 230

Cluster means:
      Dim.1      Dim.2
1 -2.953007 -1.7239364
2 -1.006285  1.4707148
3  4.121092 -0.3771564

Clustering vector:
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
   2   2   3   2   2   2   2   2   2   2   2   2   2   3   1   2   3   3   1   1   2   2   3   2   1   3   3   1   2   2   2   3   2   2   1   3   1   3   1   2   3   1   1   1   2   2   1   2   3   2   3   2   2   1   3   3   1   1   1   2
  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
   1   1   2   3   3   3   2   1   2   3   2   3   1   1   3   2   1   2   2   1   2   1   3   2   3   2   1   1   1   3   1   2   2   3   3   3   1   1   2   2   2   1   2   3   3   1   2   1   1   2   2   2   1   2   3   3   2   1   3   1
 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
   2   1   2   2   1   1   2   3   2   2   3   2   2   3   2   3   2   1   2   2   1   3   2   1   3   3   1   1   3   3   2   3   2   2   2   2   1   3   1   2   1   3   2   2   3   2   3   2   2   2   1   3   1   1   1   2   2   2   2   2
 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240
   1   1   3   2   2   2   3   1   2   1   3   1   2   1   3   1   1   2   3   2   3   1   1   1   1   3   2   1   2   1   2   2   1   3   1   1   2   2   3   1   1   3   1   2   2   3   2   3   3   1   2   2   2   3   1   1   2   2   1   1
 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300
   2   2   2   1   1   3   2   2   1   1   1   2   2   1   3   1   2   2   3   2   3   1   2   2   3   2   2   2   1   2   3   3   3   3   3   2   2   3   1   1   1   2   1   3   3   1   3   2   1   3   2   2   2   2   3   3   1   3   3   1
 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360
   3   2   2   2   1   1   3   3   3   2   2   3   1   2   1   3   2   2   3   2   3   3   3   2   2   1   2   1   1   2   2   2   2   2   1   3   3   1   1   3   1   3   1   2   2   2   2   3   1   2   2   2   2   3   2   2   2   3   2   3
 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420
   2   1   1   2   2   2   1   1   2   1   3   2   2   1   1   3   2   1   2   2   3   3   2   3   3   1   1   3   2   2   3   3   1   1   3   3   1   3   3   3   2   2   2   2   2   3   1   2   3   2   2   3   2   3   1   1   3   3   2   2
 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480
   3   1   3   3   2   2   1   3   3   2   2   1   1   3   2   1   3   3   1   3   3   2   3   2   1   3   3   1   1   3   3   2   2   1   2   3   1   1   3   1   2   2   1   2   3   2   3   2   1   2   3   3   1   1   2   3   2   3   3   2
 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540
   2   2   2   1   1   2   2   3   1   1   2   1   3   2   3   1   1   3   3   2   2   2   3   2   1   2   3   2   2   1   3   3   3   3   2   1   1   3   3   3   2   3   1   2   1   1   1   2   1   2   2   2   2   2   2   3   2   2   3   2
 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600
   2   1   3   1   1   2   1   2   2   1   3   3   1   2   1   3   2   2   3   2   1   3   1   3   1   2   1   2   3   3   2   3   2   2   1   2   1   3   2   3   1   2   1   2   2   2   3   2   1   2   2   2   2   3   2   1   3   3   2   2
 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660
   3   1   3   1   2   2   2   1   3   2   1   2   3   1   2   2   3   1   2   1   2   2   1   2   3   3   2   2   3   3   1   2   3   3   3   3   2   3   2   2   3   3   2   3   2   3   2   3   2   1   3   3   3   2   1   1   3   3   3   2
 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720
   3   2   2   3   2   1   2   1   2   3   2   2   2   2   1   3   1   1   1   3   1   3   3   1   2   2   3   1   3   3   1   2   2   3   3   3   2   3   3   1   1   3   1   3   2   1   2   3   3   2   1   3   2   2   1   2   2   3   1   2
 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756
   3   1   1   3   1   2   1   1   2   3   3   2   1   3   2   3   3   1   2   3   1   1   2   2   1   1   1   2   2   2   2   2   2   3   2   1

Within cluster sum of squares by cluster:
[1]  703.8015 1046.2561  780.7436
 (between_SS / total_SS =  74.6 %)

Available components:

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

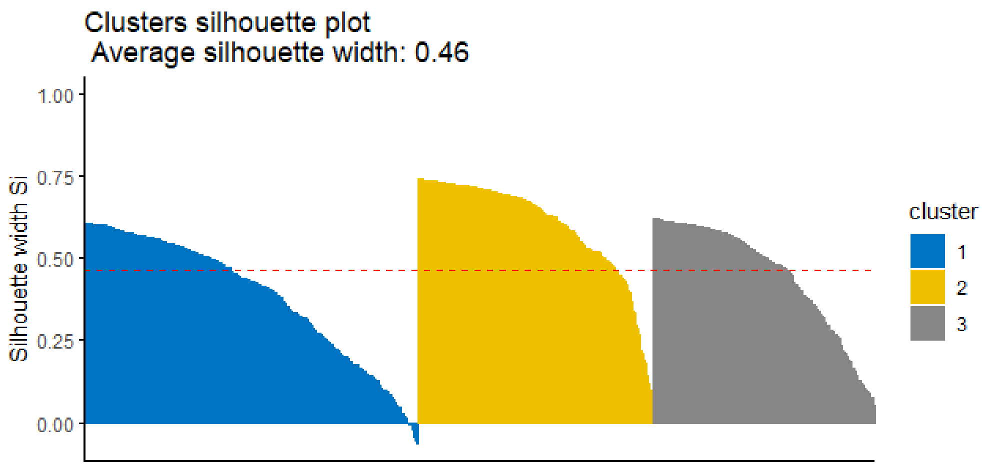
```
> #BSS
> BSS_pca <- sum(final_stat_pca$size-(colMeans(pca_result)-final_stat_pca$centers)^2)
> cat("BSS:",BSS_pca,"\n")
BSS: 1480.007
>
> #TSS
> TSS_pca <- sum((pca_result-colMeans(pca_result))^2)
> cat("TSS:",TSS_pca,"\n")
TSS: 9971.141
>
> #WSS
> WSS_pca <- sum(final_stat_pca$withinss)
> cat("WSS:",WSS_pca,"\n")
WSS: 2530.801
>
> #BSS to TSS
> ratio_BSS_to_TSS_pca <- BSS_pca/TSS_pca
> cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS_pca,"\n")
ratio_BSS_to_TSS: 0.148429
>
```

(h)

```
#Silhouette plot
library(cluster)
pam.res2 <- pam(pca_result,3,metric="euclidean",stand = FALSE)
fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())

#average Silhouette width score
sil <- silhouette(final_stat_pca$cluster, dist(pca_result))
avg_sil_width <-mean(sil[,3])
cat("average Silhouette width score:",avg_sil_width,"\n")
```

```
> pam.res2 <- pam(pca_result,3,metric="euclidean",stand = FALSE)
> fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())
  cluster size ave.sil.width
1       1  319          0.38
2       2  225          0.60
3       3  212          0.45
>
```

```
> #average Silhouette width score
> sil <- silhouette(final_stat_pca$cluster, dist(pca_result))
> avg_Sil_width <-mean(sil[,3])
> cat("average Silhouette width score:",avg_Sil_width,"\n")
average Silhouette width score: 0.4645933
```



(i) Calinski-Harabasz Index

```
library(fpc)
ch_index <- calinhara(pca_result, final_stat_pca$cluster)
print(ch_index)
barplot(ch_index, main="Calinski-Harabasz Index for K-Means Clustering", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")
plot(ch_index, type="b", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")
```

```
> library(fpc)
> ch_index <- calinhara(pca_result, final_stat_pca$cluster)
> print(ch_index)
[1] 1106.878
```

Energy Forecasting Part (part of Work Based Learning Activity)

Objectives/Deliverables (Multi-layers Neural Network)

1st Subtask Objectives:

(a)

Time-based Approach: - Time-based characteristics can be an effective tool for defining the input vector when dealing with predicting issues for electricity load. By incorporating elements like the day of the week, the hour of the day, and the month of the year, the neural network can detect seasonality and temporal trends in the data. If the dataset contains hourly electricity consumption data for several years, the neural network, for example, can use time-based characteristics to distinguish between weekday and weekend patterns, morning and evening peaks, and seasonal swings in demand.

Calendar-based Approach: - In this method, the input variables are calendar elements like the day of the week, the month of the year, and the holiday indicators. This strategy acknowledges that the amount of electricity consumed has a seasonal rhythm and is influenced by occasions on the calendar, such as vacations. This strategy has been demonstrated to increase the precision of load forecasting models.

Weather-based Approach: - This method makes use of weather input variables like temperature, humidity, and wind speed. This strategy acknowledges how the demand for electricity, particularly for heating and cooling, is influenced by the weather. This strategy has also been demonstrated to increase the precision of load forecasting models.

Economic-based Approach: - Economic input factors used in this strategy include GDP, employment, and industrial production. This strategy acknowledges how the demand for electricity is influenced by the economy, particularly in the industrial and commercial sectors. This strategy has also been demonstrated to increase the precision of load forecasting models.

(b)

```r
library(dplyr)
library(neuralnet)
library(ggplot2)
library(readxl)
setwd("D:/2nd Year/ML_CourseWork")
UOW_data <- read_excel("uow_consumption.xlsx")

# Rename columns
colnames(UOW_data) <- c("date", "time_eighteen", "time_nineteen", "time_twenty")
head(UOW_data)

```

```
> # Rename columns
> colnames(UOW_data) <- c("date", "time_eighteen", "time_nineteen", "time_twenty")
> head(UOW_data)
# A tibble: 6 × 4
  date                time_eighteen time_nineteen time_twenty
  <dttm>                      <dbl>         <dbl>       <dbl>
1 2018-01-01 00:00:00          38.9          38.9        38.9
2 2018-01-02 00:00:00          42.3          41.9        41.9
3 2018-01-03 00:00:00          40.8          40.5        40.7
4 2018-01-04 00:00:00          42.3          41.9        41.9
5 2018-01-05 00:00:00          44            44.1        44
6 2018-01-06 00:00:00          45.6          44.5        44.3
>
```

```
#apply lag method
UOW_data$lag_1 <- lag(UOW_data$time_twenty, 1)
UOW_data$lag_2 <- lag(UOW_data$time_twenty, 2)
UOW_data$lag_3 <- lag(UOW_data$time_twenty, 3)
UOW_data$lag_4 <- lag(UOW_data$time_twenty, 4)
UOW_data$lag_7 <- lag(UOW_data$time_twenty, 7)
UOW_data <- na.omit(UOW_data)

#TASK 3
#dividing data to testing and training
UOW_train <- UOW_data[1:380,]
UOW_test <- UOW_data[381:nrow(UOW_data),]
```

(c)

Normalization

normalizing data before using them in an MLP structure can help improve the convergence, avoid bias, and improve the performance of the MLP. Normalization is a standard pre-processing step in machine learning, and it is important to carefully choose the normalization method that is appropriate for the specific problem and data set.

```
#TASK 4
#normalization

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

# Exclude the date column from normalization
UOW_train_normalized <- as.data.frame(lapply(UOW_train[-1], normalize))
UOW_test_normalized <- as.data.frame(lapply(UOW_test[-1], normalize))

# Set the column names of the test_normalized data frame
colnames(UOW_test_normalized) <- colnames(UOW_train_normalized)
```

(d)

```r

input_vectors <- list(
  c("lag_1"),
  c("lag_1", "lag_2"),
  c("lag_1", "lag_2", "lag_3"),
  c("lag_1", "lag_2", "lag_3", "lag_4"),
  c("lag_1", "lag_7"),
  c("lag_1", "lag_2", "lag_7"),
  c("lag_1", "lag_2", "lag_3", "lag_7"),
  c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7")
)

build_mlp_model <- function(train_data, test_data, input_vars, hidden_structure) {
  formula <- paste("time_twenty ~", paste(input_vars, collapse = " + "))
  nn <- neuralnet(as.formula(formula), train_data, hidden = hidden_structure)
  test_matrix <- as.matrix(test_data[, input_vars, drop = FALSE])
  colnames(test_matrix) <- colnames(train_data[, input_vars, drop = FALSE])
  predictions <- predict(nn, test_matrix)
  return(list(model = nn, predictions = predictions))
}

models <- list()
for (i in 1:length(input_vectors)) {
  models[[i]] <- build_mlp_model(UoW_train_normalized, UoW_test_normalized, input_vectors[[i]], c(5))
}

```

(e)

RMSE: - The average difference between predicted values and actual values is measured by the term "RMSE," which stands for root mean squared error. The square root of the average of the squared differences between the predicted and actual values is used to calculate RMSE. To assess the precision of predictions, regression analysis frequently uses RMSE.

MAE: - MAE, or mean absolute error, is another way to quantify the average difference between projected and actual data. The average of the absolute discrepancies between projected and actual values is used to determine MAE. To assess the precision of predictions in regression analysis, MAE is frequently utilized.

MAPE: - Mean Absolute Percentage Error, or MAPE, is a measurement of the typical percentage difference between predicted values and actual values. By averaging the absolute percentage deviations between projected and actual values, MAPE is calculated. When the scale of the data varies significantly from prediction to prediction, MAPE is frequently used to assess the accuracy of forecasts.

sMAPE: - A modification of MAPE that overcomes some of its drawbacks is called sMAPE, or symmetric MAPE. sMAPE accounts for the magnitude of the projected and actual values when calculating the average percentage difference between predicted and actual values. The problem of MAPE creating infinite or undefined values when the actual values are zero is helped by this.

```r
calculate_metrics <- function(actual, predicted) {
  rmse <- sqrt(mean((actual - predicted)^2))
  mae <- mean(abs(actual - predicted))
  mape <- mean(abs(actual - predicted) / predicted)
  smape <- mean(abs(actual - predicted) / (abs(actual) + abs(predicted)) * 2) * 100
  return(list(RMSE = rmse, MAE = mae, MAPE = mape, SMAPE = smape))
}

evaluation_metrics <- list()
for (i in 1:length(models)) {
  evaluation_metrics[[i]] <- calculate_metrics(UoW_test_normalized$time_twenty, models[[i]]$predictions)
}

```

(d)

```r
#Create a comparison table of their testing performances
comparison_table <- data.frame(
  Model_Description = c("AR(1)", "AR(2)", "AR(3)", "AR(4)", "AR(1,7)", "AR(2,7)", "AR(3,7)", "AR(4,7)"),
  RMSE = sapply(evaluation_metrics, function(x) x$RMSE),
  MAE = sapply(evaluation_metrics, function(x) x$MAE),
  MAPE = sapply(evaluation_metrics, function(x) x$MAPE),
  sMAPE = sapply(evaluation_metrics, function(x) x$sMAPE)
)

print(comparison_table)
```

```
> print(comparison_table)
  Model_Description      RMSE       MAE      MAPE    sMAPE
1             AR(1) 0.1951943 0.1560261 0.3217749 34.11284
2             AR(2) 0.1973886 0.1587065 0.3223764 34.44747
3             AR(3) 0.2077762 0.1685484 0.3324528 35.63873
4             AR(4) 0.2087940 0.1692384 0.3517057 36.37538
5           AR(1,7) 0.1627568 0.1313272 0.2663793 28.67405
6           AR(2,7) 0.1635911 0.1306763 0.2638402 28.61209
7           AR(3,7) 0.1602318 0.1300400 0.2664877 28.21815
8           AR(4,7) 0.1677774 0.1342914 0.2765529 29.36676
>
```

(g)

The one-hidden layer network in this situation has fewer weight parameters (2 vs. 3) than the two-hidden layer network. As a result, the one-hidden layer network uses fewer weight parameters than other networks.

```r
# Efficiency comparison between one-hidden layer and two-hidden layer networks

model_1_hidden <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(5))
model_2_hidden <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(3, 2))

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

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

```
> model_1_hidden <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(5))
> model_2_hidden <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(3, 2))
>
> # Check the total number of weight parameters per network
> num_weights_1_hidden <- sum(sapply(model_1_hidden$model$weights, length))
> num_weights_2_hidden <- sum(sapply(model_2_hidden$model$weights, length))
>
> cat("Total number of weight parameters for the one-hidden layer network:", num_weights_1_hidden, "\n")
Total number of weight parameters for the one-hidden layer network: 2
> cat("Total number of weight parameters for the two-hidden layer network:", num_weights_2_hidden, "\n")
Total number of weight parameters for the two-hidden layer network: 3
>
```

2nd Subtask Objectives:

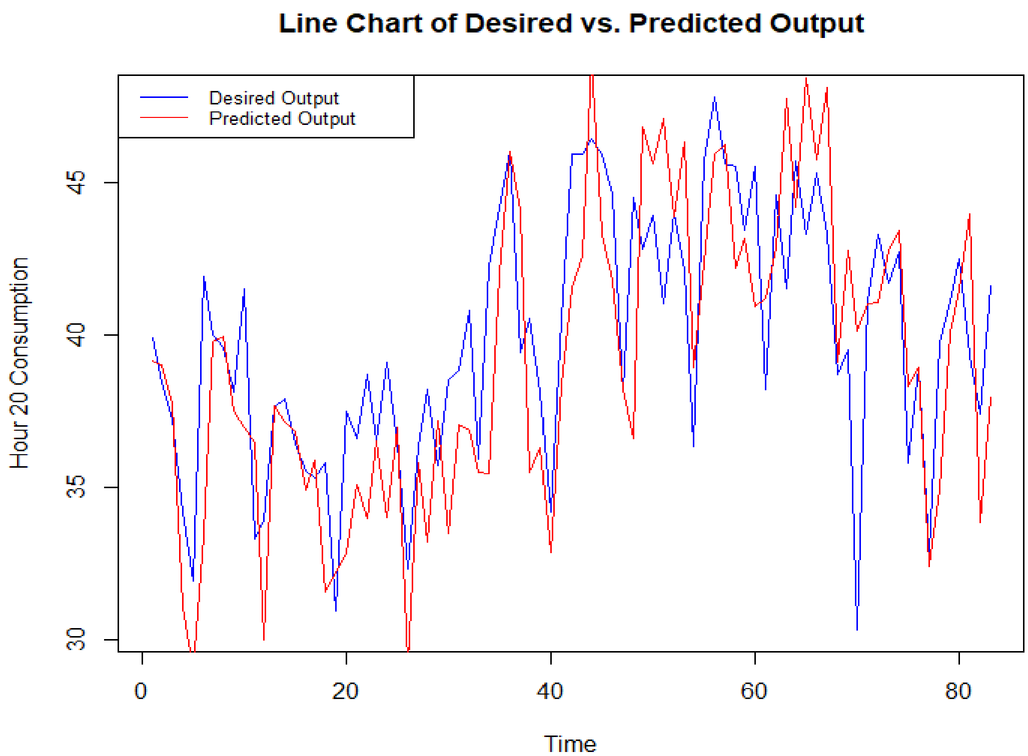
(h)

```r

#Task1
# Add the 18th and 19th hour attributes to the input vectors
narx_input_vectors <- list(
  c("lag_1", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "lag_3", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "lag_3", "lag_7", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7", "time_eighteen", "time_nineteen")
)

# Build NARX models
narx_models <- list()
for (i in 1:length(narx_input_vectors)) {
  narx_models[[i]] <- build_mlp_model(UoW_train_normalized, UoW_test_normalized, narx_input_vectors[[i]], c(5))
}

# Evaluate NARX models
narx_evaluation_metrics <- list()
for (i in 1:length(narx_models)) {
  narx_evaluation_metrics[[i]] <- calculate_metrics(UoW_test_normalized$time_twenty, narx_models[[i]]$predictions)
}

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

print(narx_comparison_table)
```

```
> print(narx_comparison_table)
  Model_Description       RMSE        MAE       MAPE    SMAPE
1     NARX(1,18,19) 0.08172061 0.04922087 0.09076291 14.78226
2     NARX(2,18,19) 0.07980414 0.04915392 0.09037292 14.44573
3     NARX(3,18,19) 0.07889581 0.05096860 0.09539820 14.85292
4   NARX(3,7,18,19) 0.07446744 0.04955539 0.09246081 14.70782
5   NARX(4,7,18,19) 0.07237179 0.04913312 0.12046973 15.12552
```

(i)

```r

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

best_model_index <- which.min(sapply(evaluation_metrics, function(x) x$RMSE))
best_model <- models[[best_model_index]]
best_model_predictions <- best_model$predictions

min_value <- min(UoW_train$time_twenty)
max_value <- max(UoW_train$time_twenty)

denormalized_predictions <- denormalize(best_model_predictions, min_value, max_value)

# Plot the predicted output vs. desired output using a line chart
plot(UoW_test$time_twenty, type = "l", col = "blue", xlab = "Time", ylab = "Hour 20 Consumption", main = "Line Chart of Desired vs. Predicted Output")
lines(denormalized_predictions, col = "red")
legend("topleft", legend = c("Desired Output", "Predicted Output"), col = c("blue", "red"), lty = 1, cex = 0.8)

```


Line Chart of Desired vs. Predicted Output
Desired Output
Predicted Output
Hour 20 Consumption
30
35
40
45
0
20
40
60
80
Time

Appendix

Part 1 sub task 1,

```r
#lord the data set
install.packages("readxl")
library(readxl)
setwd("D:/2nd Year/ML_CourseWork")
vehicles <- read_excel("vehicles.xlsx")

#define 18 attributes
install.packages("dplyr")
library(dplyr)
vehicles_subset <- select(vehicles,Comp:Holl.Ra)

#Scaling.
vehicles_scaled<-scale(vehicles_subset)

#set outlines using IQR method.
out1<- apply(vehicles_scaled,2,quantile,probs=0.25,na.rm = TRUE)
out3<- apply(vehicles_scaled,2,quantile,probs=0.75,na.rm =TRUE)
IQR<- out3 - out1

#identifies outliers.
outliers<-apply(vehicles_scaled,2,function(x){
  lower <- out1-1.5*IQR
  upper <- out3+1.5*IQR
  x < lower | x > upper
})

#remove outliers.
vehicles_cleaned <- vehicles_scaled
vehicles_cleaned[outliers] <- NA
vehicles_cleaned <- na.omit(vehicles_cleaned)

#determine the number of cluster centers
#NbClust method
install.packages("NbClust")
library(NbClust)
nb <- NbClust(vehicles_cleaned, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
print(nb)

#Elbow method
install.packages("factoextra")
library(factoextra)
x11() # creates an X11 graphics device
graphics.off() # reset the graphics device
#plot(x, y) # try plotting again
fviz_nbclust(vehicles_cleaned, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")

# Gap statistic method
set.seed(123)
fviz_nbclust(vehicles_cleaned, kmeans, nstart = 25,  method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")

```

```r

# Silhouette method
install.packages("cluster")
library(cluster)
library(factoextra)
fviz_nbclust(vehicles_cleaned, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")

#compute k-means clustering with k=3
set.seed(123)
final_stat <- kmeans(vehicles_cleaned,3,nstart = 25)
print(final_stat)

#BSS
BSS <- sum(final_stat$size-(colMeans(vehicles_cleaned)-final_stat$centers)^2)
cat("BSS:",BSS,"\n")

#TSS
TSS <- sum((vehicles_cleaned-colMeans(vehicles_cleaned))^2)
cat("TSS:",TSS,"\n")

#WSS
WSS <- sum(final_stat$withinss)
cat("WSS:",WSS,"\n")

#BSS to TSS
ratio_BSS_to_TSS <- BSS/TSS
cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS,"\n")



#Silhouette plot
pam.res2 <- pam(vehicles_cleaned,3,metric="euclidean",stand = FALSE)
fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())

#average Silhouette width score
sil <- silhouette(final_stat$cluster, dist(vehicles_cleaned))
avg_Sil_width <-mean(sil[,3])
cat("average Silhouette width score:",avg_Sil_width,"\n")


```

Part 1 sub task 2,

```r
install.packages("FactoMineR")
install.packages("factoextra")
library(factoextra)
library(FactoMineR)

#apply PCA
pca_data <- PCA(vehicles_cleaned, graph = FALSE)

# Show the eigenvalues and eigenvectors
summary(pca_data)

# Show the scree plot
fviz_eig(pca_data, addlabels = TRUE)

# Show the cumulative percentage of variance explained
eig_val <- get_eigenvalue(pca_data)
eig_val

cumulative_variances <- cumsum(eig_val/sum(eig_val)*100)
cumulative_variances

barplot(cumulative_variances, main = "Cumulative Percentage of Variance Explained", xlab = "Number of Components", ylab = "Cumulative %")

# Choose the PCs that provide at least cumulative score > 92%
num_components <- length(cumulative_variances[cumulative_variances > 92])
print(paste("Number of components needed to explain at least 92% of the variance:", num_components))

# Create a transformed data set
pca_result <- PCA(vehicles_cleaned, ncp = num_components, graph = FALSE)$ind$coord


#determine the number of cluster centers

#NbClust method
library(NbClust)
nb <- NbClust(pca_result, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
print(nb)

#Elbow method
x11() # creates an X11 graphics device
graphics.off() # reset the graphics device
#plot(x, y) # try plotting again
fviz_nbclust(pca_result, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")

# Gap statistic method
set.seed(123)
fviz_nbclust(pca_result, kmeans, nstart = 25,  method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")
```

```r

# Silhouette method
library(cluster)
library(factoextra)
fviz_nbclust(pca_result, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")

#compute k-means clustering with k=3
set.seed(123)
final_stat_pca <- kmeans(pca_result,3,nstart = 25)
print(final_stat_pca)

#BSS
BSS_pca <- sum(final_stat_pca$size-(colMeans(pca_result)-final_stat_pca$centers)^2)
cat("BSS:",BSS_pca,"\n")

#TSS
TSS_pca <- sum((pca_result-colMeans(pca_result))^2)
cat("TSS:",TSS_pca,"\n")

#WSS
WSS_pca <- sum(final_stat_pca$withinss)
cat("WSS:",WSS_pca,"\n")

#BSS to TSS
ratio_BSS_to_TSS_pca <- BSS_pca/TSS_pca
cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS_pca,"\n")

#Silhouette plot
pam.res2 <- pam(pca_result,3,metric="euclidean",stand = FALSE)
fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())

#average Silhouette width score
sil <- silhouette(final_stat_pca$cluster, dist(pca_result))
avg_Sil_width_pca <-mean(sil[,3])
cat("average Silhouette width score:",avg_Sil_width_pca,"\n")

#Calinski-Harabasz Index
install.packages("fpc")
library(fpc)
ch_index <- calinhara(pca_result, final_stat_pca$cluster)
print(ch_index)
barplot(ch_index, main="Calinski-Harabasz Index for K-Means Clustering", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")
plot(ch_index, type="b", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")


#Silhouette plot
library(cluster)
pam.res2 <- pam(pca_result,3,metric="euclidean",stand = FALSE)
fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())

#average Silhouette width score
sil <- silhouette(final_stat_pca$cluster, dist(pca_result))
avg_Sil_width <-mean(sil[,3])
cat("average Silhouette width score:",avg_Sil_width,"\n")
```

Part 2

```r
#TASK 1

install.packages("readxl")
install.packages("neuralnet")
install.packages("ggplot2")
install.packages("dplyr")

library(dplyr)
library(neuralnet)
library(ggplot2)
library(readxl)
setwd("D:/2nd Year/ML_CourseWork")
UOW_data <- read_excel("uow_consumption.xlsx")

# Rename columns
colnames(UOW_data) <- c("date", "time_eighteen", "time_nineteen", "time_twenty")
head(UOW_data)

#apply lag method
UOW_data$lag_1 <- lag(UOW_data$time_twenty, 1)
UOW_data$lag_2 <- lag(UOW_data$time_twenty, 2)
UOW_data$lag_3 <- lag(UOW_data$time_twenty, 3)
UOW_data$lag_4 <- lag(UOW_data$time_twenty, 4)
UOW_data$lag_7 <- lag(UOW_data$time_twenty, 7)
UOW_data <- na.omit(UOW_data)

#TASK 3
#dividing data to testing and training
UOW_train <- UOW_data[1:380,]
UOW_test <- UOW_data[381:nrow(UOW_data),]

#TASK 4
#normalization

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

# Exclude the date column from normalization
UOW_train_normalized <- as.data.frame(lapply(UOW_train[-1], normalize))
UOW_test_normalized <- as.data.frame(lapply(UOW_test[-1], normalize))

# Set the column names of the test_normalized data frame
colnames(UOW_test_normalized) <- colnames(UOW_train_normalized)
```

```r

input_vectors <- list(
  c("lag_1"),
  c("lag_1", "lag_2"),
  c("lag_1", "lag_2", "lag_3"),
  c("lag_1", "lag_2", "lag_3", "lag_4"),
  c("lag_1", "lag_7"),
  c("lag_1", "lag_2", "lag_7"),
  c("lag_1", "lag_2", "lag_3", "lag_7"),
  c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7")
)

build_mlp_model <- function(train_data, test_data, input_vars, hidden_structure) {
  formula <- paste("time_twenty ~", paste(input_vars, collapse = " + "))
  nn <- neuralnet(as.formula(formula), train_data, hidden = hidden_structure)
  test_matrix <- as.matrix(test_data[, input_vars, drop = FALSE])
  colnames(test_matrix) <- colnames(train_data[, input_vars, drop = FALSE])
  predictions <- predict(nn, test_matrix)
  return(list(model = nn, predictions = predictions))
}

models <- list()
for (i in 1:length(input_vectors)) {
  models[[i]] <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, input_vectors[[i]], c(5))
}


#TASK 6
#calculated using the standard statistical indices (RMSE, MAE, MAPE and SMAPE - symmetric MAPE)
calculate_metrics <- function(actual, predicted) {
  rmse <- sqrt(mean((actual - predicted)^2))
  mae <- mean(abs(actual - predicted))
  mape <- mean(abs(actual - predicted) / predicted)
  smape <- mean(abs(actual - predicted) / (abs(actual) + abs(predicted)) * 2) * 100
  return(list(RMSE = rmse, MAE = mae, MAPE = mape, SMAPE = smape))
}

evaluation_metrics <- list()
for (i in 1:length(models)) {
  evaluation_metrics[[i]] <- calculate_metrics(UOW_test_normalized$time_twenty, models[[i]]$predictions)
}


```

```r

#TASK 7
#Create a comparison table of their testing performances
comparison_table <- data.frame(
  Model_Description = c("AR(1)", "AR(2)", "AR(3)", "AR(4)", "AR(1,7)", "AR(2,7)", "AR(3,7)", "AR(4,7)"),
  RMSE = sapply(evaluation_metrics, function(x) x$RMSE),
  MAE = sapply(evaluation_metrics, function(x) x$MAE),
  MAPE = sapply(evaluation_metrics, function(x) x$MAPE),
  sMAPE = sapply(evaluation_metrics, function(x) x$sMAPE)
)

print(comparison_table)

# Add more models with different hidden layer structures and input vectors to create 12-15 models in total

# Efficiency comparison between one-hidden layer and two-hidden layer networks

model_1_hidden <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(5))
model_2_hidden <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(3, 2))

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

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

```

```r

#Task1
# Add the 18th and 19th hour attributes to the input vectors
narx_input_vectors <- list(
  c("lag_1", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "lag_3", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "lag_3", "lag_7", "time_eighteen", "time_nineteen"),
  c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7", "time_eighteen", "time_nineteen")
)

# Build NARX models
narx_models <- list()
for (i in 1:length(narx_input_vectors)) {
  narx_models[[i]] <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, narx_input_vectors[[i]], c(5))
}

# Evaluate NARX models
narx_evaluation_metrics <- list()
for (i in 1:length(narx_models)) {
  narx_evaluation_metrics[[i]] <- calculate_metrics(UOW_test_normalized$time_twenty, narx_models[[i]]$predictions)
}

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

print(narx_comparison_table)

#Task 2

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

best_model_index <- which.min(sapply(evaluation_metrics, function(x) x$RMSE))
best_model <- models[[best_model_index]]
best_model_predictions <- best_model$predictions

min_value <- min(UOW_train$time_twenty)
max_value <- max(UOW_train$time_twenty)
```

```r

denormalized_predictions <- denormalize(best_model_predictions, min_value, max_value)

# Plot the predicted output vs. desired output using a line chart
plot(UOW_test$time_twenty, type = "l", col = "blue", xlab = "Time", ylab = "Hour 20 Consumption", main = "Line Chart of Desired vs. Predicted Output")
lines(denormalized_predictions, col = "red")
legend("topleft", legend = c("Desired Output", "Predicted Output"), col = c("blue", "red"), lty = 1, cex = 0.8)



```

References

Implementation with R language - ScienceDirect. (no date). Available from https://www.sciencedirect.com/science/article/abs/pii/B9780128131275000072?via%3Dihub [Accessed 4 May 2023].

K-Means Clustering: Concepts and Implementation in R for Data Science | by Maria Gulzar | Towards Data Science. (no date). Available from https://towardsdatascience.com/k-means-clustering-concepts-and-implementation-in-r-for-data-science-32cae6a3ceba [Accessed 4 May 2023].

Performance · Advanced R. (no date). Available from http://adv-r.had.co.nz/Performance.html [Accessed 4 May 2023].