

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 to 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.

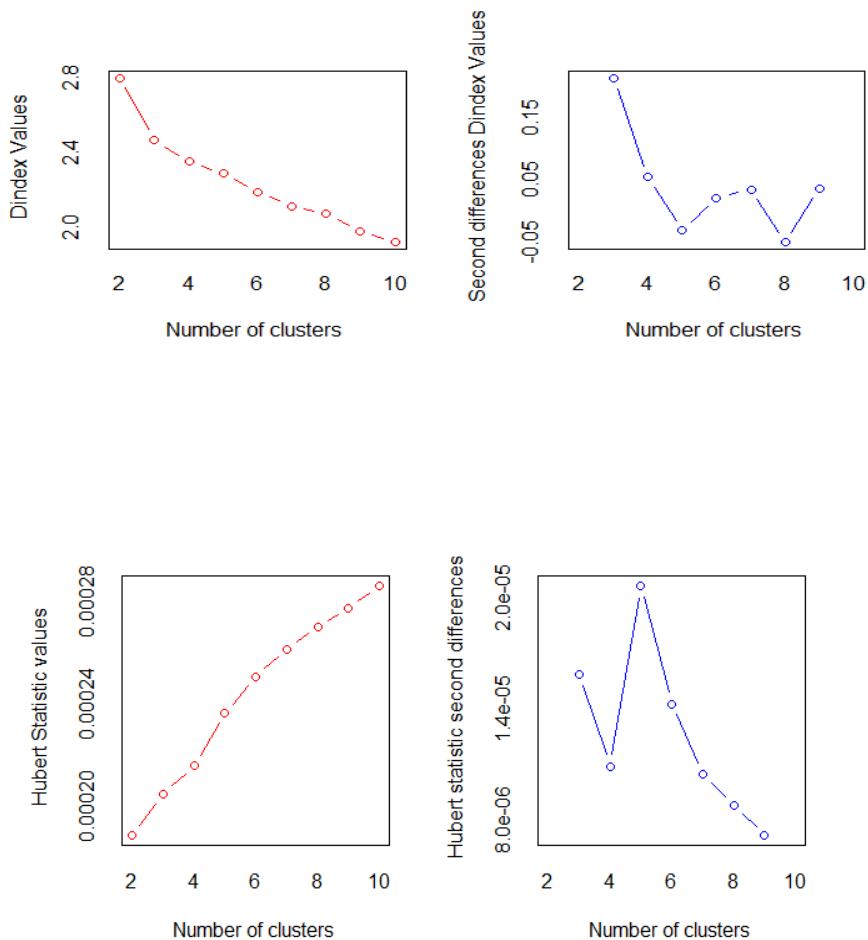
Removal: - To increase the model's accuracy and resilience, outliers can be eliminated. However, it's crucial to remember that eliminating too many outliers can also result in the loss of important data, so a careful balance must be struck. Removing outliers can aid in discovering and fixing problems 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.

```
33 #NbClust method
34 install.packages("NbClust")
35 library(NbClust)
36 nb <- NbClust(vehicles_cleaned, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
37 print(nb)
38
```



```

> nb <- hclust(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 D index
      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      McLain      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      0.8835      26.5924      1.0000
8      0.8426      29.8813      1.0000
9      0.8400      28.3715      1.0000
10     0.8400      28.3715      1.0000

```

```

Console  Terminal  Background Jobs
R 4.2.2 : D:\2nd Year\ML\Course\Work\
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      McLain      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      0.8835      26.5924      1.0000
8      0.8426      29.8813      1.0000
9      0.8400      28.3715      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 3.0000000e+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      McLain      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

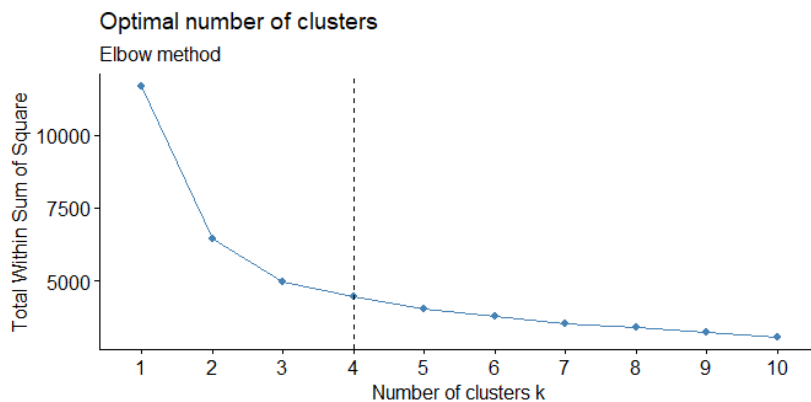
$best.partition
[1] 3 3 2 3 3 3 3 3 3 2 1 2 3 2 2 1 1 3 3 2 3 1 2 2 1 3 3 3 2 3 3 1 2 1 1 1 3 1 3 2 2 3 3 3 1 2 2 1 1 1 3 1 1 3
[64] 2 2 2 3 1 2 3 2 3 2 1 2 2 3 1 3 3 1 1 1 2 3 3 2 2 1 1 2 3 3 2 2 1 1 2 3 3 1 3 2 2 1 1 1 3 1 1 3 2 2 3 1 2 1 3 1 3 3 1 1
[127] 3 2 3 3 2 3 3 2 3 2 3 1 3 1 2 3 3 2 2 1 1 2 2 3 2 3 3 3 1 2 1 3 1 2 3 3 2 3 2 3 3 1 2 1 1 1 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 2 3 1 3 1 3 1 2 1 1 3 3 2 2 1 3 3 3 2 1 1 3 3 1 3 3 1 1 3 3 1 1 2 3 3 1 1 1 3
[253] 3 1 2 3 3 2 3 2 3 1 3 2 3 3 1 2 2 2 2 1 3 2 2 1 1 1 3 1 2 2 1 3 3 1 2 3 3 3 2 2 1 2 1 2 3 3 3 1 1 2 2 3 3 2 1 3 1
[316] 2 3 3 2 2 2 2 3 1 1 1 1 3 3 3 3 1 2 2 2 1 2 2 1 3 1 3 2 1 3 3 2 3 3 2 3 2 3 2 3 1 1 3 3 1 1 3 1 2 3 3 1 1 2 3 1
[379] 3 3 2 3 2 3 2 2 1 1 3 3 2 2 1 3 2 2 2 2 3 3 3 2 1 3 2 3 2 3 2 1 1 2 2 3 1 2 1 2 1 3 1 2 3 3 1 1 2 3 1 2 3 1 2 1 2 2
[442] 3 2 1 1 2 2 1 1 2 2 3 3 1 3 2 2 1 1 2 3 3 1 3 2 2 3 1 3 2 2 1 1 3 2 2 2 3 3 3 3 1 1 3 3 2 1 1 3 1 2 3 2 1 2 2 3 3 3 3
[505] 1 3 2 3 3 1 2 2 2 3 3 1 1 2 2 2 3 2 3 1 1 1 3 1 2 3 3 3 3 2 3 3 2 3 3 1 2 1 1 3 1 3 1 2 2 3 1 2 3 1 2 3 2 3 1 2 1 1 3 1
[568] 3 2 2 1 2 3 1 3 1 2 3 2 1 3 1 1 1 3 2 3 1 3 3 3 2 3 1 2 2 3 2 1 2 3 3 1 2 1 3 3 1 2 1 3 2 1 3 2 1 3 1 3 2 1 3 2 3 2 2
[631] 1 3 2 2 2 2 3 2 3 2 3 2 3 2 3 2 3 2 2 2 3 1 1 2 2 2 3 2 3 2 3 2 3 1 3 1 3 2 3 3 3 1 2 1 1 1 2 2 2 1 3 3 2 1 2 2 1 3 3
[694] 2 2 2 2 2 1 1 2 2 1 2 3 1 3 2 2 2 3 1 2 3 1 3 2 1 1 2 1 2 1 3 1 1 3 2 2 3 1 2 2 2 2 2 2 1 1 3 3 2 2 1 1 3 3 1 1 1 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.

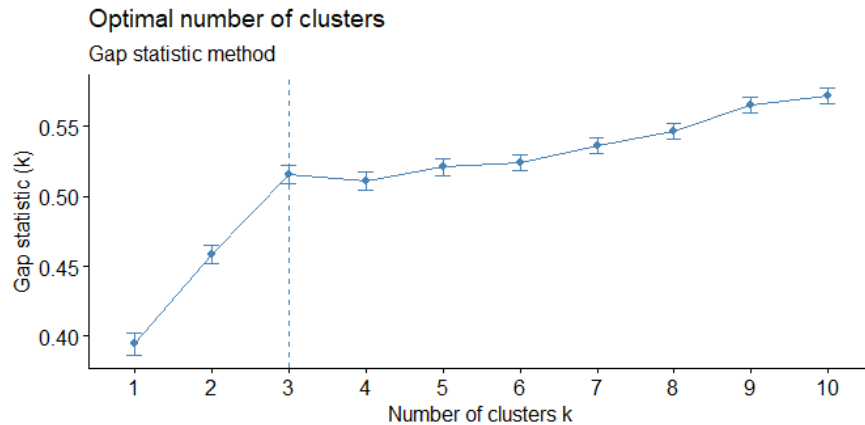
```
39 #Elbow method
40 install.packages("factoextra")
41 library(factoextra)
42 x11() # creates an x11 graphics device
43 graphics.off() # reset the graphics device
44 #plot(x, y) # try plotting again
45 fviz_nbclust(vehicles_cleaned, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")
46
```



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.

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



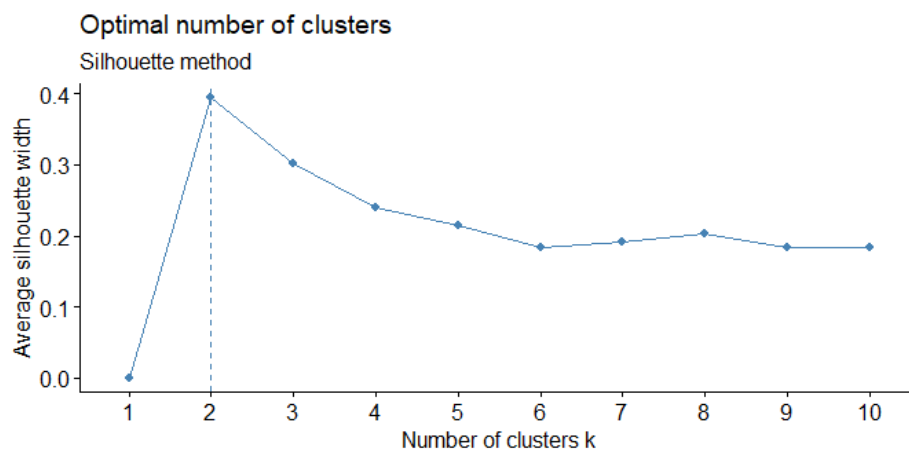
4.silhouette methods.

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.

```

51
52 # Silhouette method
53 install.packages("cluster")
54 library(cluster)
55 library(factoextra)
56 fviz_nbclust(vehicles_cleaned, kmeans, method = "silhouette") + labs(subtitle = "silhouette method")
57

```



(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.

```
58 #compute k-means clustering with k=3
59 set.seed(123)
60 final_stat <- kmeans(vehicles_cleaned,3,nstart = 25)
61 print(final_stat)
62
63 #BSS
64 BSS <- sum(final_stat$size-(colMeans(vehicles_cleaned)-final_stat$centers)^2)
65 cat("BSS:",BSS,"\n")
66
67 #TSS
68 TSS <- sum((vehicles_cleaned-colMeans(vehicles_cleaned))^2)
69 cat("TSS:",TSS,"\n")
70
71 #WSS
72 WSS <- sum(final_stat$withinss)
73 cat("WSS:",WSS,"\n")
74
75 #BSS to TSS
76 ratio_BSS_to_TSS <- BSS/TSS
77 cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS,"\n")
78
79
```

```
> 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 2 1
[78] 2 2 1 2 1 3 2 3 2 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 2 3 3 2 1 3 1 1 2 2 2 1 1 2 3 2 2 3 2 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 1 1 3 2 2 2 3 1 2 1 3 1 2 1 3 1 1 2 3 2 3 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 1 2
[232] 2 2 2 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 1 2 3 3 3 3 1 2 3 1 1 1 2 1 3 3 1 3 2 1 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 2 2 2 2 2 1 2 3 3 1 1 3 3 2 1 2 1 2 2 3 1 2 2 3 2 2 2 3 2 2 3 2 1 1 2 2 2 1 1 2 1 3 2 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 3 1 2 3 2 2 3 2 3 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 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 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 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 1 3 3 1 3 2 2 1 2 1 1 1 2 3 2 1 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 2 3 2 2 3 3 2 3 2 3 2 1 3 3 3 2 1 1 3 3 3 2 3 2 2 3 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 3 1 1 2 2 1 1 1 2 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
>

```

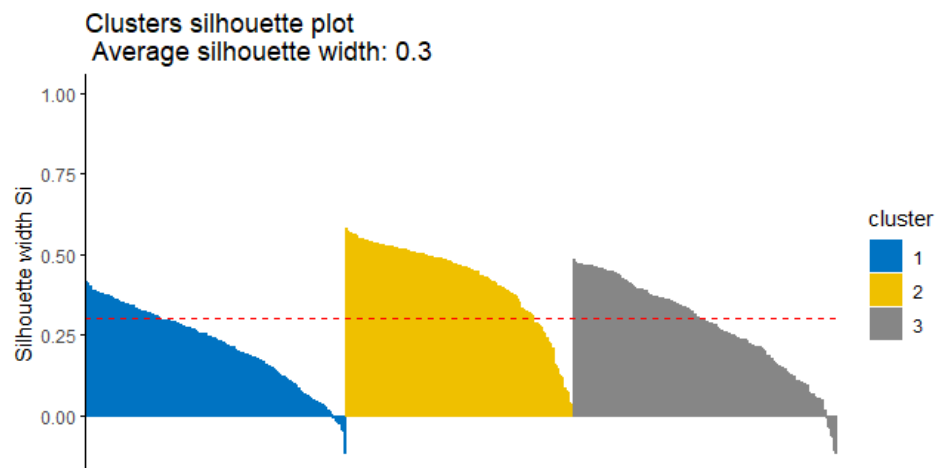
(d)

ytruyruru

```

80
81 #Silhouette plot
82 pam.res2 <- pam(vehicles_cleaned,3,metric="euclidean",stand = FALSE)
83 fviz_silhouette(pam.res2,palette="jco",ggtheme=theme_classic())
84

```




```

> #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
>

```

```

84
85 #average silhouette width score
86 sil <- silhouette(final_stat$cluster, dist(vehicles_cleaned))
87 avg_sil_width <-mean(sil[,3])
88 cat("average silhouette width score:",avg_sil_width,"\n")
89

```

```

> #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.

```

8
9 # Show the eigenvalues and eigenvectors
10 summary(pca_data)
11

```

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

Variables (the 10 first)
Comp | Dim.1 | ctr | cos2 | Dim.2 | ctr | cos2 | Dim.3 | ctr | cos2 |
Circ | 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

```
14
15 # Show the cumulative percentage of variance explained
16 eig_val <- get_eigenvalue(pca_data)
17 eig_val
18
19 cumulative_variances <- cumsum(eig_val/sum(eig_val)*100)
20 cumulative_variances
21
22 barplot(cumulative_variances, main = "Cumulative Percentage of Variance Explained", xlab = "Number of Components", ylab = "Cumulative %")
23
24 # Choose the PCs that provide at least cumulative score > 92%
25 num_components <- length(cumulative_variances[cumulative_variances > 92])
26 print(paste("Number of components needed to explain at least 92% of the variance:", num_components))
27
28 # Create a transformed data set
29 pca_result <- PCA(vehicles_cleaned, ncp = num_components, graph = FALSE)$ind$coord
30

> # 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    0.001987686  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.

```
34 #NbClust method
35 library(NbClust)
36 nb <- NbClust(pca_result, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
37 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 20537470 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
```

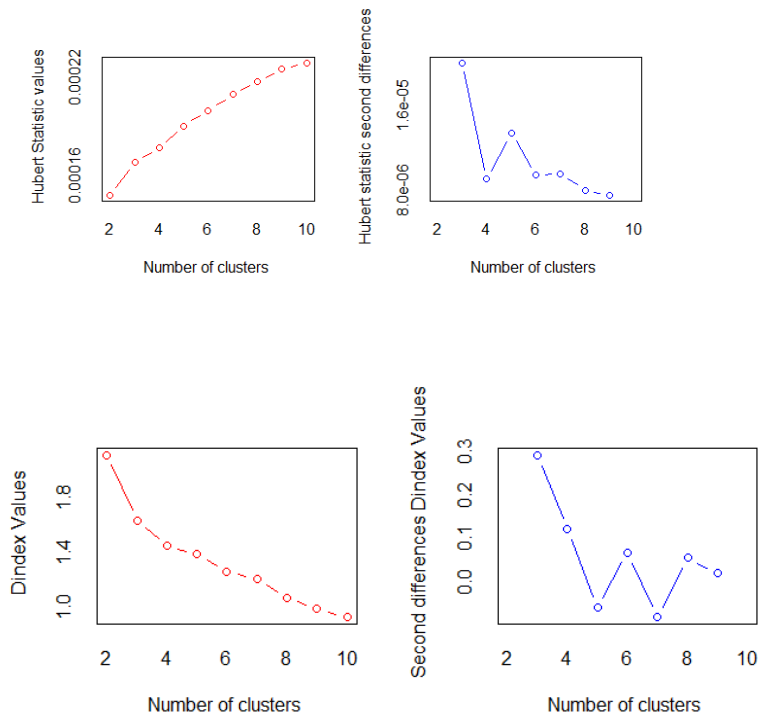
6	0.5012	233.8476	0.8602
7	0.4760	246.5460	1.0000
8	0.4913	211.2269	0.7760
9	0.4156	279.8445	1.0000
9	0.4186	212.4822	1.0000
10	0.4156	254.5320	1.0000

\$Best.nc

Number_clusters	5.0000	KL	3.0000	CH	Hartigan	3.0000	CCC	3.0000	Scott	Marriot	3.0000	4	3.0000	10.0000
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
	Ball	PtBiserial	Frey	McClain	Dunn	Hubert	Sdindex	Index	SdW					
Number_clusters	3.000													
Value_Index	1267.349		0.6616	NA	0.3884	0.0328	0	0.8477	0	0.2532				

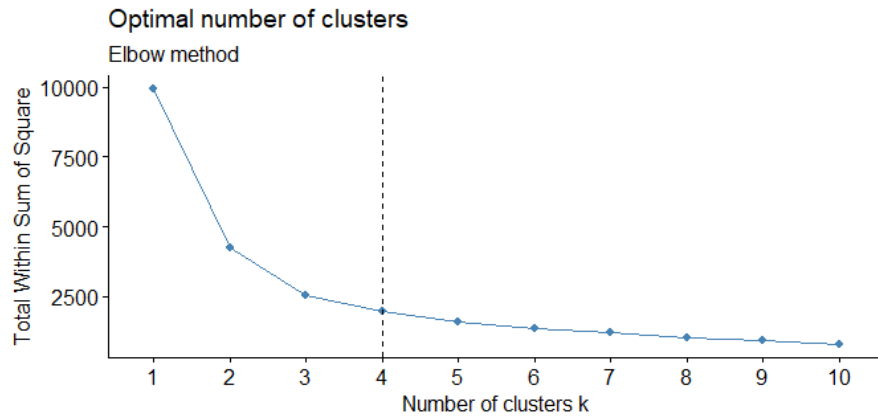
\$Best.partition

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	2	1	3	2	2	1	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	1	3	3	3	1	3	2	2	1	3	1	1	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	1	3	1	3	2	3	3	2	3	2	3	2	3	2	3	1	3	3	1	2	3	1	2	2	1	1	2	2	3	1	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	2	3	2	2	3	3	2	3	1	2	1	1	3	3	3	3	3	1	1	2	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	1	2	3	1	3	3	1	3	2	1	1	3	2	2	1	1	2	3	2	3	2	2	2	2	1	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	1	1	3	3	1	1	3	1	1	3	3	2	3	2	1	3	3	2	2	1	3	3	2	3	3	3	1	3	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	2	2	1	1	1	1	2	2	2	2	3	1	2	2	3	3	3	2	1	2	2	3	2	1	2	3	3	3	1	1	2	2	2	3	1	
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	
1	3	1	2	3	1	2	3	2	3	2	2	3	3	1	1	3	3	1	1	3	3	3	1	2	2	1	1	2	1	2	1	3	3	3	2	1	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	
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	
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	
2	2	1	1	2	2	1	2	2	2	3	3	3	3	3	2	1	3	2	3	2	3	2	3	2	1	1	2	2	3	3	2	1	2	2	3	1	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	
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	
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	
1	3	2	2	1	3	2	3	2	2	3	3	1	3	2	1	3	3	1	1	3	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	
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	
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	
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	1	2	3	2	1	3	1	
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	
3	2	3	1	3	3	3	3	2	3	3	2	1	2	3	2	1	2	1	3	3	3	1	2	3	1	3	2	1	3	3	2	1	3	3	1	3	1	
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	
2	2	3	3	2	2	1	3	2	2	2	2	3	2	3	3	2	3	2	3	2	3	2	3	1	2	2	2	3	1	1	2	2	3	2	3	2	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	
2	3	1	3	1	3	2	3	3	3	3	1	2	1	1	1	2	1	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		
1	2	3	1	3	2	2	3	1	2	3	1	3	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	
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																								



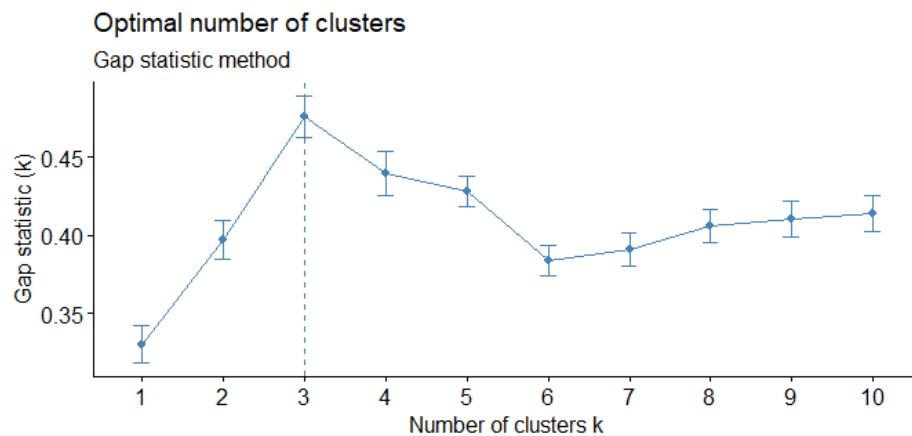
2. Elbow methods.

```
38  
39 #Elbow method  
40 x11() # creates an X11 graphics device  
41 graphics.off() # reset the graphics device  
42 #plot(x, y) # try plotting again  
43 fviz_nbclust(pca_result, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")  
44
```



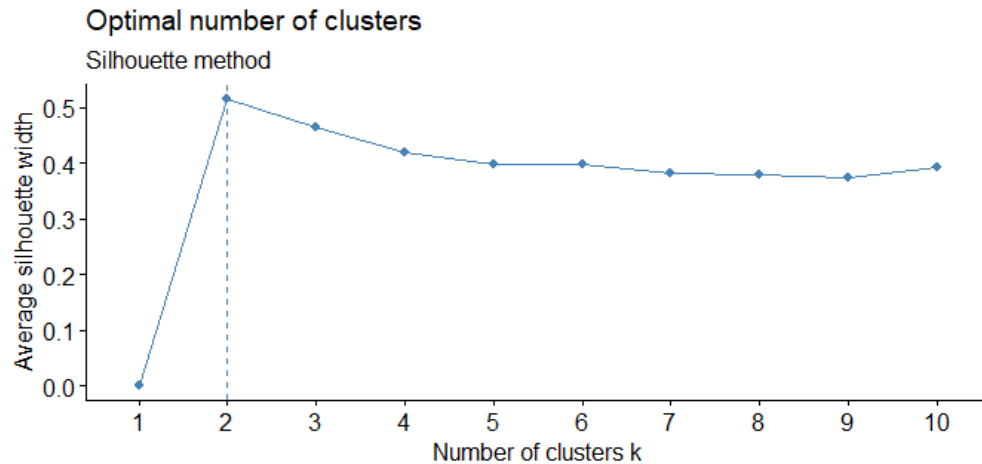
3. Gap statistics methods

```
45 # Gap statistic method  
46 set.seed(123)  
47 fviz_nbclust(pca_result, kmeans, nstart = 25, method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")  
48
```



4.silhouette methods.

```
49 # silhouette method
50 library(cluster)
51 library(factoextra)
52 fviz_nbclust(pca_result, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")
53
54
```



(g)

```
55 #compute k-means clustering with k=3
56 set.seed(123)
57 final_stat_pca <- kmeans(pca_result,3,nstart = 25)
58 print(final_stat_pca)
59
60 #BSS
61 BSS_pca <- sum(final_stat_pca$size-(colMeans(pca_result)-final_stat_pca$centers)^2)
62 cat("BSS:",BSS_pca,"\n")
63
64 #TSS
65 TSS_pca <- sum((pca_result-colMeans(pca_result))^2)
66 cat("TSS:",TSS_pca,"\n")
67
68 #WSS
69 WSS_pca <- sum(final_stat_pca$withinss)
70 cat("WSS:",WSS_pca,"\n")
71
72 #BSS to TSS
73 ratio_BSS_to_TSS_pca <- BSS_pca/TSS_pca
74 cat("ratio_BSS_to_TSS:",ratio_BSS_to_TSS_pca,"\n")
75
```

```

> #compute k-means clustering with k=3
> set.seed(223)
> final_stat_pca <- kmeans(pca_result,3,ntstart = 25)
> print(final_stat_pca)
K-means clustering with 3 clusters of sizes 215, 311, 230

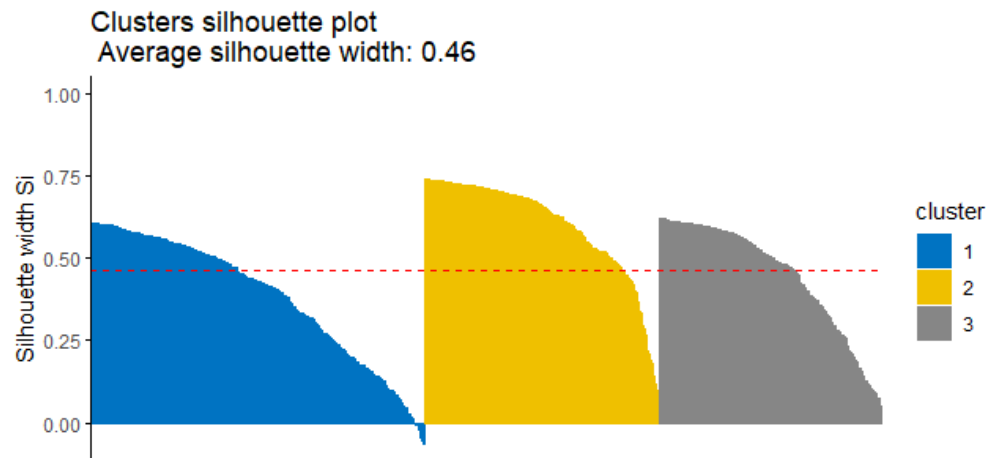
Cluster means:
      dfm.1      dfm.2
1 -2.953007 -1.7239364
2 -1.006585  1.4707448
3  4.121092 -0.3771564

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
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
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
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
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
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
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
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
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
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
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
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
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
757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800
801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860
861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920
921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980
981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040
1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100
1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137 1138 1139 1140 1141 1142 1143 1144 1145 1146 1147 1148 1149 1150 1151 1152 1153 1154 1155 1156 1157 1158 1159 1160
1161 1162 1163 1164 1165 1166 1167 1168 1169 1170 1171 1172 1173 1174 1175 1176 1177 1178 1179 1180 1181 1182 1183 1184 1185 1186 1187 1188 1189 1190 1191 1192 1193 1194 1195 1196 1197 1198 1199 1200 1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220
1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280
1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299 1300 1301 1302 1303 1304 1305 1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337 1338 1339 1340 1341 1342 1343 1344 1345 1346 1347 1348 1349 1350 1351 1352 1353 1354 1355 1356 1357 1358 1359 1360
1361 1362 1363 1364 1365 1366 1367 1368 1369 1370 1371 1372 1373 1374 1375 1376 1377 1378 1379 1380 1381 1382 1383 1384 1385 1386 1387 1388 1389 1390 1391 1392 1393 1394 1395 1396 1397 1398 1399 1400 1401 1402 1403 1404 1405 1406 1407 1408 1409 1410 1411 1412 1413 1414 1415 1416 1417 1418 1419 1420
1421 1422 1423 1424 1425 1426 1427 1428 1429 1430 1431 1432 1433 1434 1435 1436 1437 1438 1439 1440 1441 1442 1443 1444 1445 1446 1447 1448 1449 1450 1451 1452 1453 1454 1455 1456 1457 1458 1459 1460 1461 1462 1463 1464 1465 1466 1467 1468 1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480
1481 1482 1483 1484 1485 1486 1487 1488 1489 1490 1491 1492 1493 1494 1495 1496 1497 1498 1499 1500 1501 1502 1503 1504 1505 1506 1507 1508 1509 1510 1511 1512 1513 1514 1515 1516 1517 1518 1519 1520 1521 1522 1523 1524 1525 1526 1527 1528 1529 1530 1531 1532 1533 1534 1535 1536 1537 1538 1539 1540
1541 1542 1543 1544 1545 1546 1547 1548 1549 1550 1551 1552 1553 1554 1555 1556 1557 1558 1559 1560 1561 1562 1563 1564 1565 1566 1567 1568 1569 1570 1571 1572 1573 1574 1575 1576 1577 1578 1579 1580 1581 1582 1583 1584 1585 1586 1587 1588 1589 1590 1591 1592 1593 1594 1595 1596 1597 1598 1599 1600
1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1630 1631 1632 1633 1634 1635 1636 1637 1638 1639 1640 1641 1642 1643 1644 1645 1646 1647 1648 1649 1650 1651 1652 1653 1654 1655 1656 1657 1658 1659 1660
1661 1662 1663 1664 1665 1666 1667 1668 1669 1670 1671 1672 1673 1674 1675 1676 1677 1678 1679 1680 1681 1682 1683 1684 1685 1686 1687 1688 1689 1690 1691 1692 1693 1694 1695 1696 1697 1698 1699 1700 1701 1702 1703 1704 1705 1706 1707 1708 1709 1710 1711 1712 1713 1714 1715 1716 1717 1718 1719 1720
1721 1722 1723 1724 1725 1726 1727 1728 1729 1730 1731 1732 1733 1734 1735 1736 1737 1738 1739 1740 1741 1742 1743 1744 1745 1746 1747 1748 1749 1750 1751 1752 1753 1754 1755 1756 1757 1758 1759 1760 1761 1762 1763 1764 1765 1766 1767 1768 1769 1770 1771 1772 1773 1774 1775 1776 1777 1778 1779 1780
1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1820 1821 1822 1823 1824 1825 1826 1827 1828 1829 1830 1831 1832 1833 1834 1835 1836 1837 1838 1839 1840 1841 1842 1843 1844 1845 1846 1847 1848 1849 1850 1851 1852 1853 1854 1855 1856 1857 1858 1859 1860
1861 1862 1863 1864 1865 1866 1867 1868 1869 1870 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917 1918 1919 1920
1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980
1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022 2023 2024 2025 2026 2027 2028 2029 2030 2031 2032 2033 2034 2035 2036 2037 2038 2039 2040
2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100
2101 2102 2103 2104 2105 2106 2107 2108 2109 2110 2111 2112 2113 2114 2115 2116 2117 2118 2119 2120 2121 2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132 2133 2134 2135 2136 2137 2138 2139 2140 2141 2142 2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155 2156 2157 2158 2159 2160
2161 2162 2163 2164 2165 2166 2167 2168 2169 2170 2171 2172 2173 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 2184 2185 2186 2187 2188 2189 2190 2191 2192 2193 2194 2195 2196 2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214 2215 2216 2217 2218 2219 2220
2221 2222 2223 2224 2225 2226 2227 2228 2229 2230 2231 2232 2233 2234 2235 2236 2237 2238 2239 2240 2241 2242 2243 2244 2245 2246 2247 2248 2249 2250 2251 2252 2253 2254 2255 2256 2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280
2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2296 2297 2298 2299 2300 2301 2302 2303 2304 2305 2306 2307 2308 2309 2310 2311 2312 2313 2314 2315 2316 2317 2318 2319 2320 2321 2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332 2333 2334 2335 2336 2337 2338 2339 2340
2341 2342 2343 2344 2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2390 2391 2392 2393 2394 2395 2396 2397 2398 2399 2400
2401 2402 2403 2404 2405 2406 2407 2408 2409 2410 2411 2412 2413 2414 2415 2416 2417 2418 2419 2420 2421 2422 2423 2424 2425 2426 2427 2428 2429 2430 2431 2432 2433 2434 2435 2436 2437 2438 2439 2440 2441 2442 2443 2444 2445 2446 2447 2448 2449 2450 2451 2452 2453 2454 2455 2456 2457 2458 2459 2460
2461 2462 2463 2464 2465 2466 2467 2468 2469 2470 2471 2472 2473 2474 2475 2476 2477 2478 2479 2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501 2502 2503 2504 2505 2506 2507 2508 2509 2510 2511 2512 2513 2514 2515 2516 2517 2518 2519 2520
2521 2522 2523 2524 2525 2526 2527 2528 2529 2530 2531 2532 2533 2534 2535 2536 2537 2538 2539 2540 2541 2542 2543 2544 2545 2546 2547 2548 2549 2550 2551 2552 2553 2554 2555 2556 2557 2558 2559 2560 2561 2562 2563 2564 2565 2566 2567 2568 2569 2570 2571 2572 2573 2574 2575 2576 2577 2578 2579 2580 2581 2582 2583 2584 2585 2586 2587 2588 2589 2590 2591 2592 2593 2594 2595 2596 2597 2598 2599 2600
2601 2602 2603 2604 2605 2606 2607 2608 2609 2610 2611 2612 2613 2614 2615 2616 2617 2618 2619 2620 2621 2622 2623 2624 2625 2626 2627 2628 2629 2630 2631 2632 2633 2634 2635 2636 2637 2638 2639 2640 2641 2642 2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654 2655 2656 2657
```

```

> #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

```

87 library(fpc)
88 ch_index <- calinhara(pca_result, final_stat_pca$cluster)
89 print(ch_index)
90 barplot(ch_index, main="Calinski-Harabasz Index for K-Means Clustering", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")
91 plot(ch_index, type="b", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")
92

```

```

> 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)

```
8 library(dplyr)
9 library(neuralnet)
10 library(ggplot2)
11 library(readxl)
12 setwd("D:/2nd Year/ML_Coursework")
13 uow_data <- read_excel("uow_consumption.xlsx")
14
15 # Rename columns
16 colnames(uow_data) <- c("date", "time_eighteen", "time_nineteen", "time_twenty")
17 head(uow_data)
18
```

```

> # Rename columns
> colnames(uow_data) <- c("date", "time_eighteen", "time_nineteen", "time_twenty")
> head(uow_data)
# A tibble: 6 x 4
  date                time_eighteen time_nineteen time_twenty
<dtm>                <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
> |

```

```

19 #apply lag method
20 uow_data$lag_1 <- lag(uow_data$time_twenty, 1)
21 uow_data$lag_2 <- lag(uow_data$time_twenty, 2)
22 uow_data$lag_3 <- lag(uow_data$time_twenty, 3)
23 uow_data$lag_4 <- lag(uow_data$time_twenty, 4)
24 uow_data$lag_7 <- lag(uow_data$time_twenty, 7)
25 uow_data <- na.omit(uow_data)
26
27 #TASK 3
28 #dividing data to testing and training
29 uow_train <- uow_data[1:380,]
30 uow_test <- uow_data[381:nrow(uow_data),]
31

```

(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.

```

32 #TASK 4
33 #normalization
34
35 normalize <- function(x) {
36   return((x - min(x)) / (max(x) - min(x)))
37 }
38
39 # Exclude the date column from normalization
40 uow_train_normalized <- as.data.frame(lapply(uow_train[-1], normalize))
41 uow_test_normalized <- as.data.frame(lapply(uow_test[-1], normalize))
42
43 # Set the column names of the test_normalized data frame
44 colnames(uow_test_normalized) <- colnames(uow_train_normalized)
45

```

(d)

```
47
48 input_vectors <- list(
49   c("lag_1"),
50   c("lag_1", "lag_2"),
51   c("lag_1", "lag_2", "lag_3"),
52   c("lag_1", "lag_2", "lag_3", "lag_4"),
53   c("lag_1", "lag_7"),
54   c("lag_1", "lag_2", "lag_7"),
55   c("lag_1", "lag_2", "lag_3", "lag_7"),
56   c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7")
57 )
58
59 build_mlp_model <- function(train_data, test_data, input_vars, hidden_structure) {
60   formula <- paste("time_twenty ~", paste(input_vars, collapse = " + "))
61   nn <- neuralnet(as.formula(formula), train_data, hidden = hidden_structure)
62   test_matrix <- as.matrix(test_data[, input_vars, drop = FALSE])
63   colnames(test_matrix) <- colnames(train_data[, input_vars, drop = FALSE])
64   predictions <- predict(nn, test_matrix)
65   return(list(model = nn, predictions = predictions))
66 }
67
68 models <- list()
69 for (i in 1:length(input_vectors)) {
70   models[[i]] <- build_mlp_model(uow_train_normalized, uow_test_normalized, input_vectors[[i]], c(5))
71 }
72
73
```

(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.

```
76 calculate_metrics <- function(actual, predicted) {
77   rmse <- sqrt(mean((actual - predicted)^2))
78   mae <- mean(abs(actual - predicted))
79   mape <- mean(abs(actual - predicted) / predicted)
80   smape <- mean(abs(actual - predicted) / (abs(actual) + abs(predicted)) * 2) * 100
81   return(list(RMSE = rmse, MAE = mae, MAPE = mape, sMAPE = smape))
82 }
83
84 evaluation_metrics <- list()
85 for (i in 1:length(models)) {
86   evaluation_metrics[[i]] <- calculate_metrics(uow_test_normalized$time_twenty, models[[i]]$predictions)
87 }
88
```

(d)

```
91 #Create a comparison table of their testing performances
92 comparison_table <- data.frame(
93   Model_Description = c("AR(1)", "AR(2)", "AR(3)", "AR(4)", "AR(1,7)", "AR(2,7)", "AR(3,7)", "AR(4,7)"),
94   RMSE = sapply(evaluation_metrics, function(x) x$RMSE),
95   MAE = sapply(evaluation_metrics, function(x) x$MAE),
96   MAPE = sapply(evaluation_metrics, function(x) x$MAPE),
97   SMAPE = sapply(evaluation_metrics, function(x) x$SMAPE)
98 )
99
100 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.

```
103
104 # Efficiency comparison between one-hidden layer and two-hidden layer networks
105
106 model_1_hidden <- build_mlp_model(uow_train_normalized, uow_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(5))
107 model_2_hidden <- build_mlp_model(uow_train_normalized, uow_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(3, 2))
108
109 # Check the total number of weight parameters per network
110 num_weights_1_hidden <- sum(sapply(model_1_hidden$model$weights, length))
111 num_weights_2_hidden <- sum(sapply(model_2_hidden$model$weights, length))
112
113 cat("Total number of weight parameters for the one-hidden layer network:", num_weights_1_hidden, "\n")
114 cat("Total number of weight parameters for the two-hidden layer network:", num_weights_2_hidden, "\n")
115
```

```
> 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)

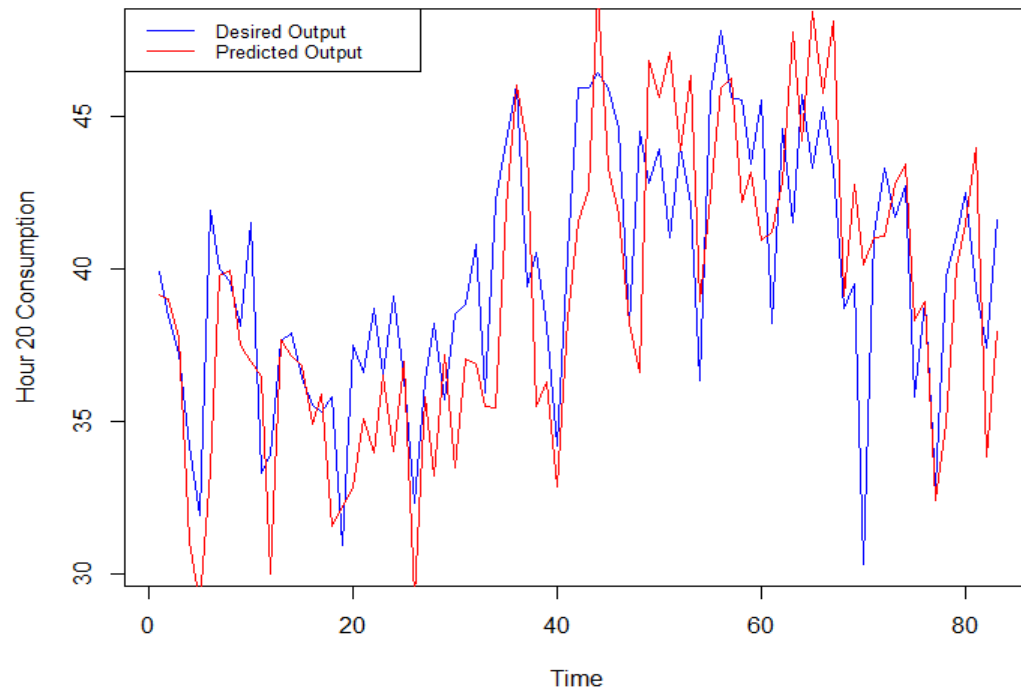
```
117 #Task1
118 # Add the 18th and 19th hour attributes to the input vectors
119 narx_input_vectors <- list(
120   c("lag_1", "time_eighteen", "time_nineteen"),
121   c("lag_1", "lag_2", "time_eighteen", "time_nineteen"),
122   c("lag_1", "lag_2", "lag_3", "time_eighteen", "time_nineteen"),
123   c("lag_1", "lag_2", "lag_3", "lag_7", "time_eighteen", "time_nineteen"),
124   c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7", "time_eighteen", "time_nineteen")
125 )
126
127 # Build NARX models
128 narx_models <- list()
129 for (i in 1:length(narx_input_vectors)) {
130   narx_models[[i]] <- build_nlp_model(uow_train_normalized, uow_test_normalized, narx_input_vectors[[i]], c(5))
131 }
132
133 # Evaluate NARX models
134 narx_evaluation_metrics <- list()
135 for (i in 1:length(narx_models)) {
136   narx_evaluation_metrics[[i]] <- calculate_metrics(uow_test_normalized$time_twenty, narx_models[[i]]$predictions)
137 }
138
139 # Create a comparison table for NARX models
140 narx_comparison_table <- data.frame(
141   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)"),
142   RMSE = sapply(narx_evaluation_metrics, function(x) x$RMSE),
143   MAE = sapply(narx_evaluation_metrics, function(x) x$MAE),
144   MAPE = sapply(narx_evaluation_metrics, function(x) x$MAPE),
145   SMAPE = sapply(narx_evaluation_metrics, function(x) x$SMAPE)
146 )
147
148 print(narx_comparison_table)
149
150
```

```
> 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)

```
152 # Denormalize the predictions
153 denormalize <- function(x, min_value, max_value) {
154   return(x * (max_value - min_value) + min_value)
155 }
156
157 best_model_index <- which.min(sapply(evaluation_metrics, function(x) x$RMSE))
158 best_model <- models[[best_model_index]]
159 best_model_predictions <- best_model$predictions
160
161 min_value <- min(uow_train$time_twenty)
162 max_value <- max(uow_train$time_twenty)
163
164 denormalized_predictions <- denormalize(best_model_predictions, min_value, max_value)
165
166 # Plot the predicted output vs. desired output using a line chart
167 plot(uow_test$time_twenty, type = "l", col = "blue", xlab = "Time", ylab = "Hour 20 Consumption", main = "Line chart of Desired vs. Predicted output")
168 lines(denormalized_predictions, col = "red")
169 legend("topleft", legend = c("Desired output", "Predicted output"), col = c("blue", "red"), lty = 1, cex = 0.8)
170
171
172
```

Line Chart of Desired vs. Predicted Output



Appendix

Part 1 sub task 1,

```
1 #load the data set
2 install.packages("readxl")
3 library(readxl)
4 setwd("D:/2nd Year/ML_Coursework")
5 vehicles <- read_excel("vehicles.xlsx")
6
7 #define 18 attributes
8 install.packages("dplyr")
9 library(dplyr)
10 vehicles_subset <- select(vehicles, Comp:Holl.Ra)
11
12 #Scaling.
13 vehicles_scaled<-scale(vehicles_subset)
14
15 #set outlines using IQR method.
16 out1<- apply(vehicles_scaled,2,quantile,probs=0.25,na.rm = TRUE)
17 out3<- apply(vehicles_scaled,2,quantile,probs=0.75,na.rm =TRUE)
18 IQR<- out3 - out1
19
20 #identifies outliers.
21 outliers<-apply(vehicles_scaled,2,function(x){
22   lower <- out1-1.5*IQR
23   upper <- out3+1.5*IQR
24   x < lower | x > upper
25 })
26
27 #remove outliers.
28 vehicles_cleaned <- vehicles_scaled
29 vehicles_cleaned[outliers] <- NA
30 vehicles_cleaned <- na.omit(vehicles_cleaned)
31
32 #determine the number of cluster centers
33 #NbClust method
34 install.packages("NbClust")
35 library(NbClust)
36 nb <- NbClust(vehicles_cleaned, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
37 print(nb)
38
39 #Elbow method
40 install.packages("factoextra")
41 library(factoextra)
42 x11() # creates an X11 graphics device
43 graphics.off() # reset the graphics device
44 #plot(x, y) # try plotting again
45 fviz_nbclust(vehicles_cleaned, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")
46
47 # Gap statistic method
48 set.seed(123)
49 fviz_nbclust(vehicles_cleaned, kmeans, nstart = 25, method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")
50
```

```

51
52 # silhouette method
53 install.packages("cluster")
54 library(cluster)
55 library(factoextra)
56 fviz_nbclust(vehicles_cleaned, kmeans, method = "silhouette") + labs(subtitle = "Silhouette method")
57
58 #compute k-means clustering with k=3
59 set.seed(123)
60 final_stat <- kmeans(vehicles_cleaned, 3, nstart = 25)
61 print(final_stat)
62
63 #BSS
64 BSS <- sum(final_stat$size - (colMeans(vehicles_cleaned) - final_stat$centers)^2)
65 cat("BSS:", BSS, "\n")
66
67 #TSS
68 TSS <- sum((vehicles_cleaned - colMeans(vehicles_cleaned))^2)
69 cat("TSS:", TSS, "\n")
70
71 #WSS
72 WSS <- sum(final_stat$withinss)
73 cat("WSS:", WSS, "\n")
74
75 #BSS to TSS
76 ratio_BSS_to_TSS <- BSS/TSS
77 cat("ratio_BSS_to_TSS:", ratio_BSS_to_TSS, "\n")
78
79
80
81 #silhouette plot
82 pam.res2 <- pam(vehicles_cleaned, 3, metric="euclidean", stand = FALSE)
83 fviz_silhouette(pam.res2, palette="jco", ggtheme=theme_classic())
84
85 #average silhouette width score
86 sil <- silhouette(final_stat$cluster, dist(vehicles_cleaned))
87 avg_sil_width <- mean(sil[, 3])
88 cat("average silhouette width score:", avg_sil_width, "\n")
89
90

```


Part 1 sub task 2,

```
1 install.packages("FactoMineR")
2 install.packages("factoextra")
3 library(factoextra)
4 library(FactoMineR)
5
6 #apply PCA
7 pca_data <- PCA(vehicles_cleaned, graph = FALSE)
8
9 # Show the eigenvalues and eigenvectors
10 summary(pca_data)
11
12 # Show the scree plot
13 fviz_eig(pca_data, addlabels = TRUE)
14
15 # Show the cumulative percentage of variance explained
16 eig_val <- get_eigenvalue(pca_data)
17 eig_val
18
19 cumulative_variances <- cumsum(eig_val/sum(eig_val)*100)
20 cumulative_variances
21
22 barplot(cumulative_variances, main = "Cumulative Percentage of Variance Explained", xlab = "Number of Components", ylab = "Cumulative %")
23
24 # Choose the PCs that provide at least cumulative score > 92%
25 num_components <- length(cumulative_variances[cumulative_variances > 92])
26 print(paste("Number of components needed to explain at least 92% of the variance:", num_components))
27
28 # Create a transformed data set
29 pca_result <- PCA(vehicles_cleaned, ncp = num_components, graph = FALSE)$ind$coord
30
31
32 #determine the number of cluster centers
33
34 #Nbclust method
35 library(NbClust)
36 nb <- NbClust(pca_result, distance = "euclidean", min.nc = 2, max.nc = 10, method = "kmeans")
37 print(nb)
38
39 #Elbow method
40 x11() # creates an X11 graphics device
41 graphics.off() # reset the graphics device
42 #plot(x, y) # try plotting again
43 fviz_nbclust(pca_result, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2) + labs(subtitle = "Elbow method")
44
45 # Gap statistic method
46 set.seed(123)
47 fviz_nbclust(pca_result, kmeans, nstart = 25, method = "gap_stat", nboot = 50) + labs(subtitle = "Gap statistic method")
48
49
```

```

49 # silhouette method
50 library(cluster)
51 library(factoextra)
52 fviz_nbclust(pca_result, kmeans, method = "silhouette") + labs(subtitle = "silhouette method")
53
54 #compute k-means clustering with k=3
55 set.seed(123)
56 final_stat_pca <- kmeans(pca_result, 3, nstart = 25)
57 print(final_stat_pca)
58
59 #BSS
60 BSS_pca <- sum(final_stat_pca$size - (colMeans(pca_result) - final_stat_pca$centers)^2)
61 cat("BSS:", BSS_pca, "\n")
62
63 #TSS
64 TSS_pca <- sum((pca_result - colMeans(pca_result))^2)
65 cat("TSS:", TSS_pca, "\n")
66
67 #WSS
68 WSS_pca <- sum(final_stat_pca$withinss)
69 cat("WSS:", WSS_pca, "\n")
70
71 #BSS to TSS
72 ratio_BSS_to_TSS_pca <- BSS_pca/TSS_pca
73 cat("ratio_BSS_to_TSS:", ratio_BSS_to_TSS_pca, "\n")
74
75 #silhouette plot
76 pam.res2 <- pam(pca_result, 3, metric="euclidean", stand = FALSE)
77 fviz_silhouette(pam.res2, palette="jco", ggtheme=theme_classic())
78
79 #average silhouette width score
80 sil <- silhouette(final_stat_pca$cluster, dist(pca_result))
81 avg_sil_width_pca <- mean(sil[, 3])
82 cat("average silhouette width score:", avg_sil_width_pca, "\n")
83
84 #Calinski-Harabasz Index
85 install.packages("fpc")
86 library(fpc)
87 ch_index <- calinhara(pca_result, final_stat_pca$cluster)
88 print(ch_index)
89 barplot(ch_index, main="Calinski-Harabasz Index for K-Means Clustering", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")
90 plot(ch_index, type="b", xlab="Number of Clusters", ylab="Calinski-Harabasz Index")
91
92 #silhouette plot
93 library(cluster)
94 pam.res2 <- pam(pca_result, 3, metric="euclidean", stand = FALSE)
95 fviz_silhouette(pam.res2, palette="jco", ggtheme=theme_classic())
96
97 #average silhouette width score
98 sil <- silhouette(final_stat_pca$cluster, dist(pca_result))
99 avg_sil_width <- mean(sil[, 3])
100 cat("average silhouette width score:", avg_sil_width, "\n")
101
102 ...

```

Part 2

```
1 #TASK 1
2
3 install.packages("readxl")
4 install.packages("neuralnet")
5 install.packages("ggplot2")
6 install.packages("dplyr")
7
8 library(dplyr)
9 library(neuralnet)
10 library(ggplot2)
11 library(readxl)
12 setwd("b:/2nd Year/ML_Coursework")
13 UOW_data <- read_excel("uow_consumption.xlsx")
14
15 # Rename columns
16 colnames(UOW_data) <- c("date", "time_eighteen", "time_nineteen", "time_twenty")
17 head(UOW_data)
18
19 #apply lag method
20 UOW_data$lag_1 <- lag(UOW_data$time_twenty, 1)
21 UOW_data$lag_2 <- lag(UOW_data$time_twenty, 2)
22 UOW_data$lag_3 <- lag(UOW_data$time_twenty, 3)
23 UOW_data$lag_4 <- lag(UOW_data$time_twenty, 4)
24 UOW_data$lag_7 <- lag(UOW_data$time_twenty, 7)
25 UOW_data <- na.omit(UOW_data)
26
27 #TASK 3
28 #dividing data to testing and training
29 UOW_train <- UOW_data[1:380,]
30 UOW_test <- UOW_data[381:nrow(UOW_data),]
31
32 #TASK 4
33 #normalization
34
35 normalize <- function(x) {
36   return((x - min(x)) / (max(x) - min(x)))
37 }
38
39 # Exclude the date column from normalization
40 UOW_train_normalized <- as.data.frame(lapply(UOW_train[-1], normalize))
41 UOW_test_normalized <- as.data.frame(lapply(UOW_test[-1], normalize))
42
43 # Set the column names of the test_normalized data frame
44 colnames(UOW_test_normalized) <- colnames(UOW_train_normalized)
45
46
47 input_vectors <- list(
48   c("lag_1"),
49   c("lag_1", "lag_2"),
50   c("lag_1", "lag_2", "lag_3"),
51   c("lag_1", "lag_2", "lag_3", "lag_4"),
52   c("lag_1", "lag_7"),
53   c("lag_1", "lag_2", "lag_7"),
54   c("lag_1", "lag_2", "lag_3", "lag_7"),
55   c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7")
56 )
57
58
59 build_mlp_model <- function(train_data, test_data, input_vars, hidden_structure) {
60   formula <- paste("time_twenty ~", paste(input_vars, collapse = " + "))
61   nn <- neuralnet(as.formula(formula), train_data, hidden = hidden_structure)
62   test_matrix <- as.matrix(test_data[, input_vars, drop = FALSE])
63   colnames(test_matrix) <- colnames(train_data[, input_vars, drop = FALSE])
64   predictions <- predict(nn, test_matrix)
65   return(list(model = nn, predictions = predictions))
66 }
67
68 models <- list()
69 for (i in 1:length(input_vectors)) {
70   models[[i]] <- build_mlp_model(UOW_train_normalized, UOW_test_normalized, input_vectors[[i]], c(5))
71 }
72
73
74 #TASK 6
75 #calculated using the standard statistical indices (RMSE, MAE, MAPE and SMAPE - symmetric MAPE)
76 calculate_metrics <- function(actual, predicted) {
77   rmse <- sqrt(mean((actual - predicted)^2))
78   mae <- mean(abs(actual - predicted))
79   mape <- mean(abs(actual - predicted) / predicted)
80   smape <- mean(abs(actual - predicted) / (abs(actual) + abs(predicted)) * 2) * 100
81   return(list(RMSE = rmse, MAE = mae, MAPE = mape, SMAPE = smape))
82 }
83
84 evaluation_metrics <- list()
85 for (i in 1:length(models)) {
86   evaluation_metrics[[i]] <- calculate_metrics(UOW_test_normalized$time_twenty, models[[i]]$predictions)
87 }
88
89
```

```

89
90 #TASK 7
91 #Create a comparison table of their testing performances
92 comparison_table <- data.frame(
93   Model_Description = c("AR(1)", "AR(2)", "AR(3)", "AR(4)", "AR(1,7)", "AR(2,7)", "AR(3,7)", "AR(4,7)"),
94   RMSE = sapply(evaluation_metrics, function(x) x$RMSE),
95   MAE = sapply(evaluation_metrics, function(x) x$MAE),
96   MAPE = sapply(evaluation_metrics, function(x) x$MAPE),
97   SMAPE = sapply(evaluation_metrics, function(x) x$SMAPE)
98 )
99
100 print(comparison_table)
101
102 # Add more models with different hidden layer structures and input vectors to create 12-15 models in total
103
104 # Efficiency comparison between one-hidden layer and two-hidden layer networks
105
106 model_1_hidden <- build_mlp_model(uow_train_normalized, uow_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(5))
107 model_2_hidden <- build_mlp_model(uow_train_normalized, uow_test_normalized, c("lag_1", "lag_2", "lag_3", "lag_7"), c(3, 2))
108
109 # Check the total number of weight parameters per network
110 num_weights_1_hidden <- sum(sapply(model_1_hidden$model$weights, length))
111 num_weights_2_hidden <- sum(sapply(model_2_hidden$model$weights, length))
112
113 cat("Total number of weight parameters for the one-hidden layer network:", num_weights_1_hidden, "\n")
114 cat("Total number of weight parameters for the two-hidden layer network:", num_weights_2_hidden, "\n")
115

```

```

117
118 #Task1
119 # Add the 18th and 19th hour attributes to the input vectors
120 narx_input_vectors <- list(
121   c("lag_1", "time_eighteen", "time_nineteen"),
122   c("lag_1", "lag_2", "time_eighteen", "time_nineteen"),
123   c("lag_1", "lag_2", "lag_3", "time_eighteen", "time_nineteen"),
124   c("lag_1", "lag_2", "lag_3", "lag_7", "time_eighteen", "time_nineteen"),
125   c("lag_1", "lag_2", "lag_3", "lag_4", "lag_7", "time_eighteen", "time_nineteen")
126 )
127
128 # Build NARX models
129 narx_models <- list()
130 for (i in 1:length(narx_input_vectors)) {
131   narx_models[[i]] <- build_mlp_model(uow_train_normalized, uow_test_normalized, narx_input_vectors[[i]], c(5))
132 }
133
134 # Evaluate NARX models
135 narx_evaluation_metrics <- list()
136 for (i in 1:length(narx_models)) {
137   narx_evaluation_metrics[[i]] <- calculate_metrics(uow_test_normalized$time_twenty, narx_models[[i]]$predictions)
138 }
139
140 # Create a comparison table for NARX models
141 narx_comparison_table <- data.frame(
142   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)"),
143   RMSE = sapply(narx_evaluation_metrics, function(x) x$RMSE),
144   MAE = sapply(narx_evaluation_metrics, function(x) x$MAE),
145   MAPE = sapply(narx_evaluation_metrics, function(x) x$MAPE),
146   SMAPE = sapply(narx_evaluation_metrics, function(x) x$SMAPE)
147 )
148
149 print(narx_comparison_table)
150
151 #Task 2
152
153 # Denormalize the predictions
154 denormalize <- function(x, min_value, max_value) {
155   return(x * (max_value - min_value) + min_value)
156 }
157
158 best_model_index <- which.min(sapply(evaluation_metrics, function(x) x$RMSE))
159 best_model <- models[[best_model_index]]
160 best_model_predictions <- best_model$predictions
161
162 min_value <- min(uow_train$time_twenty)
163 max_value <- max(uow_train$time_twenty)
164

```

```

164
165 denormalized_predictions <- denormalize(best_model_predictions, min_value, max_value)
166
167 # Plot the predicted output vs. desired output using a line chart
168 plot(uow_test$time_twenty, type = "l", col = "blue", xlab = "Time", ylab = "Hour 20 Consumption", main = "Line Chart of Desired vs. Predicted Output")
169 lines(denormalized_predictions, col = "red")
170 legend("topleft", legend = c("Desired Output", "Predicted Output"), col = c("blue", "red"), lty = 1, cex = 0.8)
171
172
173

```

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].