INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology

Machine Learning & Data Mining

5DATA001C

Coursework Report

Name: Rayaan Rilshad

IIT Student ID: 20211315

UoW ID: w1899330

Tutorial Group: R

Table of Contents

Partitioning Clustering

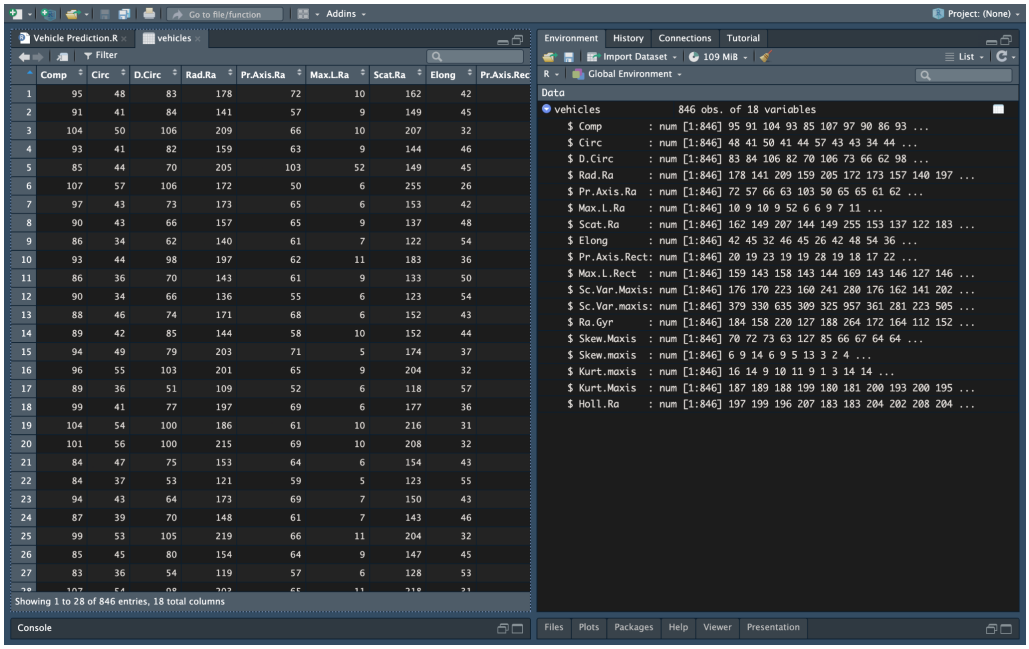
1st Subtask

a. Pre-Processing

Data preprocessing is the process of cleaning and transforming raw data into a format that is suitable for analysis. This process is important because raw data can be incomplete, inconsistent, or contain errors, which can affect the accuracy and efficiency of the analysis. Data pre-processing tasks include removing missing or null values, normalizing data (scaling), and handling outliers.

The first step in the pre-processing task is to load the data and remove any unwanted data, such as missing or null values. For example, in the "vehicles.xlsx" dataset, the "Sample" and "Class" columns are not important because the "Sample" column contains the row number, and the "Class" column contains the vehicle type, which is not required for unsupervised learning.

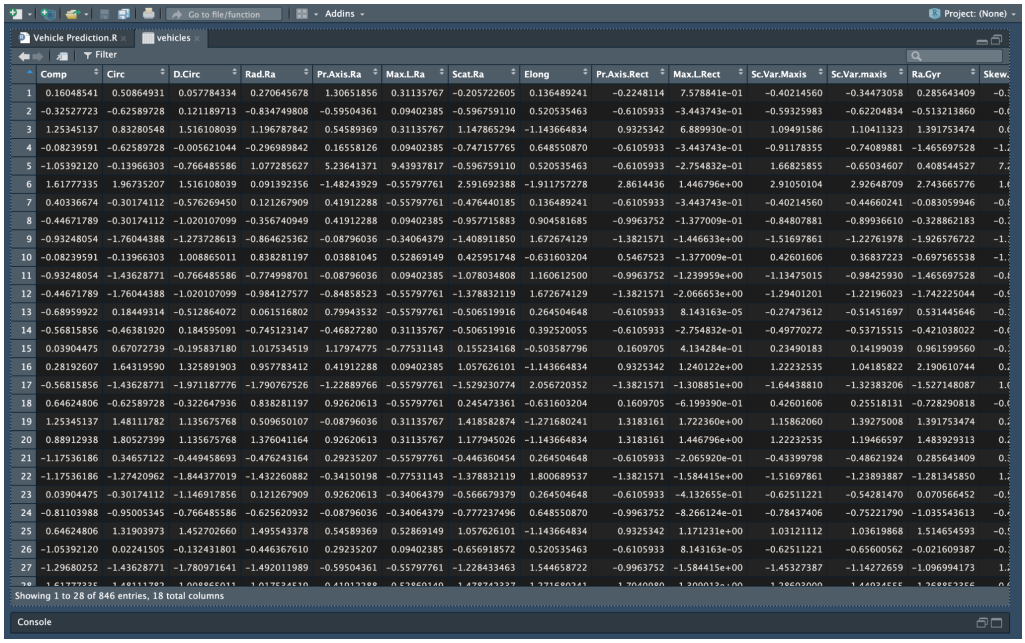
```r
library(readxl)
# Read the Excel file
vehicles <- read_excel("/Users/rayaan/Edu/ML & DS/ML Coursework/PC Part/vehicles.xlsx")

# Remove the first and last columns
vehicles <- vehicles[, -c(1, ncol(vehicles))]

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



The data points in the dataset are calculated and measured using different methods. This can make it difficult for the machine learning module to understand the data, so it is essential to scale the data to the same scale.

```
vehicles <- scale(vehicles)
```

In the R code, the scale() function was used to scale the dataset using the z-score method. This ensures that all data points contribute equally to the analysis and are on the same scale. The following image shows the dataset after the scaling process.



Outliers are data points that are very different from the rest of the data. They can be caused by mistakes in data entry, measurement errors, or just natural variations. Outliers can mess up machine learning models, so finding and removing them is essential before training a model.

One way to identify outliers is to use the z-score method. A z-score tells how far a data point is from the average of its data group. If a data point has a z-score that's bigger than a certain number, it's an outlier. In this case, a z-score of 3 or more is considered to be an outlier.

$$Z\ Score\ (z) = \frac{(Observation - Mean)}{Standard\ Deviation} = \frac{x - \mu}{\sigma}$$

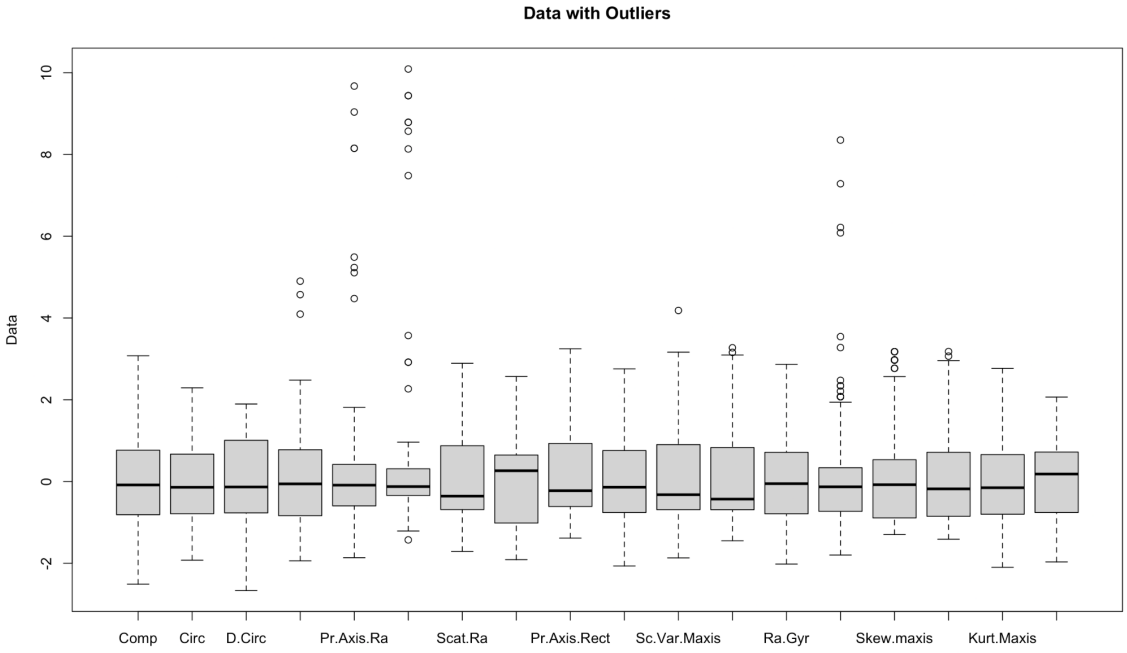
```
outliers <- apply(vehicles, 1, function(x) any(x > 3 | x < -3))
```

According to the below image, 22 outliers have been detected out of 846 data points in the vehicles dataset.

```
> summary(outliers)
   Mode   FALSE    TRUE
logical     824      22
```

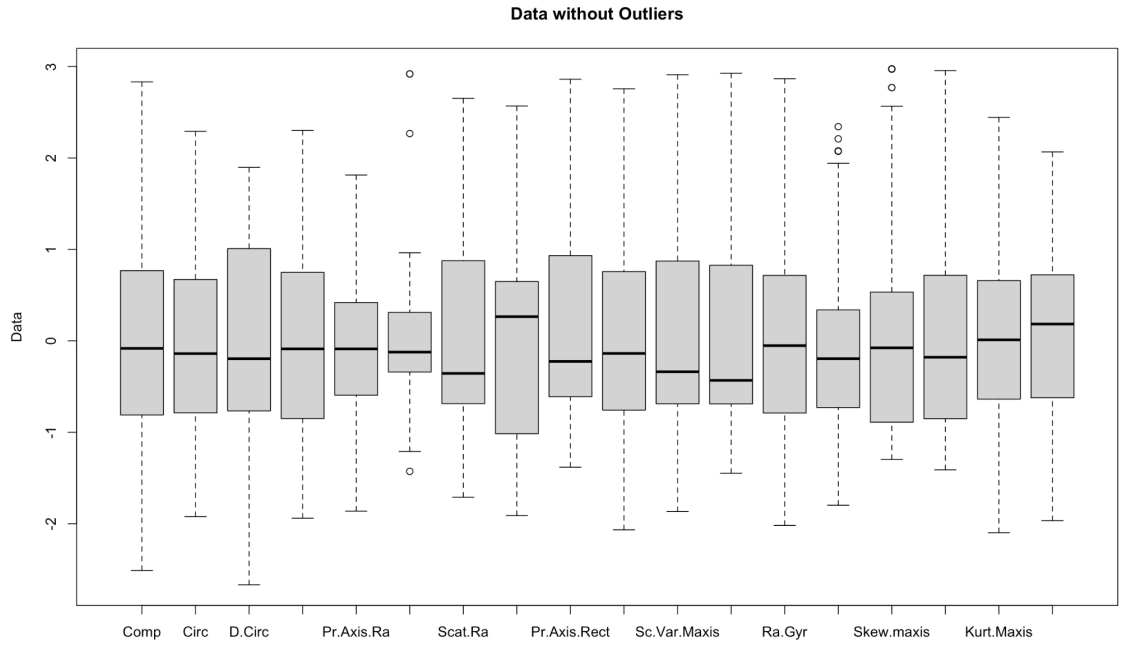
And the boxplot method will help to visualize the outliers.

```
boxplot(vehicles)
```



According to the above image and the summary of the outliers, there are several outliers in the dataset. Therefore, it is necessary to remove them from the dataset in order to get an accurate analysis.

```
vehicles <- subset(vehicles, !outliers)
```



All outliers that are greater than 3 or less than (-3) are removed from the dataset.

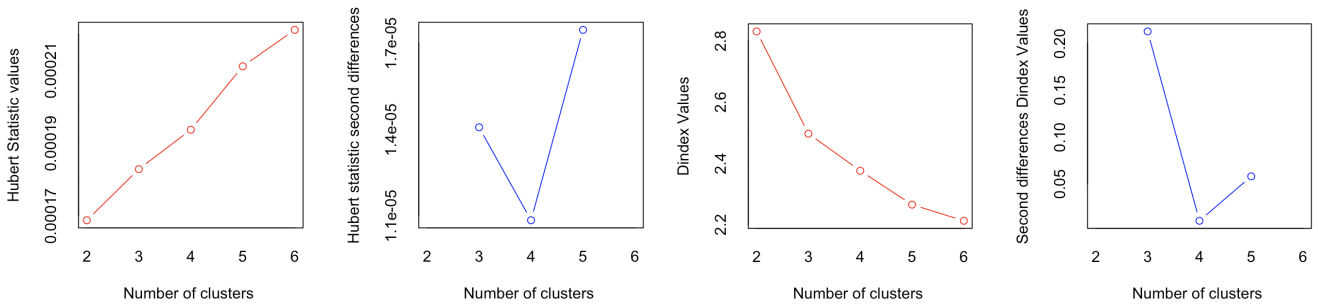
b. Determining the Number of Cluster Centres Using Automated Tools

A cluster is a group of similar data points in a dataset. In this project, automated tools, including NBclust, Elbow, Gap Statistics, and Silhouette methods, must be used to determine the number of cluster centers in the vehicles dataset.

I. NBclust

The NBclust method is used as the first method to determine the best number of clusters using automated tools. To do this, it is required to use the NbClust package and pass the relevant inputs to it. In this case, the “vehicles” dataset is used, and the Euclidean distance metric is used to measure the distance between data points. The clusters are then calculated using the k-means method.

```
library(NbClust)
NBclust <- NbClust(vehicles, distance = "euclidean", min.nc=2, max.nc =6, method = "kmeans")
```



As shown in the above graphs (Hubert index and the D index), as well as in the below figure, the NBClust method clearly predicts that the best number of clusters is 3, according to the majority rule.

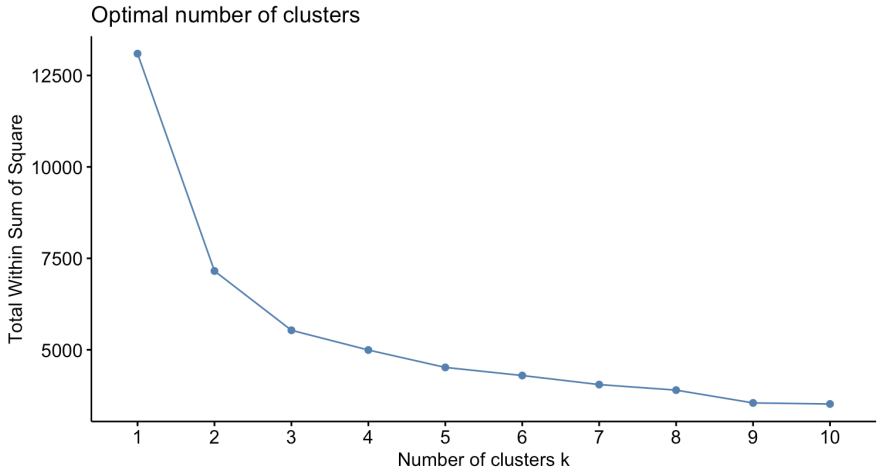
```
*******************************************************************
* Among all indices:
* 8 proposed 2 as the best number of clusters
* 13 proposed 3 as the best number of clusters
* 2 proposed 5 as the best number of clusters
* 1 proposed 6 as the best number of clusters

                   ***** Conclusion *****

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


*******************************************************************
```

II. Elbow

The next method is the elbow method. It is a simple way to find the best number of clusters in a dataset. It plots the within-cluster sum of squares (WSS) against the number of clusters. The WSS is a measure of how spread out the data is within each cluster.

```
fviz_nbclust(vehicles, kmeans, method = "wss")
```

Basically, the elbow method finds the point on the curve where the WSS starts to decrease more slowly. This point is called the "elbow," and the number of clusters at the elbow is the best number of clusters because it means the data is most evenly distributed among the clusters.

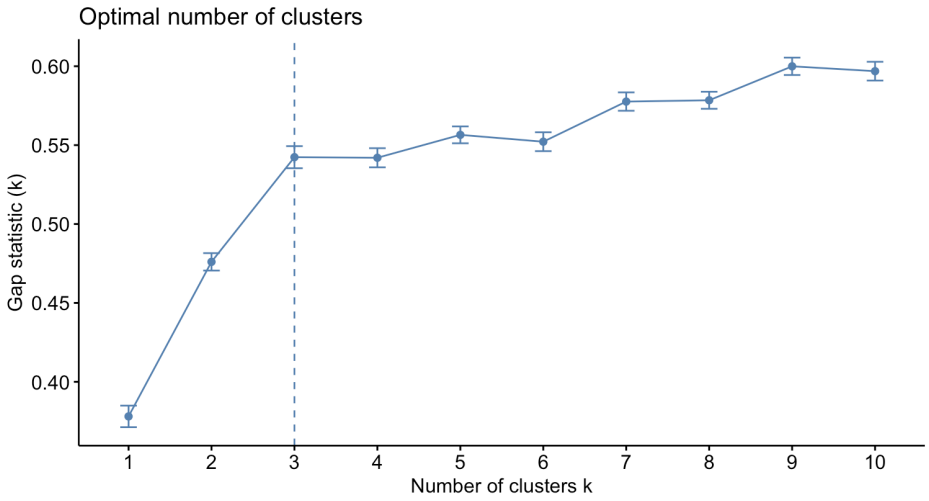


According to the elbow method, 3 clusters seem to be the best number of clusters to use in the vehicles dataset.

III. Gap Statistics

Gap statistics is all about comparing the WSS of the data to the WSS of randomly generated data with the same number of clusters.

```
fviz_nbclust(vehicles, kmeans, method = "gap_stat")
```

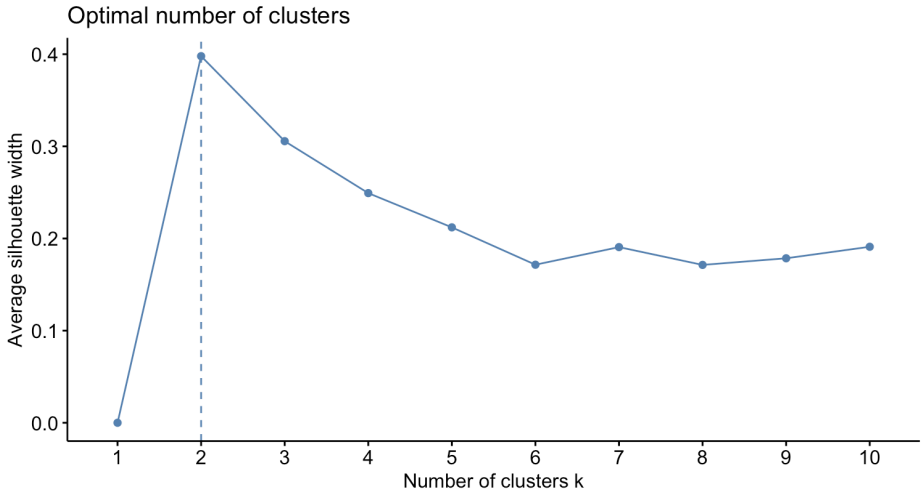


In the gap statistic method, the best number of clusters is the one with the biggest gap between the two WSS values. In this case, the gap statistic predicts that the best number of clusters is 3.

IV. Silhouette

Silhouette is another automated tool to calculate the number of clusters. It calculates the average silhouette width for each cluster, which tells how similar each data point is to its own cluster compared to other clusters.

```
fviz_nbclust(vehicles, kmeans, method = "silhouette")
```



In this case, the best number of clusters is the one with the highest average silhouette width across all other clusters. Based on the silhouette method, the best number of clusters for this dataset is 2.

Conclusion

Method	NBclust	Elbow	Gap Statistic	Silhouette
Suggested Clusters	3	3	3	2

Based on the results of the four automated tools, the optimal number of clusters seems to be 3 because the majority of methods suggest it. The NBclust algorithm, the elbow method, and the gap statistics method all suggest 3 clusters, while only the silhouette method suggests 2. Therefore, k=3 is the best fit for the data.

c. k-Means Clustering

Based on the results of the four automated tools, the best number of clusters in this dataset is three.

Basically, K-means start by randomly assigning each data point to a cluster. Then, it calculates the mean of each cluster and moves each data point to the cluster with the closest mean. This process continues until the clusters do not change anymore.

```
k <- 3
km <- kmeans(vehicles, centers = k)
```

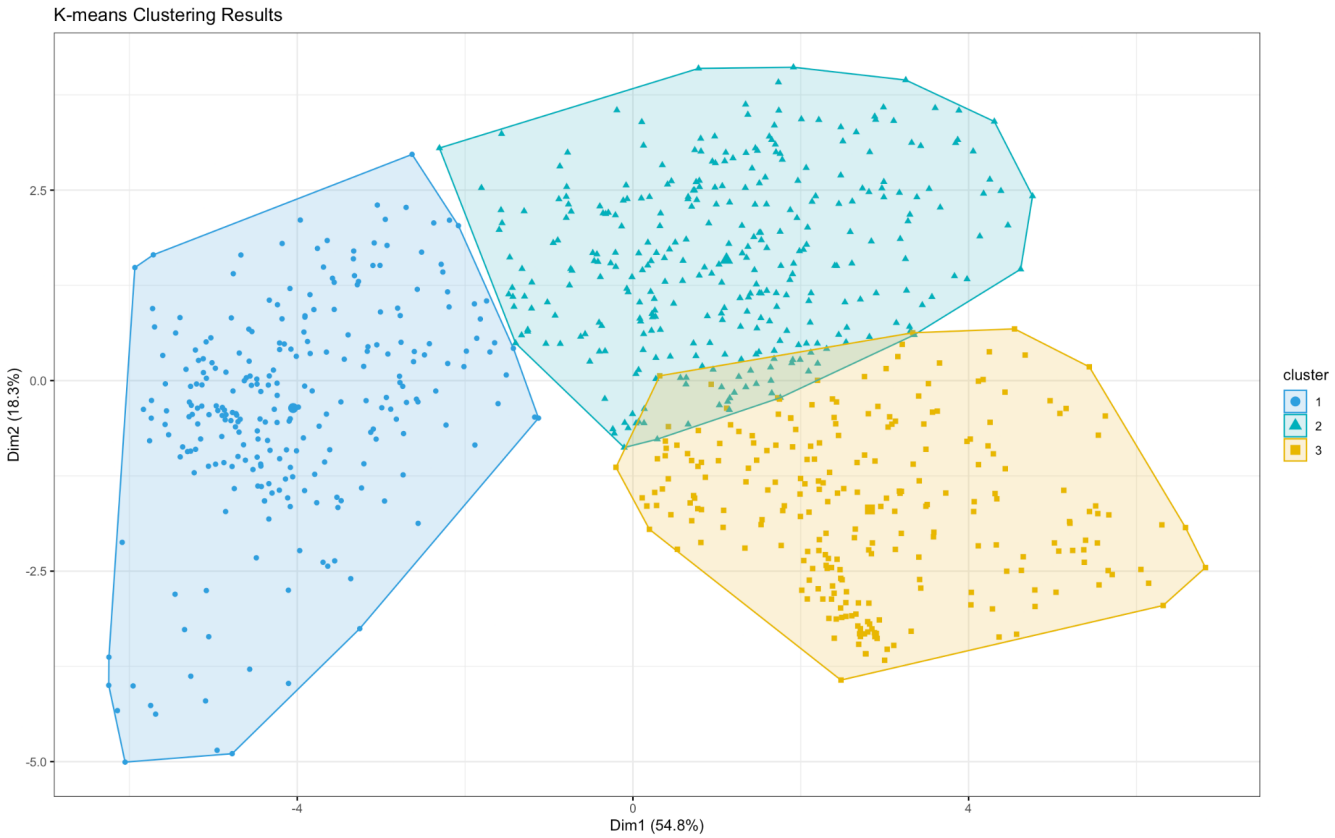
Here are the k-means clustering centers for each variable, as an output of the above code.

```
> km$centers
       Comp       Circ     D.Circ     Rad.Ra Pr.Axis.Ra   Max.L.Ra    Scat.Ra      Elong Pr.Axis.Rect
1  1.1165376  1.1619679  1.1934506  0.9731249  0.1151666  0.2257667  1.2593577 -1.1965515    1.2572464
2 -0.2330582 -0.5696764 -0.3014468 -0.0452883  0.1802391 -0.1837559 -0.4638443  0.3301125   -0.4912420
3 -0.9290108 -0.5312966 -0.9039070 -1.0946062 -0.5603757 -0.2945158 -0.7788336  0.8674833   -0.7460106
  Max.L.Rect Sc.Var.Maxis Sc.Var.maxis     Ra.Gyr Skew.Maxis  Skew.maxis   Kurt.maxis  Kurt.Maxis    Holl.Ra
1  1.0850506    1.1621871    1.2619905  1.0567412 -0.1230984  0.13450468  0.248048532  0.02533749  0.2035719
2 -0.5381307   -0.4286229   -0.4680026 -0.6003008 -0.6460788 -0.05442748  0.001129095  0.80841042  0.6955240
3 -0.4911547   -0.7995852   -0.7826871 -0.3947021  0.7990842 -0.12894793 -0.295146631 -1.06047363 -1.1066826
>
```

According to the k-means clustering algorithm, the image below shows which data point is assigned to which cluster with 3 clusters.

```
> km$cluster
  [1] 2 2 1 2 1 2 2 2 2 2 2 2 2 1 3 2 1 1 3 3 2 2 1 2 3 1 1 3 2 2 2 1 2 2 3 1 3 1 3 3 2 3
 [44] 3 3 3 2 3 2 1 2 1 2 2 3 1 3 1 3 3 3 2 3 3 1 2 1 1 1 2 3 2 1 2 3 1 3 3 1 2 3 2 2 3 2 3
 [87] 1 2 1 2 3 1 3 3 1 3 2 2 3 1 1 1 3 3 1 2 2 3 3 3 2 1 1 3 2 3 3 2 3 3 3 3 3 2 1 1 2 3 1 3
[130] 2 3 2 2 3 1 3 2 1 2 2 2 2 1 2 2 1 2 1 2 3 2 2 3 1 2 2 1 1 2 1 3 3 1 1 2 1 2 2 2 2 2 3
[173] 1 3 2 3 1 2 2 2 1 2 1 2 2 1 2 3 1 3 3 3 2 2 1 1 2 2 2 3 3 1 2 2 2 1 3 2 3 1 3 2 1 3 1
[216] 3 3 2 1 2 1 3 3 3 3 1 2 3 2 3 1 3 2 2 3 1 3 3 2 2 1 3 3 1 3 2 2 1 2 2 1 1 3 2 2 2 1 3
[259] 3 2 2 3 3 2 2 2 1 2 3 3 1 2 2 3 3 1 3 2 2 3 1 3 2 2 2 1 2 1 3 2 2 1 2 2 2 3 2 1 1 1 1
[302] 1 3 2 1 3 3 3 2 3 1 1 3 1 2 3 1 2 2 2 2 1 1 3 1 1 3 1 2 2 2 3 3 1 1 1 1 2 2 2 1 3 2 3
[345] 1 2 2 1 2 1 1 1 2 2 3 1 3 3 3 2 2 2 2 2 3 1 1 3 3 1 3 1 3 1 2 3 2 3 1 3 2 2 1 2 2 2 2
[388] 1 2 1 2 1 2 3 3 2 2 2 3 3 2 3 1 2 2 3 2 3 1 2 3 2 2 1 2 1 2 1 1 3 3 1 2 3 3 2 1 1 3 2
[431] 1 1 3 1 1 1 2 2 2 2 2 1 3 3 2 1 2 2 1 2 3 1 3 3 1 1 2 3 1 1 1 3 1 1 3 2 3 1 1 2 2 3 3
[474] 1 2 3 1 1 2 3 1 1 2 1 3 3 1 1 1 3 3 1 1 1 2 2 1 3 2 1 3 3 1 3 2 2 3 2 1 2 1 1 2 3 2 1
[517] 1 3 3 2 1 2 1 1 2 2 2 2 3 3 3 2 2 1 3 3 2 3 1 2 1 3 3 1 1 2 1 2 2 2 1 2 3 2 1 2 2 3 1
[560] 1 1 1 2 3 3 3 1 1 1 2 1 3 2 1 3 3 3 2 3 1 2 2 2 2 2 2 1 2 2 1 2 2 2 3 1 3 3 2 3 2 2 3
[603] 3 1 1 3 2 2 1 2 2 1 2 3 1 3 1 3 3 2 3 2 1 1 3 1 2 2 3 2 3 1 2 1 3 2 2 2 3 3 3 2 1 2 1
[646] 3 2 2 2 2 2 1 2 3 1 2 1 2 2 1 3 1 3 2 2 2 3 1 2 3 2 3 1 2 2 1 3 2 3 2 2 3 2 1 1 2 2 1 1
[689] 2 3 2 1 1 1 1 2 1 2 2 1 1 2 1 2 1 2 3 1 2 3 1 1 1 2 3 3 1 1 1 2 1 2 2 1 2 3 2 3 2 1 2
[732] 3 2 2 2 3 1 3 3 3 1 3 1 1 3 2 2 1 2 3 1 1 3 2 2 1 1 1 3 1 2 1 1 3 3 1 3 1 2 3 2 1 1 2
[775] 3 2 1 1 2 2 3 2 2 1 3 2 1 3 3 1 3 2 3 3 3 2 1 1 2 3 1 2 1 1 3 2 1 3 3 2 2 1 3 3 3 2 2
[818] 2 2 2 2 1 2 3
>
```

Finally, the following image visually represents the three clusters formed by the K-means clustering algorithm, along with their data points.



In clustering analysis, the within-cluster sum of squares (WSS), the between-cluster sum of squares (BSS), and the total sum of squares (TSS) can be used to evaluate the quality of the clustering solution.

```
wss <- sum(km$withinss)
bss <- sum(km$betweenss)
tss <- wss + bss

cat("Ratio of BSS over TSS:", bss / tss)
```

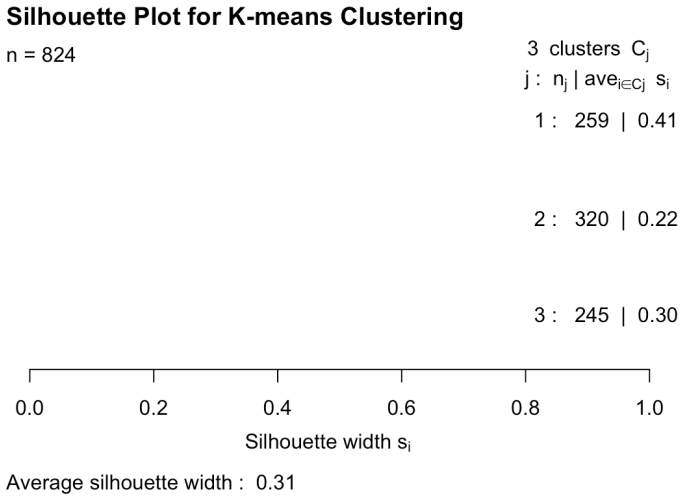
```
> cat("WSS :", wss)
WSS : 5533.808> cat("BSS :", bss)
BSS : 7561.925> cat("TSS :", tss)
TSS : 13095.73> cat("Ratio of BSS over TSS:", bss / tss)
Ratio of BSS over TSS: 0.5774343
>
```

The ratio of BSS over TSS is 0.577, which is close to 0.5, indicating that the model is well-fit.

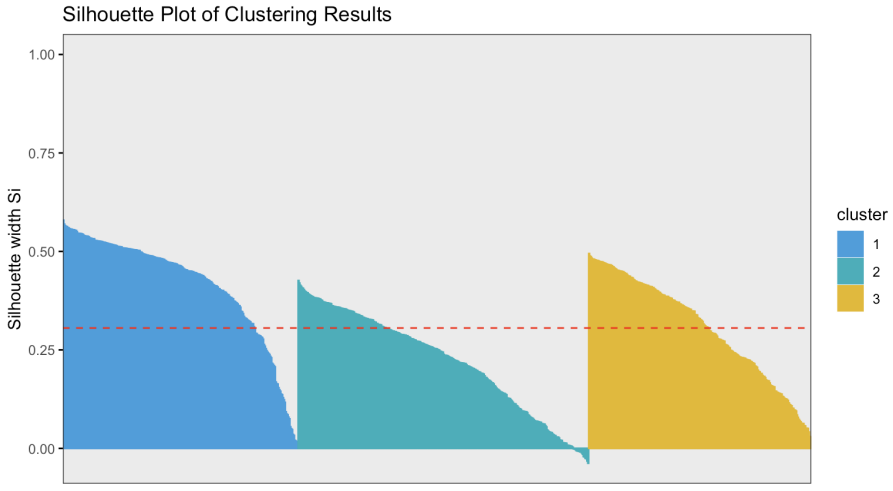
d. Silhouette Plot

A silhouette plot is a useful tool for evaluating the quality of a clustering solution by measuring how close each point in one cluster is to its neighbouring cluster. A high average silhouette width indicates that the clustering solution is good, as most of the observations are well-separated from other observations in their own cluster.

```
sil_width <- silhouette(km$cluster, dist(vehicles))
plot(sil_width, main = "Silhouette Plot for K-means Clustering")
```



The average silhouette width for the clustering solution is 0.31, which indicates that the clustering solution is relatively good. This is because most of the observations are well-separated from other observations in their own cluster



This plot shows that the three clusters are well-defined. The observations in each cluster are close to one another, and they are also well-separated from the observations in other clusters. This suggests that the clustering solution is relatively good.

2nd Subtask

e. PCA Analysis

Not all data sections in the dataset are required for analysis. Principal Component Analysis (PCA) is the process of reducing the dimensionality of a dataset while retaining as much of the original information as possible.

In this project, PCA is performed on the pre-processed "vehicles" dataset using the prcomp() function.

```
pca <- prcomp(vehicles)
```

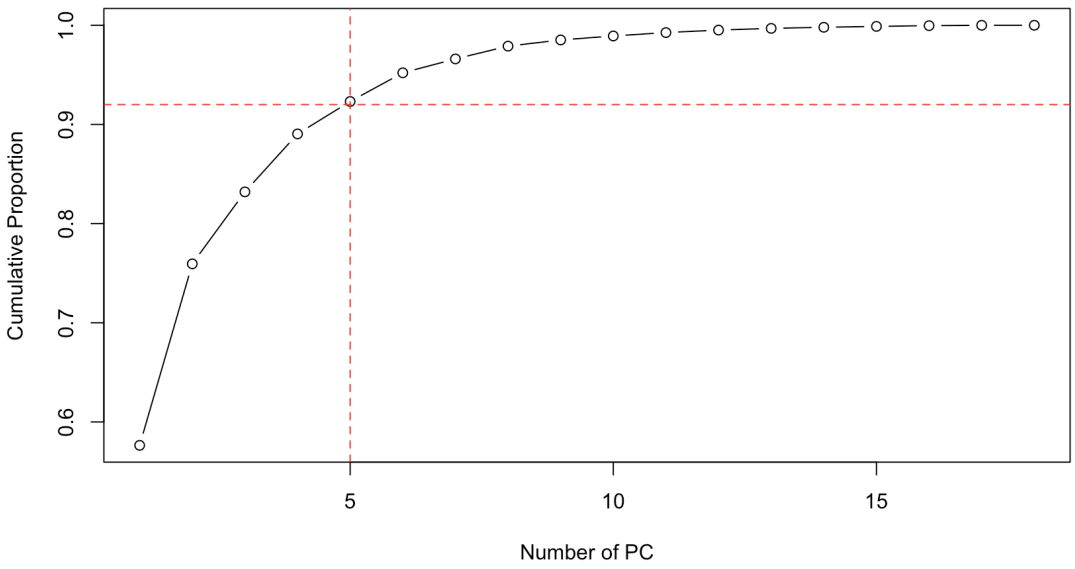
The eigenvalues, which represent the variance explained by each principal component, and the eigenvectors, which represent the direction of the PC in the original variable space, are extracted from the PCA object using the “sdev” and “rotation” attributes, respectively.

```
eigenvalues <- pca$sdev^2
eigenvectors <- pca$rotation
```

The cumulative proportion of variance explained is a measurement of how much of the total variation in the data is accounted for by each PC in a PCA.

```
cum_prop_var <- cumsum(eigenvalues / sum(eigenvalues))
```

Choosing the correct number of PCs is an important step in PCA. The first step in choosing PCs is to determine how much variance we want to explain. In this case, it is required to explain at least 92% of the cumulative variance.

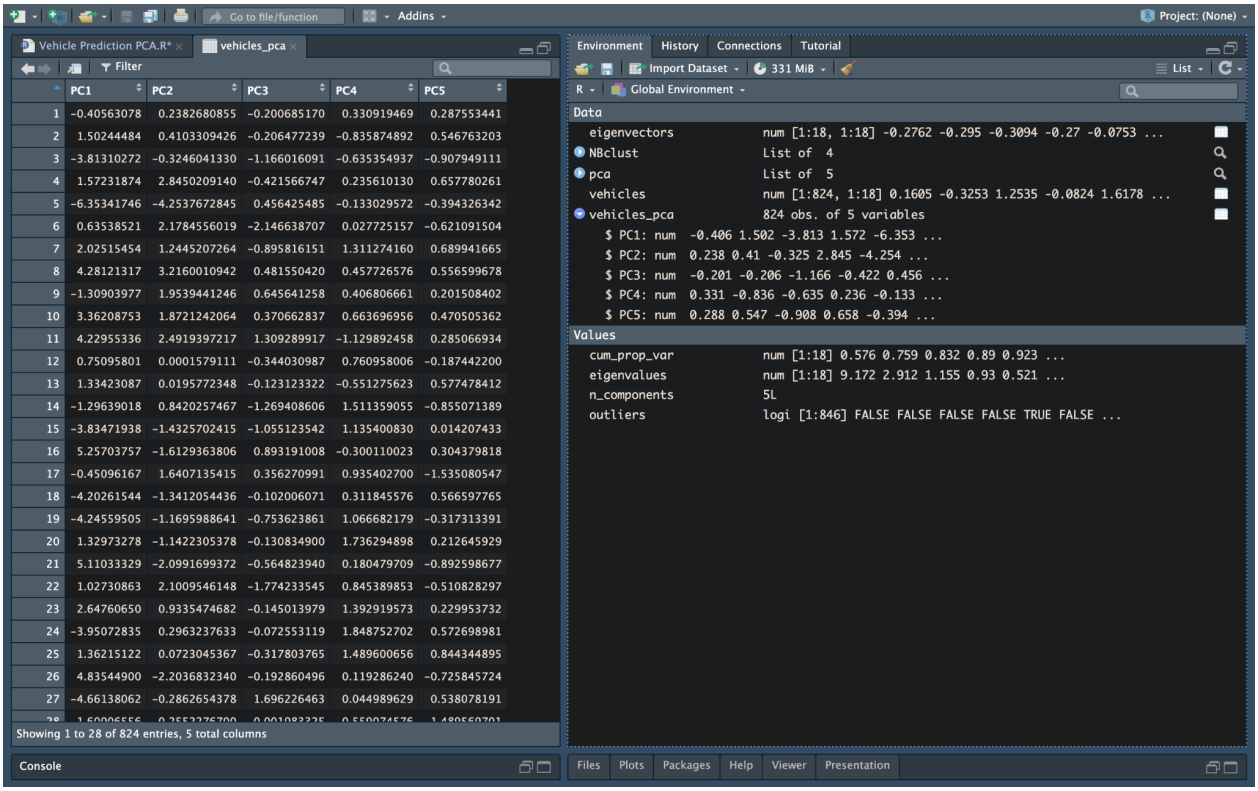


```
> cat("Number of components to explain 92% variance:", n_components, "\n")
Number of components to explain 92% variance: 5
>
```

According to the above images, 5 components are required to explain 92% of the variance. Therefore, a new dataset should be created with those chosen 5 PCs.

```
vehicles_pca <- predict(pca, newdata = vehicles)[, 1:n_components]
```

The newly created table looks like the figure below.



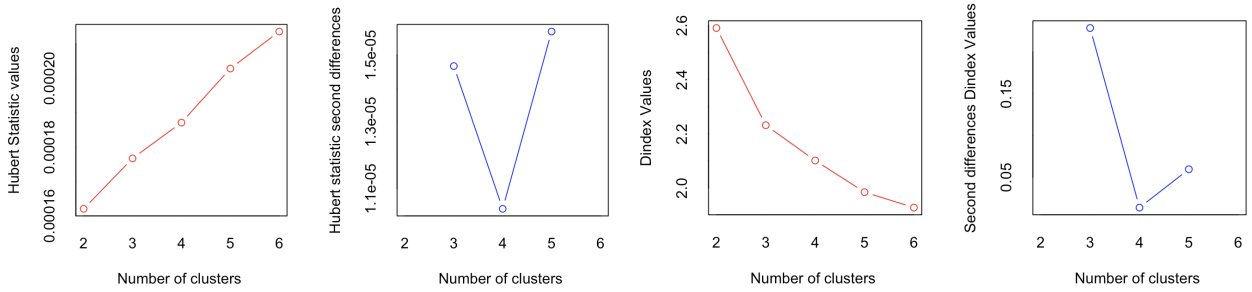
	PC1	PC2	PC3	PC4	PC5
1	-0.40563078	0.2382680855	-0.200685170	0.330919469	0.287553441
2	1.50244484	0.4103309426	-0.206477239	-0.835874892	0.546763203
3	-3.81310272	-0.3246041330	-1.166016091	-0.635354937	-0.907949111
4	1.57231874	2.8450209140	-0.421566747	0.235610130	0.657780261
5	-6.35341746	-4.2537672845	0.456425485	-0.133029572	-0.394326342
6	0.63538521	2.1784556019	-2.146638707	0.027725157	-0.621091504
7	2.02515454	1.2445207264	-0.895816151	1.311274160	0.689941665
8	4.28121317	3.2160010942	0.481550420	0.457726576	0.556599678
9	-1.30903977	1.9539441246	0.645641258	0.406806661	0.201508402
10	3.36208753	1.8721242064	0.370662837	0.663696956	0.470505362
11	4.22955336	2.4919397217	1.309289917	-1.129892458	0.285066934
12	0.75095801	0.0001579111	-0.344030987	0.760958006	-0.187442200
13	1.33423087	0.0195772348	-0.123123322	-0.551275623	0.577478412
14	-1.29639018	0.8420257467	-1.269408606	1.511359055	-0.855071389
15	-3.83471938	-1.4325702415	-1.055123542	1.135400830	0.014207433
16	5.25703757	-1.6129363806	0.893191008	-0.300110023	0.304379818
17	-0.45096167	1.6407135415	0.356270991	0.935402700	-1.535080547
18	-4.20261544	-1.3412054436	-0.102006071	0.311845576	0.566597765
19	-4.24559505	-1.1695988641	-0.753623861	1.066682179	-0.317313391
20	1.32973278	-1.1422305378	-0.130834900	1.736294898	0.212645929
21	5.11033329	-2.0991699372	-0.564823940	0.180479709	-0.892598677
22	1.02730863	2.1009546148	-1.774233545	0.845389853	-0.510828297
23	2.64760650	0.9335474682	-0.145013979	1.392919573	0.229953732
24	-3.95072835	0.2963237633	-0.072553119	1.848752702	0.572698981
25	1.36215122	0.0723045367	-0.317803765	1.489600656	0.844344895
26	4.83544900	-2.2036832340	-0.192860496	0.119286240	-0.725845724
27	-4.66138062	-0.2862654378	1.696226463	0.044989629	0.538078191

These 5 data columns will be the dataset used in the PCA analysis.

f. Determining the Number of Cluster Centres for the PCA-based Dataset

As discussed in the first subtask, NBclust, Elbow, Gap statistics, and silhouette methods can be used to find the most suitable number of clusters for the dataset.

I. NBclust

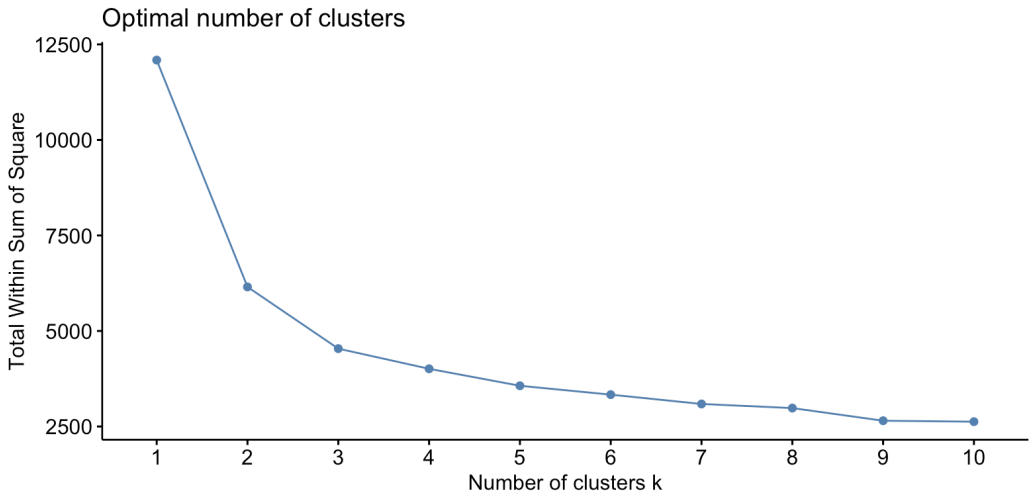


As shown in the above graphs (Hubert index and the D index) and also in the below figure, the NBClust method clearly predicts that the best number of clusters is 3, according to the majority rule.

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

                   ***** Conclusion *****

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

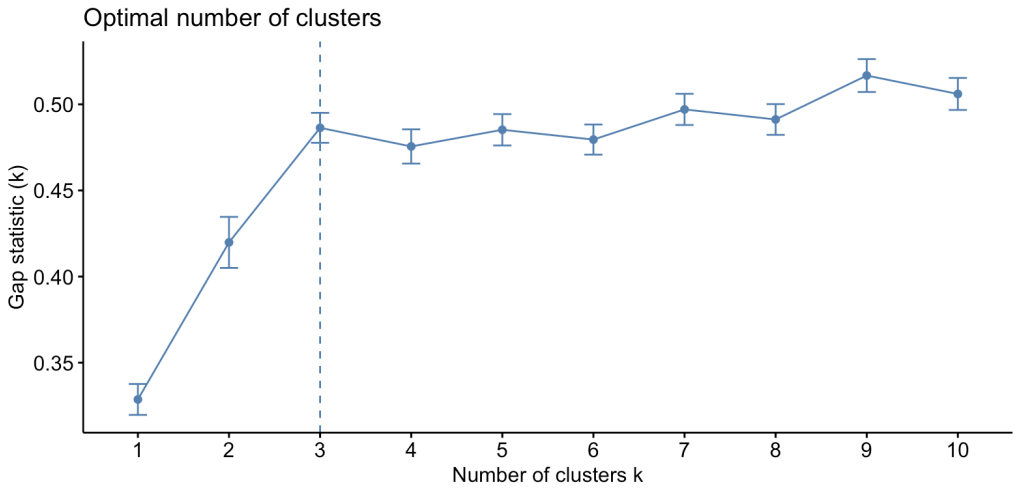
II. Elbow

According to the plot below, the elbow of the curve is at 3 clusters. Therefore, the elbow method is suggesting that 3 clusters are the better option.



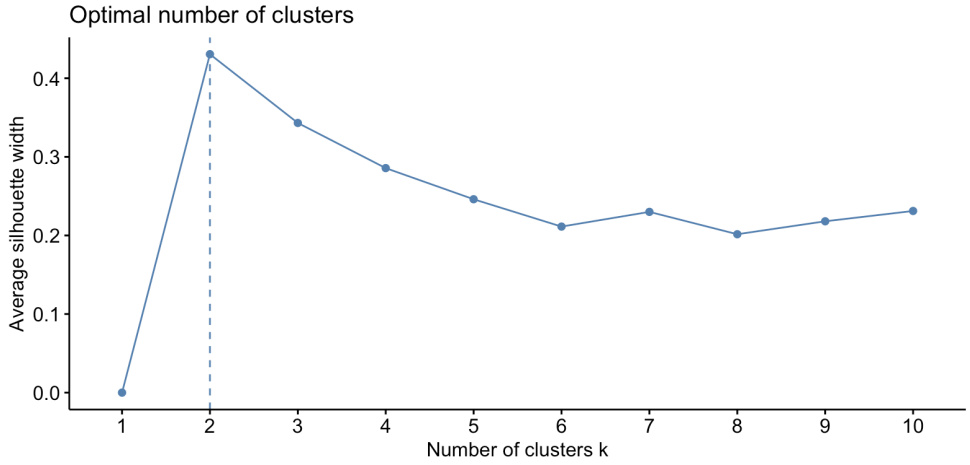
III. Gap Statistics

Based on the results of the gap statistic method, it is recommended to use 3 as the number of clusters.



IV. Silhouette

The previous 3 automated methods are suggested k=3, but the Silhouette method is suggesting the number 2 as clusters for the PCA.



Conclusion

Method	NBclust	Elbow	Gap Statistic	Silhouette
Suggested Clusters	3	3	3	2

Based on the results of the four automated tools, most suggest that the optimal number of clusters is 3. Therefore as a conclusion, k = 3.

g. k-means Clustering for PCA-based Dataset

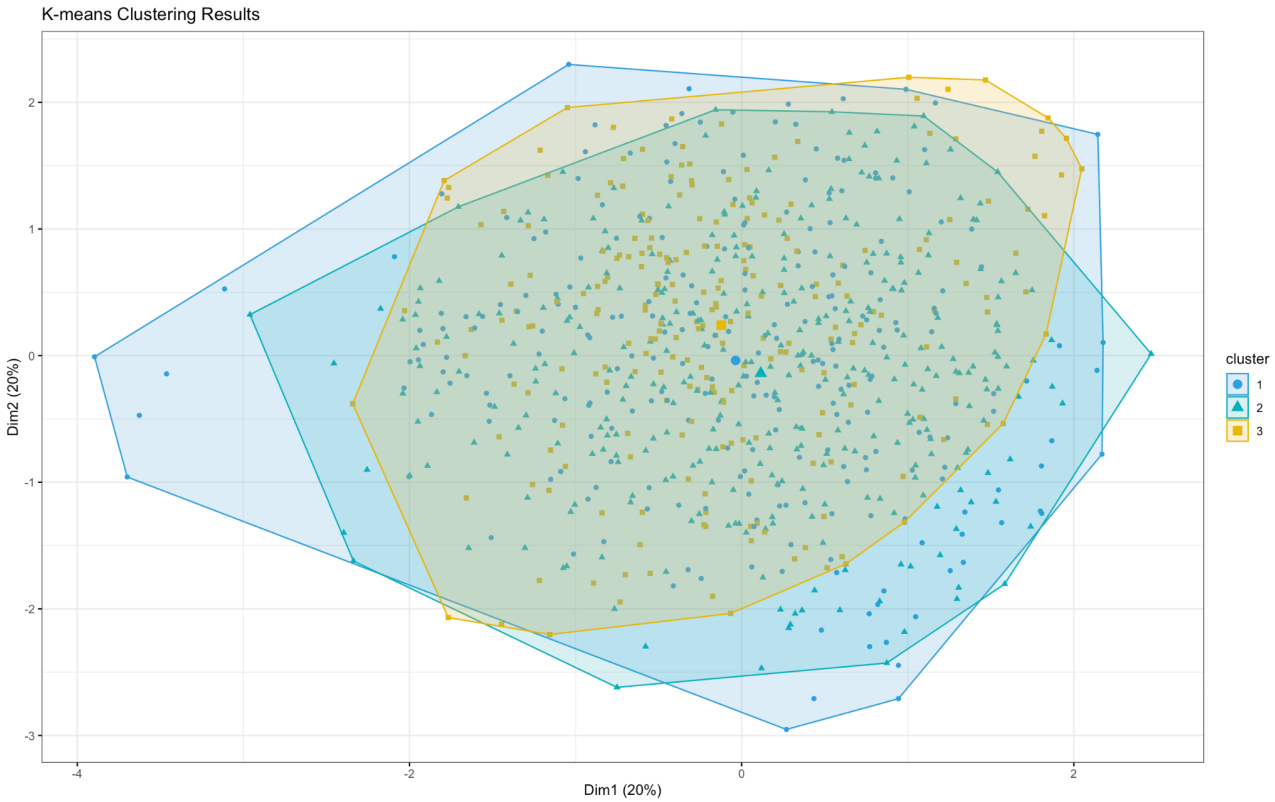
According to the conclusion of the automated tools, k=3 will be the most suitable number of clusters.

```
k_pca <- 3
km_pca <- kmeans(vehicles_pca, centers = k_pca)
```

As an output of the above code, here are the k-means clustering centers for each variable and the cluster data points.

```
> km_pca$centers
        PC1         PC2          PC3         PC4         PC5
1 -3.931465 -0.2627187  0.060458012 -0.0769224 -0.01392689
2  1.129339  1.4005849 -0.044490507  0.1517494 -0.03214412
3  2.754009 -1.6903824 -0.003984049 -0.1295134  0.06088363
> km_pca$cluster
  [1] 2 2 1 2 1 2 2 2 2 2 2 2 2 1 3 2 1 1 3 3 2 2 1 2 3 1 1 3 2 2 2 1 2 2 3 1 3 1 1 3 3 2 3 3 3 3 2 3 2 1 2 1 2 2 3 1 3 1 3 3 3
 [62] 2 3 3 1 2 1 1 1 2 3 2 1 2 3 1 3 3 1 2 3 2 2 3 2 3 1 2 1 2 3 1 3 3 1 3 2 2 3 1 1 1 3 3 1 2 2 3 3 3 2 1 1 3 2 3 3 2 3 3 3 3
[123] 2 1 1 2 3 1 3 2 3 2 2 3 1 3 2 1 2 2 2 2 1 2 2 1 2 1 2 3 2 2 3 1 2 2 1 1 2 1 3 3 1 1 2 1 2 2 2 2 2 3 1 3 2 3 1 2 2 2 1 2 1
[184] 2 2 1 2 3 1 3 3 3 2 2 1 1 2 2 2 3 3 1 2 2 2 1 3 2 3 1 3 2 1 3 1 3 3 2 1 2 1 3 3 3 3 1 2 3 2 3 1 3 2 2 3 1 3 3 2 2 1 3 3 1
[245] 3 2 2 1 2 2 1 1 3 2 2 2 1 3 3 2 2 3 3 2 2 2 1 2 3 3 1 2 2 3 3 1 3 2 2 3 1 3 2 2 2 1 2 1 3 2 2 1 2 2 2 3 2 1 1 1 1 1 2 2 1
[306] 3 3 3 2 3 1 1 3 1 2 3 1 2 2 2 2 1 1 3 1 1 3 1 2 2 2 3 3 1 1 1 1 2 2 2 1 3 2 3 1 2 2 1 2 1 1 1 2 2 3 1 2 3 3 2 2 2 2 2 3 1
[367] 1 3 3 1 3 1 3 1 2 2 2 2 1 3 2 2 1 2 2 2 2 1 2 1 2 1 2 3 3 2 2 2 3 3 2 3 1 2 2 3 2 3 1 2 3 2 2 1 2 1 2 1 1 3 3 1 2 3 3 2 1
[428] 1 3 2 1 1 3 1 1 1 2 2 2 2 2 1 3 3 2 1 2 2 1 2 3 1 3 3 1 1 2 3 1 1 1 3 1 1 2 2 3 1 1 2 2 3 3 1 2 3 1 1 2 3 1 1 2 1 2 3 1 1
[489] 1 3 3 1 1 1 2 2 1 3 2 1 3 3 1 3 2 2 3 2 1 2 1 1 2 3 2 1 1 3 3 2 1 2 1 1 2 2 2 2 2 3 3 2 2 1 3 3 2 3 1 2 1 3 3 1 1 2 1 2 2
[550] 2 1 2 3 2 1 2 2 3 1 1 1 1 2 3 3 3 1 1 1 2 1 3 2 1 3 3 3 2 3 1 2 2 2 2 2 2 1 2 2 1 2 2 2 3 1 3 3 2 3 2 2 3 3 1 1 3 2 2 1 2
[611] 2 1 2 3 1 3 1 3 3 2 3 2 1 1 2 1 2 2 3 2 3 1 2 1 3 2 2 2 3 2 3 2 1 2 1 3 2 2 2 2 1 2 3 1 2 1 2 2 1 3 1 3 2 2 2 3 1 2 3 2 3
[672] 1 2 2 1 3 2 3 2 2 3 2 1 1 2 2 1 1 2 3 2 1 1 1 1 2 1 2 2 1 1 2 1 2 1 2 2 1 2 3 1 1 1 2 3 3 1 1 1 2 1 2 2 1 2 3 2 3 2 1 2 3
[733] 2 2 2 3 1 3 3 3 1 3 1 1 3 2 2 1 2 3 1 1 3 2 2 1 1 1 3 1 2 1 1 3 3 1 3 1 2 3 2 1 1 2 3 2 1 1 2 2 3 2 2 1 3 2 1 3 3 1 3 2 3
[794] 3 3 2 1 1 2 3 1 2 1 1 3 2 1 3 3 2 2 1 2 3 3 2 2 2 2 2 2 1 2 3
>
```

As a summary of the K-means clustering, the image below visually represents the three clusters and their variables.



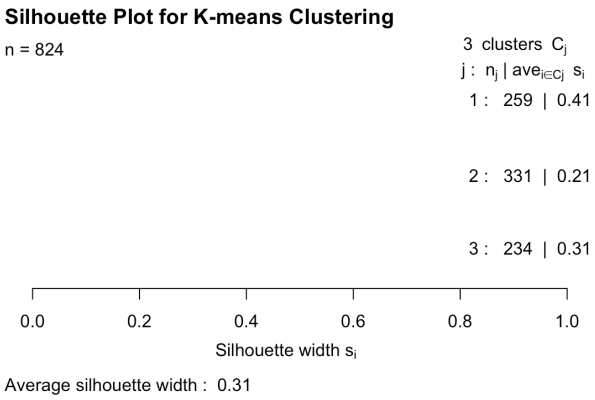
The ratio of BSS over TSS is used to evaluate the quality of the clustering solution.

```
> cat("WSS :", wss_pca)
WSS : 4537.671> cat("BSS :", bss_pca)
BSS : 7551.914> cat("TSS :", tss_pca)
TSS : 12089.58> cat("Ratio of BSS over TSS:", bss_pca / tss_pca)
Ratio of BSS over TSS: 0.6246628
>
```

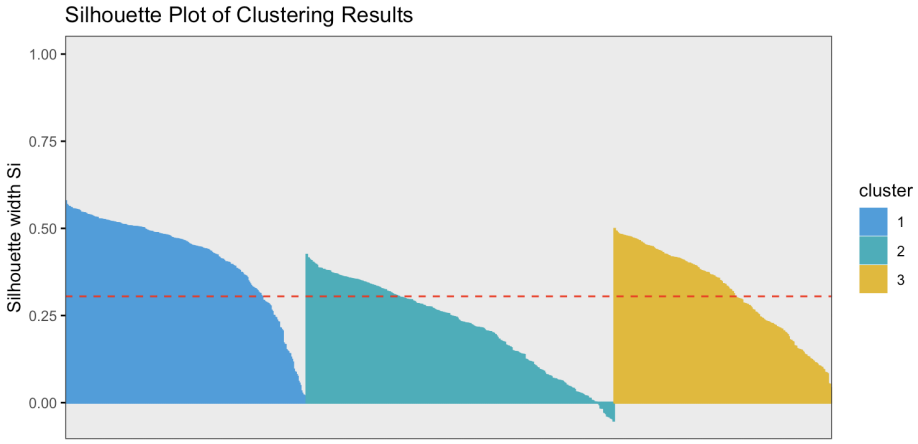
The BSS over TSS ratio for the PCA-based dataset is 0.62, while 0.577 for the original dataset. Therefore, the PCA-based clustering provides better accuracy than the original dataset.

h. Silhouette Plot for PCA-based Dataset

The Silhouette method can evaluate how close each point in one cluster is to its neighboring clusters.



The average silhouette width for the k-means clustering solution is 0.31, similar to the normal k-means clustering part.



i. Calinski-Harabasz Index

The Calinski-Harabasz Index (CH Index) is a method for evaluating the quality of clustering results. It is a measure of separation between clusters and can be used to compare different clustering solutions. The CH Index measures the ratio of between-cluster variance to within-cluster variance.

$$CH\ Index = \frac{BSS/(k-1)}{WSS/(n-k)}$$

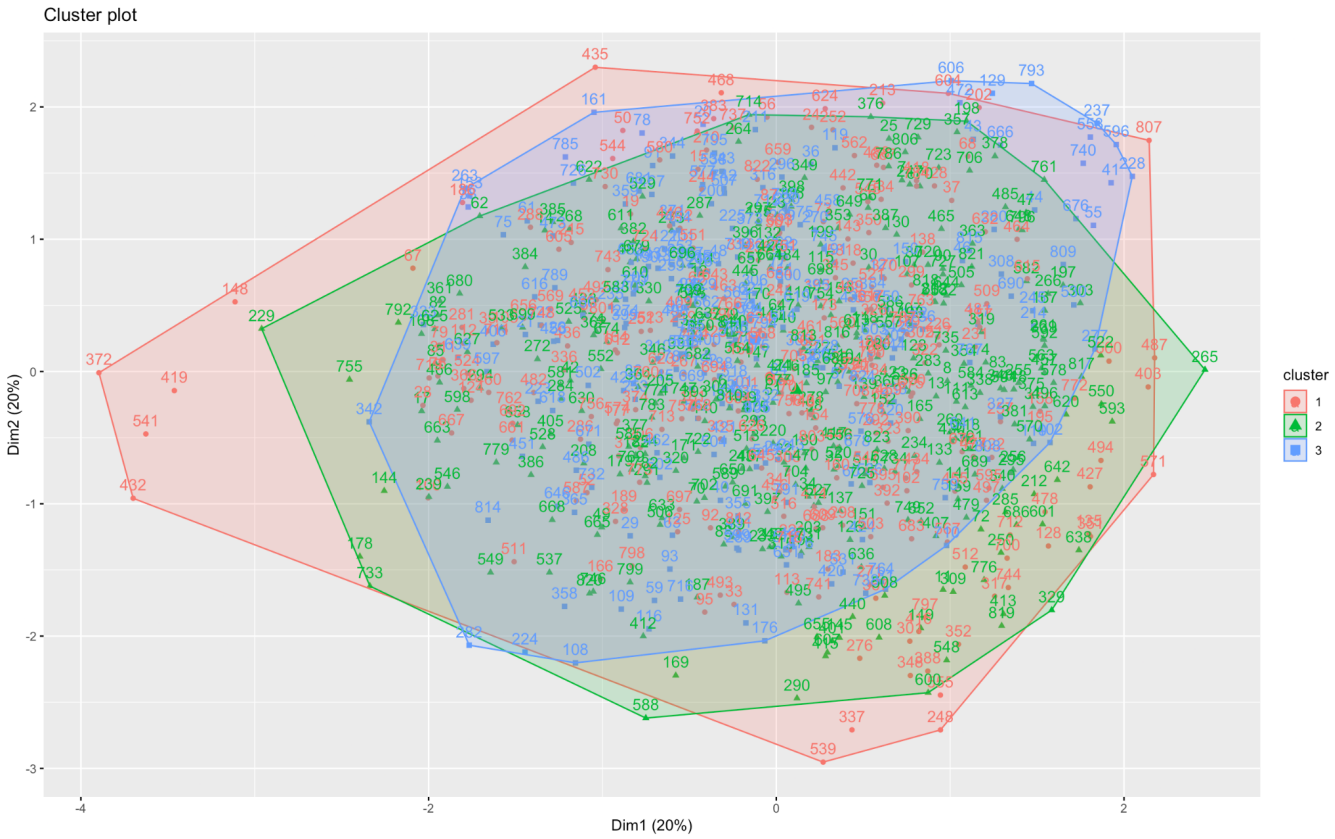
For this purpose, calinhara ( ) function in the fvp library is used.

```
ch_index <- calinhara(vehicles_pca, km_pca$cluster)
```

A higher value of the CH index indicates that the clusters are more well-separated.

```
> cat("Calinski-Harabasz Index:", ch_index)
Calinski-Harabasz Index: 683.1832
```

The CH index is 683.18, which is a relatively higher value. In conclusion, the clusters are very well-separated.



Energy Forecasting

a. Input Variables for MLP Models in Electricity Load Forecasting

In electricity load forecasting, the input variables used in MLP models typically include:

- **Lagged variables:** This method uses past values of energy usage (previous hour, two hours ago, three hours ago, etc.) as inputs to the neural network to predict future usage. (Zhu et al., 2019)
- **Weather variables:** Temperature, humidity, and wind speed can impact energy usage. Therefore, they are also used as inputs to the neural network. (Zhu et al., 2019)
- **Day and time variables:** Energy usage changes throughout the day and week. For example, it is typically higher on weekdays than at weekends and higher during the day than at night. These factors can also be used as inputs to the neural network for energy forecasting. (Zhu et al., 2019)
- **Other variables:** Economic indicators, population, and events in the area can also be used as inputs to the neural network. (Zhu et al., 2019)

These variables are used to capture the patterns and trends in electricity consumption. Some commonly used forecasting schemas use these variables to predict future usage. Some of them are:

- **Autoregressive (AR) approach:** This approach uses past energy usage values as inputs to predict future value. (Hyndman, 2018)
- **Moving average (MA) approach:** The MA approach uses past forecast errors as inputs to the forecasting model. (Hyndman, 2018)
- **Exponential smoothing (ES) approach:** This approach uses a weighted average of past energy consumption values to make predictions. (Hyndman, 2018)
- **Artificial neural networks (ANN) approach:** ANNs are used to understand complex relationships between variables and are trained on historical data to make accurate predictions. (Hyndman, 2018)

1st Subtask (AR Approach)

b. Time-Delayed Input/Output (I/O) Matrix

As discussed, the AR approach uses past values of the target variable as input to the neural network. In this task, the target variable at 20:00 hours is used. As mentioned in the objectives, the electricity consumption forecast depends on the load value one week before. Therefore, obtaining the past 7 days of "time-delayed" electricity loads at 20:00 hours is beneficial.

```
t_1 = lag(energyUsage_20, 1)
t_2 = lag(energyUsage_20, 2)
t_3 = lag(energyUsage_20, 3)
t_4 = lag(energyUsage_20, 4)
t_5 = lag(energyUsage_20, 5)
t_6 = lag(energyUsage_20, 6)
t_7 = lag(energyUsage_20, 7)
```

```
t_1                         470 obs. of 1 variable
  $ 20:00: num [1:470] NA 38.9 41.9 40.7 41.9 44 44.3 35.7 43.6 42.5 ...
t_2                         470 obs. of 1 variable
  $ 20:00: num [1:470] NA NA 38.9 41.9 40.7 41.9 44 44.3 35.7 43.6 ...
t_3                         470 obs. of 1 variable
  $ 20:00: num [1:470] NA NA NA 38.9 41.9 40.7 41.9 44 44.3 35.7 ...
t_4                         470 obs. of 1 variable
  $ 20:00: num [1:470] NA NA NA NA 38.9 41.9 40.7 41.9 44 44.3 ...
t_5                         470 obs. of 1 variable
  $ 20:00: num [1:470] NA NA NA NA NA 38.9 41.9 40.7 41.9 44 ...
t_6                         470 obs. of 1 variable
  $ 20:00: num [1:470] NA NA NA NA NA NA 38.9 41.9 40.7 41.9 ...
t_7                         470 obs. of 1 variable
  $ 20:00: num [1:470] NA NA NA NA NA NA NA 38.9 41.9 40.7 ...
```

To create an input-output (IO) matrix, it is required to add input and output data. The input data is the time-delayed electricity load, and the output data is the expected result, which is the electricity usage at 20:00 hours. Several matrices were created to identify the best-performing matrix for the most accurate result.

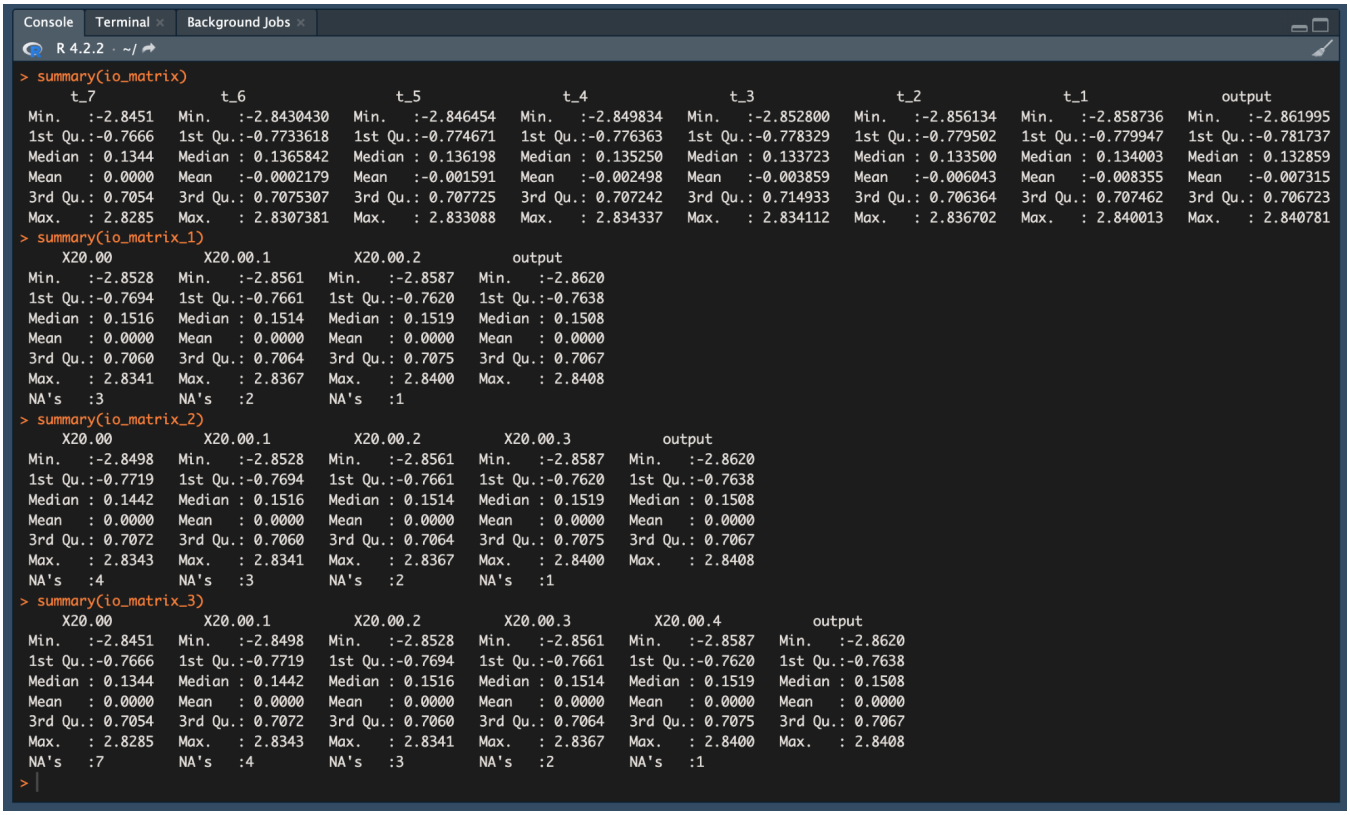
```
input_data <- data.frame(t_7, t_6, t_5, t_4, t_3, t_2, t_1)
input_data_1 <- data.frame(t_3, t_2, t_1)
input_data_2 <- data.frame(t_4, t_3, t_2, t_1)
input_data_3 <- data.frame(t_7, t_4, t_3, t_2, t_1)

output_data <- energyUsage_20

# Normalize the data adn create the IO Matrix (Run only required matrix)
io_matrix <- cbind(scale(input_data), scale(output_data))
io_matrix_1 <- cbind(scale(input_data_1), scale(output_data))
io_matrix_2 <- cbind(scale(input_data_2), scale(output_data))
io_matrix_3 <- cbind(scale(input_data_3), scale(output_data))
```

As a result of the above code, these 4 matrices will be created.

```
io_matrix      num [1:463, 1:8] 0.0988 0.634 0.4199 0.634 1.0087 ...
io_matrix_1    num [1:470, 1:4] NA NA NA 0.098 0.634 ...
io_matrix_2    num [1:470, 1:5] NA NA NA NA 0.0995 ...
io_matrix_3    num [1:470, 1:6] NA NA NA NA NA ...
```

The above image represents the io_matrix data, and the below figure shows the summary of those matrices. These data are already scaled because the matrix's scaling and creation happen in a single code line, as shown in the below snippet.

```
io_matrix <- cbind(scale(input_data), scale(output_data))
```



c. IO Matrix Normalization

In a dataset, data points can be measured or calculated using different methods and measurements. Machine learning models cannot understand these differences, and MLPs are sensitive to the scale of their inputs. This can affect the efficiency and accuracy of the machine-learning model. Therefore, it is essential to normalize all data to the same scale. This is done through the process of normalization.

Normalization is converting data to the same scale (-1 to 1). This can be done in several ways, but using the scale() function in R is a common method, which uses the z-score to normalize the data.

```
io_matrix <- cbind(scale(input_data), scale(output_data))
```

In this code, the input and output data are normalized before they are bound into the io_matrix. Once the data is normalized, it can be used to train the MLP model. The image below shows the normalized matrix data.

Energy Forecast AR.R* | io_matrix

Filter

	X20.00	X20.00.1	X20.00.2	X20.00.3	X20.00.4	X20.00.5	X20.00.6	output
1	NA	NA	NA	NA	NA	NA	NA	0.096992212
2	NA	NA	NA	NA	NA	NA	0.098162074	0.634989927
3	NA	NA	NA	NA	NA	0.097695950	0.635779974	0.419790841
4	NA	NA	NA	NA	0.097956454	0.634755909	0.420732814	0.634989927
5	NA	NA	NA	0.099500223	0.634457594	0.419931925	0.635779974	1.011588328
6	NA	NA	0.100477994	0.635742753	0.419857138	0.634755909	1.012112503	1.065388100
7	NA	0.100900032	0.636283812	0.421245741	0.634457594	1.010697880	1.065874293	-0.476872017
8	0.098764837	0.636162396	0.421961485	0.635742753	1.010008393	1.064403876	-0.475297019	0.939855299
9	0.634013991	0.422057450	0.636283812	1.011112524	1.063658507	-0.475168006	0.940430117	0.742589471
10	0.419914330	0.636162396	1.011347884	1.064736777	-0.474311429	0.939089886	0.743303554	0.993655071
11	0.634013991	1.010846050	1.064928466	-0.472491808	0.938474907	0.742167901	0.994191907	0.724656213
12	1.008688400	1.064372287	-0.471048211	0.939613520	0.741757822	0.992795882	0.725382957	1.172987643
13	1.062213315	-0.470046490	0.939907109	0.742991259	0.992125021	0.724265902	1.173397873	0.276324784
14	-0.472167595	0.939477735	0.743444975	0.993237773	0.723874451	1.171815868	0.277368041	-0.781737389
15	0.937321846	0.743214868	0.993487690	0.725116508	1.170958735	0.276715936	-0.779947162	0.348057813
16	0.741063822	0.993003971	0.725584781	1.171985282	0.276790167	-0.779501983	0.349050427	0.168725241

Showing 1 to 16 of 470 entries, 8 total columns

After clearing the NA values, the dataset now looks like this.

Energy Forecast AR.R* | io_matrix

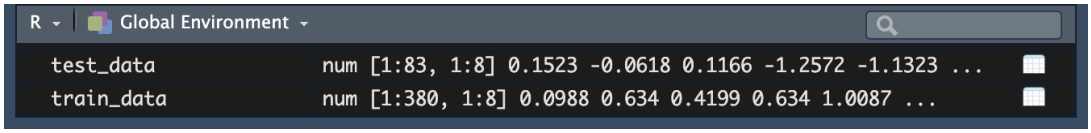
Filter

	X20.00	X20.00.1	X20.00.2	X20.00.3	X20.00.4	X20.00.5	X20.00.6	output
1	0.098764837	0.636162396	0.421961485	0.635742753	1.010008393	1.064403876	-0.475297019	0.939855299
2	0.634013991	0.422057450	0.636283812	1.011112524	1.063658507	-0.475168006	0.940430117	0.742589471
3	0.419914330	0.636162396	1.011347884	1.064736777	-0.474311429	0.939089886	0.743303554	0.993655071
4	0.634013991	1.010846050	1.064928466	-0.472491808	0.938474907	0.742167901	0.994191907	0.724656213
5	1.008688400	1.064372287	-0.471048211	0.939613520	0.741757822	0.992795882	0.725382957	1.172987643
6	1.062213315	-0.470046490	0.939907109	0.742991259	0.992125021	0.724265902	1.173397873	0.276324784
7	-0.472167595	0.939477735	0.743444975	0.993237773	0.723874451	1.171815868	0.277368041	-0.781737389
8	0.937321846	0.743214868	0.993487690	0.725116508	1.170958735	0.276715936	-0.779947162	0.348057813
9	0.741063822	0.993003971	0.725584781	1.171985282	0.276790167	-0.779501983	0.349050427	0.168725241
10	0.990846761	0.725372790	1.172089630	0.278247733	-0.778328742	0.348323931	0.169844461	0.419790841
11	0.723222184	1.171434759	0.279079033	0.776362575	0.348323653	0.169303044	0.420732814	0.886055528

Showing 1 to 11 of 463 entries, 8 total columns

d. Training & Testing the MLP-NN Model

Training and testing are the next steps after data normalization and creating a matrix. As required, the first 380 matrices are added to the training dataset, and the rest of the matrices are added to the testing dataset.



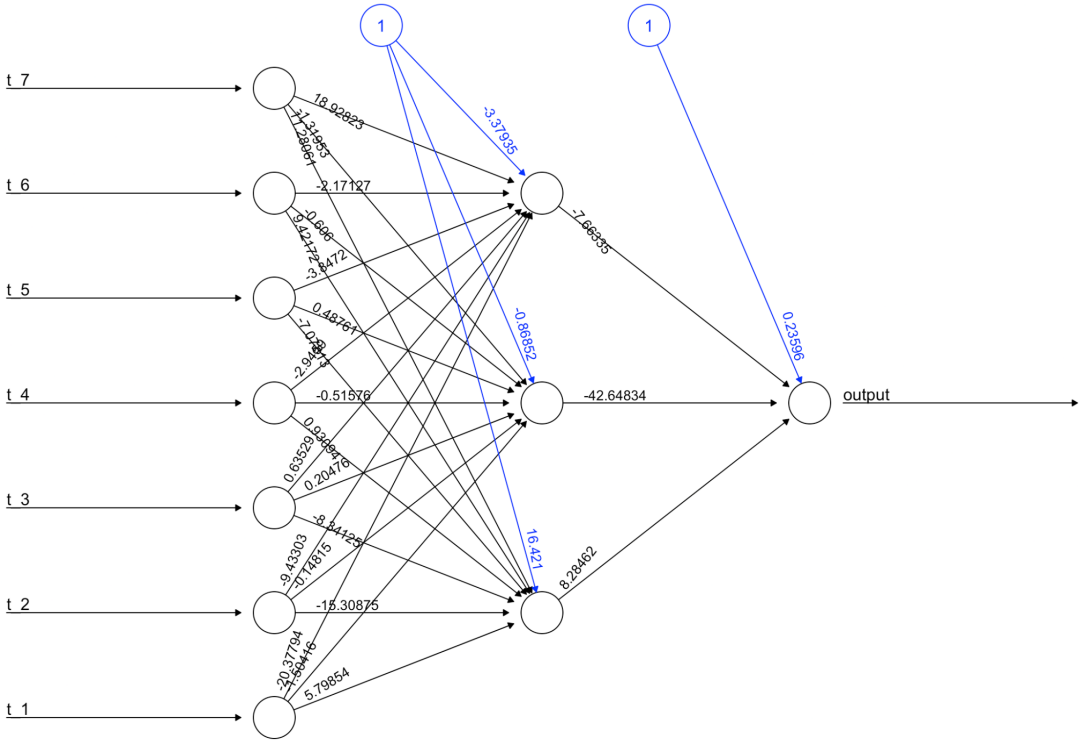
Training the MLP model is the central part of this process. As previously mentioned, there are different IO matrices. Therefore, training a different MLP model for each of them is necessary. This will allow to get a conclusion based on the results and find the most accurate and efficient model.

```
mlp_1 <- neuralnet(output ~ t_3 + t_2 + t_1,
                   data = train_data, hidden = 3,
                   act.fct = "logistic",linear.output = FALSE,stepmax = 1e7)
```

The above code shows a MLP model. It will train the neural network to predict the output variable based on the given input variables in the train_data dataset, using a given activation function in the hidden layers and a non-linear activation function in the output layer.

```
mlp_2 <- neuralnet(output ~ t_3 + t_2 + t_1,
                   data = train_data, hidden = 5, act.fct = "logistic", linear.output = FALSE, stepmax = 1e7)

mlp_3 <- neuralnet(output ~ t_3 + t_2 + t_1,
                   data = train_data, hidden = c(3,2), act.fct = "logistic", linear.output = FALSE, stepmax = 1e7)

mlp_4 <- neuralnet(output ~ t_3 + t_2 + t_1,
                   data = train_data, hidden = c(5,7), act.fct = "logistic", linear.output = FALSE, stepmax = 1e7)

mlp_5 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                   data = train_data, hidden = 3, act.fct = "logistic", linear.output = FALSE, stepmax = 1e7)

mlp_6 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                   data = train_data, hidden = 5, act.fct = "logistic", linear.output = FALSE, stepmax = 1e7)

mlp_7 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                   data = train_data, hidden = c(3,2), act.fct = "logistic", linear.output = FALSE, stepmax = 1e7)

mlp_8 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                   data = train_data, hidden = c(5,7), act.fct = "logistic", linear.output = FALSE, stepmax = 1e7)

mlp_9 <- neuralnet(output ~ t_3 + t_2 + t_1,
                   data = train_data, hidden = 3, act.fct = "tanh", linear.output = FALSE, stepmax = 1e7)

mlp_10 <- neuralnet(output ~ t_3 + t_2 + t_1,
                    data = train_data, hidden = 5, act.fct = "tanh", linear.output = FALSE, stepmax = 1e7)

mlp_11 <- neuralnet(output ~ t_3 + t_2 + t_1,
                    data = train_data, hidden = c(3,2), act.fct = "tanh", linear.output = FALSE, stepmax = 1e7)

mlp_12 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                    data = train_data, hidden = 3, act.fct = "tanh", linear.output = FALSE, stepmax = 1e7)

mlp_13 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                    data = train_data, hidden = 5, act.fct = "tanh", linear.output = FALSE, stepmax = 1e7)

mlp_14 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                    data = train_data, hidden = c(3,2), act.fct = "tanh", linear.output = FALSE, stepmax = 1e7)
```

The code above shows the different types of models. In the original code, the default IOmatrix represents all the different time-delayed data, and when creating the MLP, it is only required to add the time lag name. Furthermore, this model creation scenario considers two activation functions (logistic and tanh), input variables of small and large sizes (3 and 7), and the option of using one or two hidden layers with small and large numbers of neurons for a better evaluation. This will be further discussed in the Performance Comparison.
Below is the plot for MLP-5 (7 inputs, 3 hidden layers), which performs well among all other MLPs.



The MLP model is trained using the neural net. It is trained by using the output column as the prediction target and all other data in the train_data dataset as past data parameters. The model finds the pattern of the data and trains itself according to the given hidden layer counts and activation function. It then predicts the results accordingly.

Before conducting further analysis process, it is necessary to select the MLP that is required. For example, if it is required to analyze MLP-5 further, do the rest of the process for it or calculate something related to it, the “mlp_5” variable must be assigned to the “mlp” variable.

```
mlp <- mlp_5  #Select the mlp to do the rest of the process
```

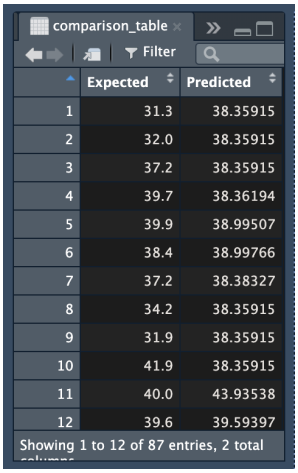
The code snippet below is the one that does the prediction.

```
test_pred <- neuralnet::compute(mlp, test_data[, -ncol(test_data)])
```

At the moment, all data is scaled. The data points must be denormalized to compare predicted and expected values. In the denormalization process, scaled data will be converted back to their original values or the predicted original values. Because the training process is complete, it is not necessary to keep the data in normalized form.

```r
# Backscale the predictions back to the original scale
test_pred_rescaled<- test_pred$net.result * sd(output_data$output) + mean(output_data$output)

# Backscale the output back to the original scale
test_data_df <- as.data.frame(test_data)
test_output_rescaled <- test_data_df$output * sd(output_data$output) + mean(output_data$output)
```

After de-normalization, the predicted results can be compared with the expected results. The images below show the comparison table on the left, which represents the expected and predicted data numerically, and the expected vs. predicted plot on the right, which represents the graph of the expected and predicted results.

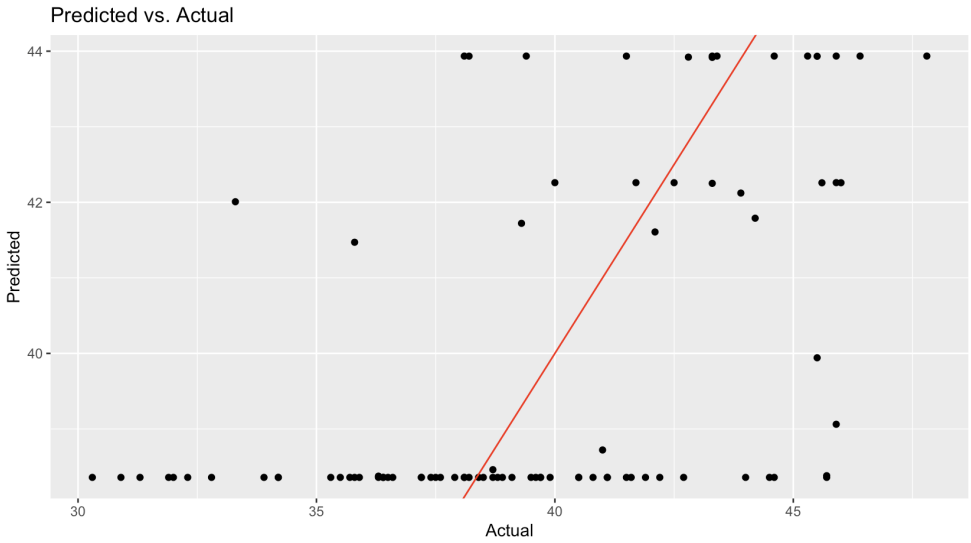


comparison_table

Filter

	Expected	Predicted
1	31.3	38.35915
2	32.0	38.35915
3	37.2	38.35915
4	39.7	38.36194
5	39.9	38.99507
6	38.4	38.99766
7	37.2	38.38327
8	34.2	38.35915
9	31.9	38.35915
10	41.9	38.35915
11	40.0	43.93538
12	39.6	39.59397

Showing 1 to 12 of 87 entries, 2 total



It is important to evaluate the performance of a model. To do this, RMSE, MAE, MAPE, and sMAPE are commonly used metrics. These metrics can help to get a better understanding of the model's performance.

```r
mse <- mean((test_pred_rescaled - test_output_rescaled)^2)
rmse <- sqrt(mse)
mae <- mean(abs(test_pred_rescaled - test_output_rescaled))
mape <- mean(abs((test_output_rescaled - test_pred_rescaled)/test_output_rescaled)) * 100
smape <- mean(2 * abs(test_pred_rescaled - test_output_rescaled) / (abs(test_pred_rescaled) +
         abs(test_output_rescaled))) * 100
```

```
> # Print model performance
> cat("MSE:", mse, "\n")
MSE: 13.09001
> cat("RMSE:", rmse, "\n")
RMSE: 3.618012
> cat("MAE:", mae, "\n")
MAE: 2.844775
> cat("MAPE:", mape, "% \n")
MAPE: 7.431512 %
> cat("sMAPE:", smape, "% \n")
sMAPE: 7.273791 %
>
```

e. Standard Statistical Indices

It is important to identify eros and the accuracy performance in a machine learning model. So, There are several standard statistical indices used in machine learning to evaluate the model performance:

- **Root mean squared error (RMSE):** RSME is one of the most common error metrics. It is calculated by taking the square root of the average of the squared differences between the actual and predicted values. RMSE penalizes large errors more heavily than small errors and is sensitive to outliers. A low RMSE value indicates better performance.

$$RSME = \sqrt{\frac{\sum(Actual - Predicted)^2}{n}}$$

- **Mean absolute error (MAE):** This is another common error measure. It is calculated by taking the average of the absolute differences between the actual and predicted values. MAE is less sensitive to outliers than RMSE, and a lower MAE value indicates better performance.

$$MAE = \frac{\sum| Actual - Predicted |}{n}$$

- **Mean absolute percentage error (MAPE):** MAPE is calculated by taking the average of the absolute percentage differences between the predicted and actual values. MAPE is easy to interpret as a percentage. A low MAPE value represents better performance.

$$MAPE = \frac{1}{n}\sum\left|\frac{Actual - Predicted}{Actual}\right|$$

- **Symmetric mean absolute percentage error (sMAPE):** This is calculated by taking the average of the absolute percentage differences between the predicted values and the actual values, and dividing it by the average of the absolute percentage differences between the actual values and the mean of the actual values. A low sMAPE value represents better performance.

$$sMAPE = \frac{1}{n}\sum\frac{|Predicted - Actual|}{\frac{|Actual| + |Predicted|}{2}}$$

f. Performance Comparison

In this scenario, two activation functions (logistic and tanh) are considered. Additionally, there are input variables of relatively small and slightly large sizes (3: [t_3, t_2, t_1] and 7: [t_7, t_6, t_5, t_4, t_3, t_2, t_1]). This is also considered. The option of using one hidden layer (with Small num and Big num) or two hidden layers (with Small num and Big num) is also available.
In this case, all of these variables will be compared with other groups of variables to get a better performance conclusion.

MLP	Input Variables	Activation Function	Input Layers	Hidden Layers	RSME	MAE	MAPE (%)	sMAPE (%)
MLP-01	t_3, t_2, t_1	logistic	3	3	3.47	2.80	7.30	7.16
MLP-02	t_3, t_2, t_1	logistic	3	5	3.64	3.01	7.78	7.67
MLP-03	t_3, t_2, t_1	logistic	3	3,2	3.46	2.73	7.14	7.00
MLP-04	t_3, t_2, t_1	logistic	3	5,7	3.69	2.89	7.53	7.40
MLP-05	t_7, t_6, t_5, t_4, t_3, t_2, t_1	logistic	7	3	3.51	2.58	6.80	6.56
MLP-06	t_7, t_6, t_5, t_4, t_3, t_2, t_1	logistic	7	5	3.61	2.72	7.14	6.89
MLP-07	t_7, t_6, t_5, t_4, t_3, t_2, t_1	logistic	7	3,2	3.51	2.59	6.82	6.58
MLP-08	t_7, t_6, t_5, t_4, t_3, t_2, t_1	logistic	7	5,7	3.87	2.89	7.61	7.31
MLP-09	t_3, t_2, t_1	tanh	3	3	3.67	3.00	7.71	7.72
MLP-10	t_3, t_2, t_1	tanh	3	5	4.17	3.26	8.36	8.40
MLP-11	t_3, t_2, t_1	tanh	3	3,2	3.87	3.01	7.75	7.74
MLP-12	t_7, t_6, t_5, t_4, t_3, t_2, t_1	tanh	7	3	3.70	2.90	7.46	7.40
MLP-13	t_7, t_6, t_5, t_4, t_3, t_2, t_1	tanh	7	5	4.17	3.26	8.36	8.40
MLP-14	t_7, t_6, t_5, t_4, t_3, t_2, t_1	tanh	7	3,2	3.89	2.93	7.45	7.55

*mlp-16 ; t_7, t_6, t_5, t_4, t_3, t_2, t_1 ; tanh ; 7 input layers, (5,7) hidden layers getting error
*mlp-12; t_3, t_2, t_1 ; tanh ; 3 input layer ; (5,7) hidden layers getting error

g. The Efficiency of Hidden-Layer Networks

To check the efficiency of one-hidden-layer and two-hidden-layer networks, we need to calculate the total number of weight parameters per network. The number of weight parameters equals the number of connections between neurons in the network.

For the one-hidden layer networks, the total number of weight parameters is follows:

- MLP-01: (3 inputN * 3 hiddenN) + (3 hiddenN * 1 outputN) = 12
- MLP-02: (3 inputN * 5 hiddenN) + (5 hiddenN * 1 outputN) = 20
- MLP-05: (7 inputN * 3 hiddenN) + (3 hiddenN * 1 outputN) = 24
- MLP-06: (7 inputN * 5 hiddenN) + (5 hiddenN * 1 outputN) = 40
- MLP-09: (3 inputN * 3 hiddenN) + (3 hiddenN * 1 outputN) = 12
- MLP-10: (3 inputN * 5 hiddenN) + (5 hiddenN * 1 outputN) = 20
- MLP-12: (7 inputN * 3 hiddenN) + (3 hiddenN * 1 outputN) = 24
- MLP-13: (7 inputN * 5 hiddenN) + (5 hiddenN * 1 outputN) = 40

For the two-hidden layer networks, the total number of weight parameters is calculated as follows:

- MLP-03: (3 inputN * 3 hiddenN) + (3 hiddenN * 2 hiddenN) + (2 hiddenN * 1 outputN) = 17
- MLP-04: (3 inputN * 5 hiddenN) + (5 hiddenN * 7 hiddenN) + (7 hiddenN * 1 outputN) = 59
- MLP-07: (7 inputN * 3 hiddenN) + (3 hiddenN * 2 hiddenN) + (2 hiddenN * 1 outputN) = 23
- MLP-08: (7 inputN * 5 hiddenN) + (5 hiddenN * 7 hiddenN) + (7 hiddenN * 1 outputN) = 83
- MLP-11: (3 inputN * 3 hiddenN) + (3 hiddenN * 2 hiddenN) + (2 hiddenN * 1 outputN) = 17
- MLP-14: (7 inputN * 3 hiddenN) + (3 hiddenN * 2 hiddenN) + (2 hiddenN * 1 outputN) = 23

Based on the above results, one-hidden layer neural networks have fewer weight parameters than two-layer neural networks. This means that one-hidden-layer NNs are more efficient than two-hidden-layer NNs.

Additionally, according to the comparison table, one-hidden-layer networks are more accurate than two-hidden-layer networks. In the table, the cells with the lowest error values are coloured yellow. MLP-05 has the lowest MAE, MAPE, and sMAPE values. MLP-03, which is a two-hidden-layer network, has the lowest RSME value. However, the RSME value of MLP-05 is much closer to the value of MLP-03. Therefore, MLP-05 has the lowest error measurements overall.

Therefore, using a one-hidden layer neural network is preferable in this case.

2nd Subtask (NARX Approach)

The NARX approach uses past values of the target variable and past values of the exogenous variables as input to the neural network, while the AR approach uses only the past values of the target variable. Both are used to predict the next value of the target variable.

In the previous task, only the 20:00 hour was considered. However, for the NARX approach, it is recommended to use all 18:00, 19:00, and 20:00 hours to input more information and understand more complex relationships between variables to training the model.

```
energyUsage <- read_excel("/Users/rayaan/Edu/ML & DS/MLP-NN Part/uow_consumption.xlsx") %>%
 rename("18:00" = 2, "19:00" = 3, "20:00" = 4)  %>%
 select("18:00", "19:00", "20:00")
```

h. NARX Findings

As discussed in the first subtask (AR Approach), most of the steps are happening to the pre-processed dataset in the same way as in the NARX approach. However, there are slight changes in the time lag, io_matrix, and also in the model.

More time lags are considered in this case than in the AR approach.

```
t1_18 = lag(energyUsage$`18:00`, 1)
t2_18 = lag(energyUsage$`18:00`, 2)
t3_18 = lag(energyUsage$`18:00`, 3)
t4_18 = lag(energyUsage$`18:00`, 4)
t7_18 = lag(energyUsage$`18:00`, 7)
t1_19 = lag(energyUsage$`19:00`, 1)
t2_19 = lag(energyUsage$`19:00`, 2)
t3_19 = lag(energyUsage$`19:00`, 3)
t4_19 = lag(energyUsage$`19:00`, 4)
t7_19 = lag(energyUsage$`19:00`, 7)
t1_20 = lag(energyUsage$`20:00`, 1)
t2_20 = lag(energyUsage$`20:00`, 2)
t3_20 = lag(energyUsage$`20:00`, 3)
t4_20 = lag(energyUsage$`20:00`, 4)
t7_20 = lag(energyUsage$`20:00`, 7)
```

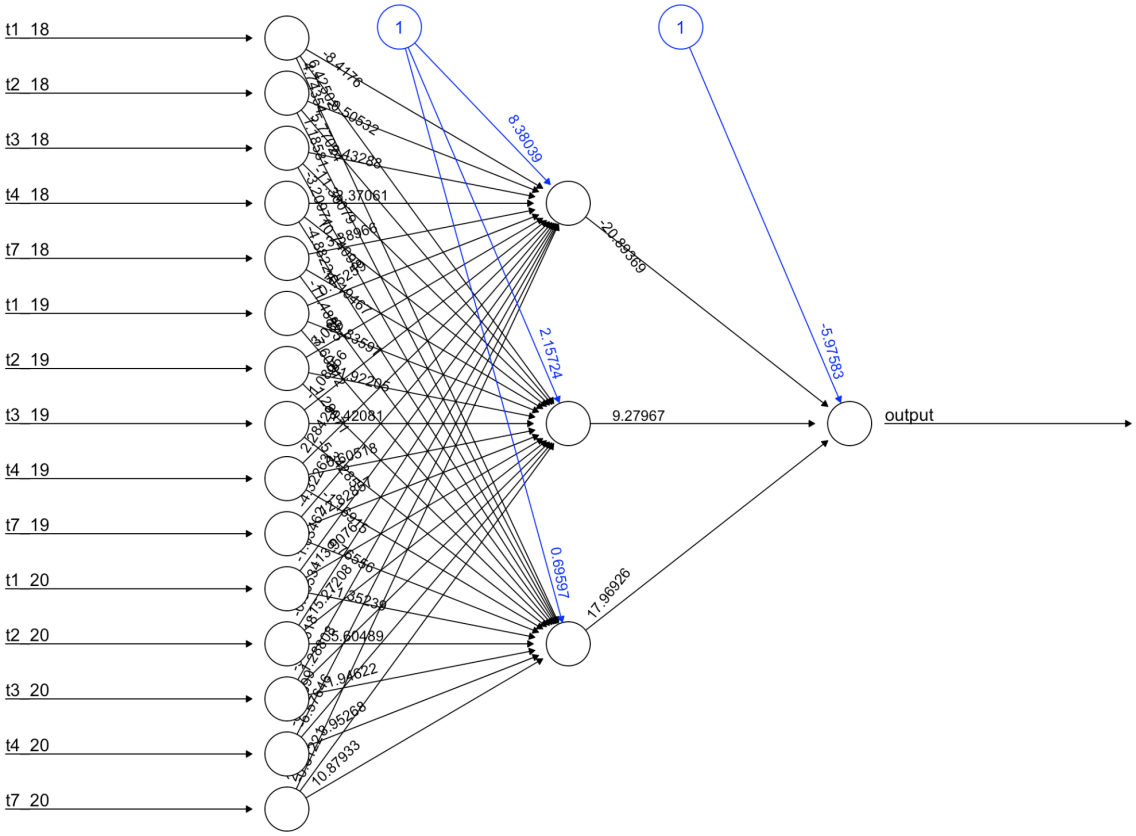
And according to that, the IO matrices also slightly changed.

```
input_data_1 <- data.frame(t1_18, t2_18, t3_18, t4_18, t7_18, t1_19, t2_19, t3_19,
                      t4_19, t7_19, t1_20, t2_20, t3_20, t4_20, t7_20)
input_data_2 <- data.frame(t1_18, t1_19, t1_20)
input_data_3 <- data.frame(t1_18, t2_18, t3_18, t4_18, t1_19, t2_19, t3_19, t4_19)

io_matrix <- cbind(scale(input_data_1), scale(output_data))
io_matrix_1 <- cbind(scale(input_data_2), scale(output_data))
io_matrix_2 <- cbind(scale(input_data_3), scale(output_data))
```

As requested in the objectives, 5 MLP models were created.

```
mlp_1 <- neuralnet(output ~  t1_18 + t2_18 + t3_18 + t4_18 + t7_18 + t1_19 + t2_19 +
                     t3_19 + t4_19 + t7_19 + t1_20 + t2_20 + t3_20 + t4_20 + t7_20,
                   data = train_data, hidden = 3,
                   act.fct = "logistic",
                   linear.output = FALSE,
                   stepmax = 1e5)

mlp_2 <- neuralnet(output ~ t1_18 + t2_18 + t3_18 + t4_18 + t1_19 + t2_19 + t3_19 + t4_19,
                   data =train_data, hidden=c(3,2), act.fct="logistic",linear.output = FALSE, stepmax =
1e5)

mlp_3 <- neuralnet(output ~ t1_18 + t2_18 + t3_18 + t4_18 + t1_19 + t2_19 + t3_19 + t4_19,
                   data = train_data, hidden = 5, act.fct = "tanh",linear.output = FALSE, stepmax = 1e5)

mlp_4 <- neuralnet(output ~ t1_18 + t2_18 + t3_18 + t4_18 + t7_18 + t1_19 + t2_19 +
                     t3_19 + t4_19 + t7_19 + t1_20 + t2_20 + t3_20 + t4_20 + t7_20,
                   data =train_data, hidden =c(5,7), act.fct ="logistic",linear.output= FALSE, stepmax
=1e5)

mlp_5 <- neuralnet(output ~ t1_18 + t1_19 + t1_20,
                   data = train_data, hidden = 3, act.fct = "tanh",linear.output = FALSE, stepmax = 1e5)
```

Below image represents the plot for the mlp_1



Comparition Table for the NARX approach

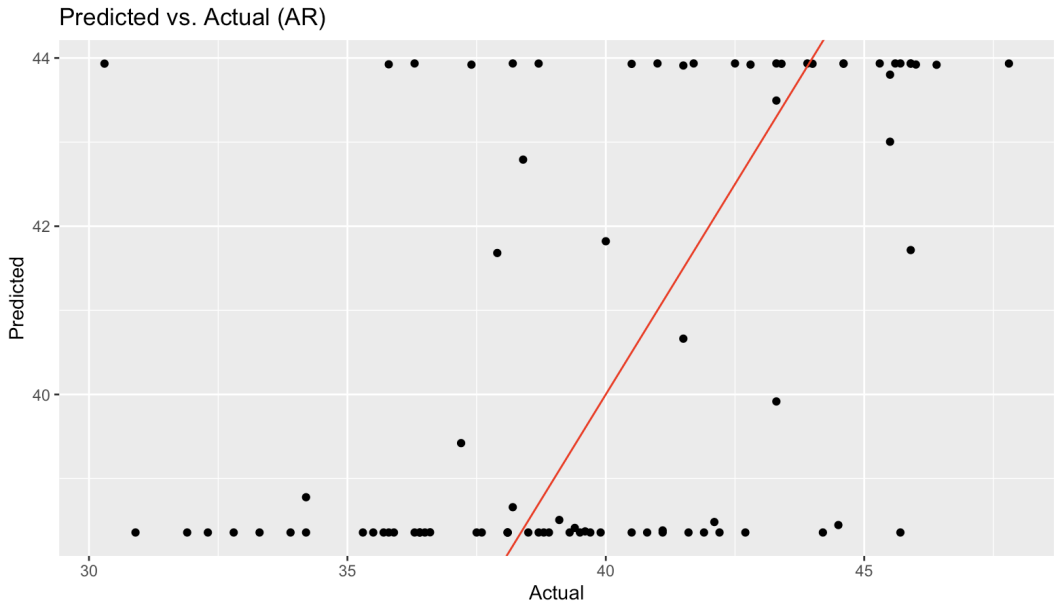
MLP	Input Variables	Activation Function	Input Layers	Hidden Layers	RSME	MAE	MAPE (%)	sMAPE (%)
MLP-01	t1_18, t2_18, t3_18, t4_18, t7_18, t1_19, t2_19, t3_19, t4_19, t7_19, t1_20, t2_20, t3_20, t4_20, t7_20	logistic	15	3	3.18	2.52	6.57	6.43
MLP-02	t1_18, t2_18, t3_18, t4_18, t1_19, t2_19, t3_19, t4_19	logistic	8	3,2	3.44	2.72	7.09	6.93
MLP-03	t1_18, t2_18, t3_18, t4_18, t1_19, t2_19, t3_19, t4_19	tanh	8	5	4.11	3.31	8.48	8.50
MLP-04	t1_18, t2_18, t3_18, t4_18, t7_18, t1_19, t2_19, t3_19, t4_19, t7_19, t1_20, t2_20, t3_20, t4_20, t7_20	logistic	15	5,7	3.29	2.58	6.70	6.56
MLP-05	t1_18, t1_19, t1_20	tanh	3	3	3.44	2.73	6.89	6.97

In this table, the cells with a yellow background indicate the lowest error in that column compared to the other models. It seems like the best performance is given by MLP-01, because it has the lowest error rate in both RSME, MAE, MAPE, and sMAPE across all other MLPs.

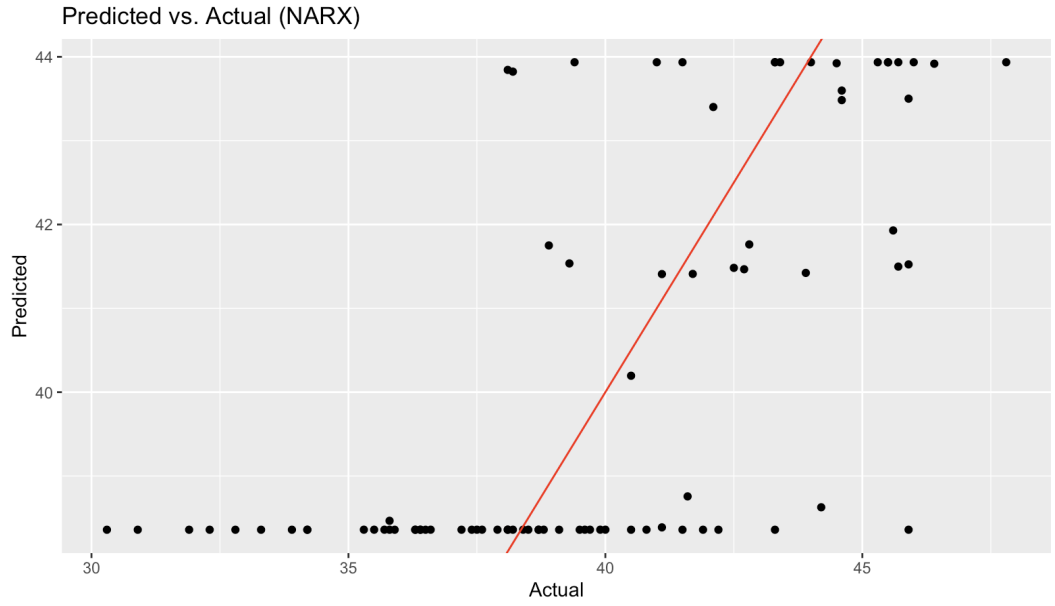
Therefore, in conclusion, the io_matrix 1 with 15 input layers and 3 layers is performing better.

i. Best MLP Network

Below is the scatter plot image of the best-performing AR model according to the comparison table. (MLP-05; logistic; 7 inputs; 3 hidden layers)



And below is the scatter plot that performed well for the NARX model according to the comparison table. (MLP-05; logistic; 15 inputs; 3 hidden layers)



Based on the results of the scatter plot, both models seem similar. However, upon closer look, the data points of the NARX model data points are more closely to the regression (trend) line than the AR model.

Therefore, the best MLP network will be a NARX model with a single hidden layer.

Appendix

1. Clustering Code

```
library(readxl)
library(cluster)
library(NbClust)
library(factoextra)

# Read the Excel file
vehicles <- read_excel("/Users/rayaan/Edu/ML & DS/ML Coursework/PC Part/vehicles.xlsx")

# <----------------------------- 1 - A ----------------------------->

# Remove the first and last columns
vehicles <- vehicles[, -c(1, ncol(vehicles))]

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

# Scale the data
vehicles <- scale(vehicles)

# Create a boxplot of the data
boxplot(vehicles, main = "Data with Outliers", ylab = "Data")

# Identify the outliers and remove them
outliers <- apply(vehicles, 1, function(x) any(x > 3 | x < -3))
vehicles <- subset(vehicles, !outliers)

summary(outliers)

boxplot(vehicles, main = "Data without Outliers", ylab = "Data")



# <----------------------------- 1 - B ----------------------------->

# NBclust
NBclust <- NbClust(vehicles, distance = "euclidean", min.nc = 2, max.nc = 6, method =
"kmeans")

# Elbow method
fviz_nbclust(vehicles, kmeans, method = "wss")
```

```r
# Gap statistics
fviz_nbclust(vehicles, kmeans, method = "gap_stat")

# Determine the optimal number of clusters using the silhouette method
fviz_nbclust(vehicles, kmeans, method = "silhouette")



# <---------------------------- 1 - C ----------------------------->

# Set the number of clusters
k <- 3

# Perform k-means clustering
km <- kmeans(vehicles, centers = k)

# Print the cluster centers & cluster assignments for each data point
km$centers
km$cluster

# Create a plot of the clustering results
fviz_cluster(km, data = vehicles,
             palette = c("#2E9FDF", "#00AFBB", "#E7B800"),
             geom = "point",
             ellipse.type = "convex",
             ggtheme = theme_bw(),
             main = "K-means Clustering Results"
)

# Compute the WSS, BSS and TSS
wss <- sum(km$withinss)
bss <- sum(km$betweenss)
tss <- wss + bss

# Print the ratio of BSS over TSS
cat("WSS :", wss)
cat("BSS :", bss)
cat("TSS :", tss)
cat("Ratio of BSS over TSS:", bss / tss)



# <---------------------------- 1 - D ----------------------------->
```

```r
# Generate and plot the silhouette plot
sil_width <- silhouette(km$cluster, dist(vehicles))
plot(sil_width, main = "Silhouette Plot for K-means Clustering")

# Calculate and print the average silhouette width
cat("Average Silhouette Width:", mean(sil_width[,k]))

# Create a colored silhouette plot
fviz_silhouette(sil_width, palette = c("#2E9FDF", "#00AFBB", "#E7B800"), ggtheme =
theme_bw(),
                main = "Silhouette Plot of Clustering Results")
```

2. PCA Code

```r
library(readxl)
library(cluster)
library(NbClust)
library(factoextra)
library(fpc)

vehicles <- read_excel("/Users/rayaan/Edu/ML & DS/ML Coursework/PC Part/vehicles.xlsx") #
Read the Excel file
vehicles <- vehicles[, -c(1, ncol(vehicles))]  # Remove the first and last columns
vehicles <- na.omit(vehicles) # Remove rows with missing values
vehicles <- scale(vehicles) # Scale the data
outliers <- apply(vehicles, 1, function(x) any(x > 3 | x < -3)) # Identify the outliers
vehicles <- subset(vehicles, !outliers) # Remove outliers



# <---------------------------- 1 - E ---------------------------->

# Perform PCA
pca <- prcomp(vehicles)

# Extract eigenvalues and eigenvectors
eigenvalues <- pca$sdev^2
eigenvectors <- pca$rotation
```

```r
# Calculate the cumulative proportion of variance explained
cum_prop_var <- cumsum(eigenvalues / sum(eigenvalues))

# Print the number of components required to explain 92% variance
n_components <- which(cum_prop_var >= 0.92)[1]
cat("Number of components to explain 92% variance:", n_components, "\n")

# Mark the plot at 92% cumulative proportion of variance explained
plot(cum_prop_var, type = "b", xlab = "Number of PC", ylab = "Cumulative Proportion")
abline(h = 0.92, v = n_components, col = "red", lty = "dashed")

# Create a new dataset with the chosen principal components
vehicles_pca <- predict(pca, newdata = vehicles)[, 1:n_components]

vehicles_pca <- as.data.frame(vehicles_pca)


# <----------------------------- 1 - F ----------------------------->

# NBclust
NBclust <- NbClust(vehicles_pca, distance = "euclidean", min.nc = 2, max.nc = 6, method =
"kmeans")

# Perform elbow method and plot WSS for different values of k
fviz_nbclust(vehicles_pca, kmeans, method = "wss")

# Gap statistics
fviz_nbclust(vehicles_pca, kmeans, method = "gap_stat")

# Determine the optimal number of clusters using the silhouette method
fviz_nbclust(vehicles_pca, kmeans, method = "silhouette")



# <----------------------------- 1 - G ----------------------------->

# Choose the best k from the automated methods
k_pca <- 3

# Perform k-means clustering on the PCA-based dataset
km_pca <- kmeans(vehicles_pca, centers = k_pca)

# Print the cluster centers & cluster assignments for each data point
km_pca$centers
km_pca$cluster
```

```r
# Create a plot of the clustering results
fviz_cluster(km_pca, data = vehicles_pca, palette = c("#2E9FDF", "#00AFBB", "#E7B800"),
             geom = "point", ellipse.type = "convex", ggtheme = theme_bw(),
             main = "K-means Clustering Results"
)

# Compute the WSS, BSS and TSS
wss_pca <- sum(km_pca$withinss)
bss_pca <- sum(km_pca$betweenss)
tss_pca <- wss_pca + bss_pca

# Print the ratio of BSS over TSS
cat("WSS :", wss_pca)
cat("BSS :", bss_pca)
cat("TSS :", tss_pca)
cat("Ratio of BSS over TSS:", bss_pca / tss_pca)



# <----------------------------- 1 - H ------------------------------>

# Generate and plot the silhouette plot
sil_width_pca <- silhouette(km_pca$cluster, dist(vehicles))
plot(sil_width_pca, main = "Silhouette Plot for K-means Clustering")

# Calculate and print the average silhouette width
cat("Average Silhouette Width:", mean(sil_width_pca[,k_pca]))

# Create a colored silhouette plot
fviz_silhouette(sil_width_pca, palette = c("#2E9FDF", "#00AFBB", "#E7B800"), ggtheme =
theme_bw(),
                main = "Silhouette Plot of Clustering Results")



# Calculate the Calinski-Harabasz Index
ch_index <- calinhara(vehicles_pca, km_pca$cluster)

# Print the CH index value
cat("Calinski-Harabasz Index:", ch_index)

# Visualize the clustering results
fviz_cluster(km_pca, data = vehicles_pca)
```

3. AR Model Code

```
library(readxl)
library(dplyr)
library(neuralnet)
library(ggplot2)

# Import data & Rename columns
energyUsage_20 <- read_excel("/Users/rayaan/Edu/ML & DS/ML Coursework/MLP-NN
Part/uow_consumption.xlsx") %>%
 rename("18:00" = 2, "19:00" = 3, "20:00" = 4) %>%
 select("20:00")

# time-delayed data
t_1 = lag(energyUsage_20, 1)
t_2 = lag(energyUsage_20, 2)
t_3 = lag(energyUsage_20, 3)
t_4 = lag(energyUsage_20, 4)
t_5 = lag(energyUsage_20, 5)
t_6 = lag(energyUsage_20, 6)
t_7 = lag(energyUsage_20, 7)

# Create input/output matrices with time-delayed electricity loads
input_data <- data.frame(t_7, t_6, t_5, t_4, t_3, t_2, t_1)
input_data_1 <- data.frame(t_3, t_2, t_1)
input_data_2 <- data.frame(t_4, t_3, t_2, t_1)
input_data_3 <- data.frame(t_7, t_4, t_3, t_2, t_1)

colnames(input_data) <- paste0("t_", 7:1)
output_data <- energyUsage_20
colnames(output_data) <- "output"

# Normalize the data adn create the IO Matrix (Run only required matrix)
io_matrix <- cbind(scale(input_data), scale(output_data))
io_matrix_1 <- cbind(scale(input_data_1), scale(output_data))
io_matrix_2 <- cbind(scale(input_data_2), scale(output_data))
io_matrix_3 <- cbind(scale(input_data_3), scale(output_data))

io_matrix <- na.omit(io_matrix)
summary(io_matrix)

# Subset the data into training and testing sets
train_data <- io_matrix[1:380, ]
test_data <- io_matrix[381:nrow(io_matrix), ]
```

```r
# Train MLP model with modified parameters
mlp_1 <- neuralnet(output ~ t_3 + t_2 + t_1,
                   data = train_data, hidden = 3,
                   act.fct = "logistic",
                   linear.output = FALSE,
                   stepmax = 1e7)

mlp_2 <- neuralnet(output ~ t_3 + t_2 + t_1,
                    data = train_data, hidden = 5, act.fct = "logistic", linear.output = FALSE,
stepmax = 1e7)

mlp_3 <- neuralnet(output ~ t_3 + t_2 + t_1,
                    data = train_data, hidden = c(3,2), act.fct = "logistic", linear.output =
FALSE, stepmax = 1e7)

mlp_4 <- neuralnet(output ~ t_3 + t_2 + t_1,
                    data = train_data, hidden = c(5,7), act.fct = "logistic", linear.output =
FALSE, stepmax = 1e7)

mlp_5 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                    data = train_data, hidden = 3, act.fct = "logistic", linear.output = FALSE,
stepmax = 1e7)

mlp_6 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                    data = train_data, hidden = 5, act.fct = "logistic", linear.output = FALSE,
stepmax = 1e7)

mlp_7 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                    data = train_data, hidden = c(3,2), act.fct = "logistic", linear.output =
FALSE, stepmax = 1e7)

mlp_8 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                    data = train_data, hidden = c(5,7), act.fct = "logistic", linear.output =
FALSE, stepmax = 1e7)

mlp_9 <- neuralnet(output ~ t_3 + t_2 + t_1,
                    data = train_data, hidden = 3, act.fct = "tanh", linear.output = FALSE,
stepmax = 1e7)

mlp_10 <- neuralnet(output ~ t_3 + t_2 + t_1,
                     data = train_data, hidden = 5, act.fct = "tanh", linear.output = FALSE,
stepmax = 1e7)

mlp_11 <- neuralnet(output ~ t_3 + t_2 + t_1,
```

```
                  data = train_data, hidden = c(3,2), act.fct = "tanh", linear.output =
FALSE, stepmax = 1e7)

mlp_12 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                  data = train_data, hidden = 3, act.fct = "tanh", linear.output = FALSE,
stepmax = 1e7)

mlp_13 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                  data = train_data, hidden = 5, act.fct = "tanh", linear.output = FALSE,
stepmax = 1e7)

mlp_14 <- neuralnet(output ~ t_7 + t_6 + t_5 + t_4 + t_3 + t_2 + t_1,
                  data = train_data, hidden = c(3,2), act.fct = "tanh", linear.output =
FALSE, stepmax = 1e7)

#Select the mlp to do the rest of the process
mlp <- mlp_5

# Make predictions on the test data
test_pred <- neuralnet::compute(mlp, test_data[, -ncol(test_data)])

# Backscale the predictions back to the original scale
test_pred_rescaled <- test_pred$net.result * sd(output_data$output) +
mean(output_data$output)

# Backscale the output back to the original scale
test_data_df <- as.data.frame(test_data)
test_output_rescaled <- test_data_df$output * sd(output_data$output) +
mean(output_data$output)

# Create actual and predicted comparison table
comparison_table <- data.frame(Expected = test_output_rescaled, Predicted =
test_pred_rescaled)


# Evaluate model performance
mse <- mean((test_pred_rescaled - test_output_rescaled)^2)
rmse <- sqrt(mse)
mae <- mean(abs(test_pred_rescaled - test_output_rescaled))
mape <- mean(abs((test_output_rescaled - test_pred_rescaled)/test_output_rescaled)) * 100
smape <- mean(2 * abs(test_pred_rescaled - test_output_rescaled) / (abs(test_pred_rescaled) +
abs(test_output_rescaled))) * 100


# Print model performance
```

```r
cat("MSE:", mse, "\n")
cat("RMSE:", rmse, "\n")
cat("MAE:", mae, "\n")
cat("MAPE:", mape, "% \n")
cat("sMAPE:", smape, "% \n")

plot(mlp)


# Create a data frame with predicted and observed values
plot_data <- data.frame(Predicted = test_pred_rescaled, Expected = test_output_rescaled)

# Create a scatter plot with a 45-degree line
ggplot(plot_data, aes(x = Expected, y = Predicted)) +
 geom_point() +
 geom_abline(intercept = 0, slope = 1, color = "red") +
 labs(x = "Actual", y = "Predicted", title = "Predicted vs. Actual (AR)")
```

4. NARX Model Code

```r
library(readxl)
library(dplyr)
library(neuralnet)
library(ggplot2)

# Import data & Rename columns
energyUsage <- read_excel("/Users/rayaan/Edu/ML & DS/ML Coursework/MLP-NN
Part/uow_consumption.xlsx") %>%
 rename("18:00" = 2, "19:00" = 3, "20:00" = 4)  %>%
 select("18:00", "19:00", "20:00")

# time-delayed data
t1_18 = lag(energyUsage$`18:00`, 1)
t2_18 = lag(energyUsage$`18:00`, 2)
t3_18 = lag(energyUsage$`18:00`, 3)
t4_18 = lag(energyUsage$`18:00`, 4)
t7_18 = lag(energyUsage$`18:00`, 7)
t1_19 = lag(energyUsage$`19:00`, 1)
```

```r
t2_19 = lag(energyUsage$`19:00`, 2)
t3_19 = lag(energyUsage$`19:00`, 3)
t4_19 = lag(energyUsage$`19:00`, 4)
t7_19 = lag(energyUsage$`19:00`, 7)
t1_20 = lag(energyUsage$`20:00`, 1)
t2_20 = lag(energyUsage$`20:00`, 2)
t3_20 = lag(energyUsage$`20:00`, 3)
t4_20 = lag(energyUsage$`20:00`, 4)
t7_20 = lag(energyUsage$`20:00`, 7)

# Create input/output matrices with time-delayed electricity loads

input_data_1 <- data.frame(t1_18, t2_18, t3_18, t4_18, t7_18, t1_19, t2_19, t3_19,
                           t4_19, t7_19, t1_20, t2_20, t3_20, t4_20, t7_20)
input_data_2 <- data.frame(t1_18, t1_19, t1_20)
input_data_3 <- data.frame(t1_18, t2_18, t3_18, t4_18, t1_19, t2_19, t3_19, t4_19)

input_data = input_data_1 #Select the input data

output_data <- energyUsage["20:00"]
colnames(output_data) <- "output"

# Normalize the data
io_matrix <- cbind(scale(input_data_1), scale(output_data))
io_matrix_1 <- cbind(scale(input_data_2), scale(output_data))
io_matrix_2 <- cbind(scale(input_data_3), scale(output_data))

io_matrix <- na.omit(io_matrix)

# Subset the data into training and testing sets
train_data <- io_matrix[1:380, ]
test_data <- io_matrix[381:nrow(io_matrix), ]

# Train MLP model with modified parameters
mlp_1 <- neuralnet(output ~  t1_18 + t2_18 + t3_18 + t4_18 + t7_18 + t1_19 + t2_19 +
                     t3_19 + t4_19 + t7_19 + t1_20 + t2_20 + t3_20 + t4_20 + t7_20,
                   data = train_data, hidden = 3,
                   act.fct = "logistic",
                   linear.output = FALSE,
                   stepmax = 1e5)

mlp_2 <- neuralnet(output ~ t1_18 + t2_18 + t3_18 + t4_18 + t1_19 + t2_19 + t3_19 + t4_19,
                   data = train_data, hidden = c(3,2), act.fct = "logistic",
                   linear.output = FALSE, stepmax = 1e5)
```

```r
mlp_3 <- neuralnet(output ~ t1_18 + t2_18 + t3_18 + t4_18 + t1_19 + t2_19 + t3_19 + t4_19,
                   data = train_data, hidden = 5, act.fct = "tanh",
                   linear.output = FALSE, stepmax = 1e5)

mlp_4 <- neuralnet(output ~ t1_18 + t2_18 + t3_18 + t4_18 + t7_18 + t1_19 + t2_19 +
                       t3_19 + t4_19 + t7_19 + t1_20 + t2_20 + t3_20 + t4_20 + t7_20,
                   data = train_data, hidden = c(5,7), act.fct = "logistic",
                   linear.output = FALSE, stepmax = 1e5)

mlp_5 <- neuralnet(output ~ t1_18 + t1_19 + t1_20,
                   data = train_data, hidden = 3, act.fct = "tanh",
                   linear.output = FALSE, stepmax = 1e5)


selected_mlp = mlp_1 #Select the mlp for the rest of the prediction

# Make predictions on the test data & Rescale the predictions back to the original scale
test_pred <- neuralnet::compute(selected_mlp, test_data[, -ncol(test_data)])

# Rescale the predictions back to the original scale
test_pred_rescaled <- test_pred$net.result * sd(output_data$output) +
mean(output_data$output)

# Rescale the output back to the original scale
test_data_df <- as.data.frame(test_data)
test_output_rescaled <- test_data_df$output * sd(output_data$output) +
mean(output_data$output)

# Create actual and predicted comparison table
comparison_table <- data.frame(Actual = test_output_rescaled, Predicted = test_pred_rescaled)

# Evaluate model performance
mse <- mean((test_pred_rescaled - test_output_rescaled)^2)
rmse <- sqrt(mse)
mae <- mean(abs(test_pred_rescaled - test_output_rescaled))
mape <- mean(abs((test_output_rescaled - test_pred_rescaled)/test_output_rescaled)) * 100
smape <- mean(2 * abs(test_pred_rescaled - test_output_rescaled) / (abs(test_pred_rescaled) +
abs(test_output_rescaled))) * 100


# Print model performance
cat("MSE:", mse, "\n")
cat("RMSE:", rmse, "\n")
cat("MAE:", mae, "\n")
cat("MAPE:", mape, "% \n")
```

```r
cat("sMAPE:", smape, "% \n")

plot(selected_mlp)



# Create a data frame with predicted and observed values
plot_data <- data.frame(Predicted = test_pred_rescaled, Observed = test_output_rescaled)

# Create a scatter plot with a 45-degree line
ggplot(plot_data, aes(x = Observed, y = Predicted)) +
 geom_point() +
 geom_abline(intercept = 0, slope = 1, color = "red") +
 labs(x = "Actual", y = "Predicted", title = "Predicted vs. Actual (NARX)")
```

References

Alboukadel Kassambara. (no date). Determining The Optimal Number Of Clusters: 3 Must Know Methods - Datanovia. *Datanovia*. Available at https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/.

Hyndman, R.J., & Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. Available at OTexts.com/fpp2.

Sagala, N.T.M. and Gunawan, A.A.S. (2022). Discovering the Optimal Number of Crime Cluster Using Elbow, Silhouette, Gap Statistics, and NbClust Methods. *ComTech: Computer, Mathematics and Engineering Applications*, 13 (1), 1–10. Available at https://doi.org/10.21512/comtech.v13i1.7270.

StatQuest: PCA in R. (2017). *YouTube*. Available at https://www.youtube.com/watch?v=0Jp4gsfOLMs.

Zhu, J. et al. (2019). Electric Vehicle Charging Load Forecasting: A Comparative Study of Deep Learning Approaches. *Energies*, 12 (14), 2692. Available at https://doi.org/10.3390/en12142692.