UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

INFORMATICS INSTITUTE OF TECHNOLOGY

IN COLLABORATION WITH

UNIVERSITY OF WESTMINSTER (UOW)

B.Eng. (Hons) Software Engineering

5DATA001C.2 – Machine Learning and Data Mining

Module Leader: Achala Aponso

Coursework 01

2021

UOW ID: w1761265

Student ID: 2019281

Student Full Name: Mohammed Nazhim Kalam

CLUSTERING PART

1. **Discussion of the methodologies used in *reducing the dimensionality*.**

- Since, this dataset "**vehicles.xlsx**" has a large number of features or columns (19 features), this is referred to as **a high dimensional dataset**.
- Having a large number of dimensions will lead to the **curse of dimensionality**.
- **Curse of dimensionality** means that's as the number of dimensions (features/columns) of the dataset increases the points go further apart or data becomes extremely sparse leading the accuracy to decrease.
- **PCA** (Principal Component Analysis) is the most popular technique which can be used for dimensionality reduction, PCA helps to identify all those high correlated variables, which are not related to the target variable at all and drop them out from further analysis.
- Therefore, using PCA we can convert **a high dimension data into low dimension** without losing any of the important features which are correlated to the target class.

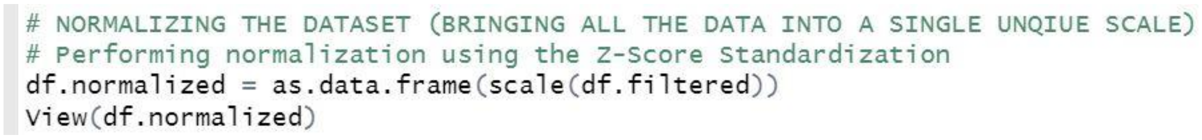
2. **What is PCA?**

- **PCA** or also known as Principal Component Analysis, is an unsupervised learning algorithm that is used specifically for dimensionality reduction in machine learning.
- When the PCA algorithm is applied it produces something called as the **Principal Components**.
- The aim of the PCA algorithm is to **lower-dimensions** from a higher dimension but still retain the quality of the data.

3. **What is Scaling and why do we need them?**

- Scaling or feature scaling is a technique which is used to standardize the data into a fixed range scale. In other words, bring all the data which belonged to different scales into a single common unique scale.
- If the data is not scaled or standardized, then a better result or a higher accuracy cannot be achieved, because all the features contribute in a different proportion not equally.

  E.g.: - Price of the house sale with time, in general price of the house increase with time but time and price of house are of different scale and the difference between the values of the price and time are quite large hence data needs to be scaled down



```
# NORMALIZING THE DATASET (BRINGING ALL THE DATA INTO A SINGLE UNQIUE SCALE)
# Performing normalization using the Z-Score Standardization
df.normalized = as.data.frame(scale(df.filtered))
View(df.normalized)
```

~ This is an image of the normalization step taken in the Kmeans clustering problem statement ~

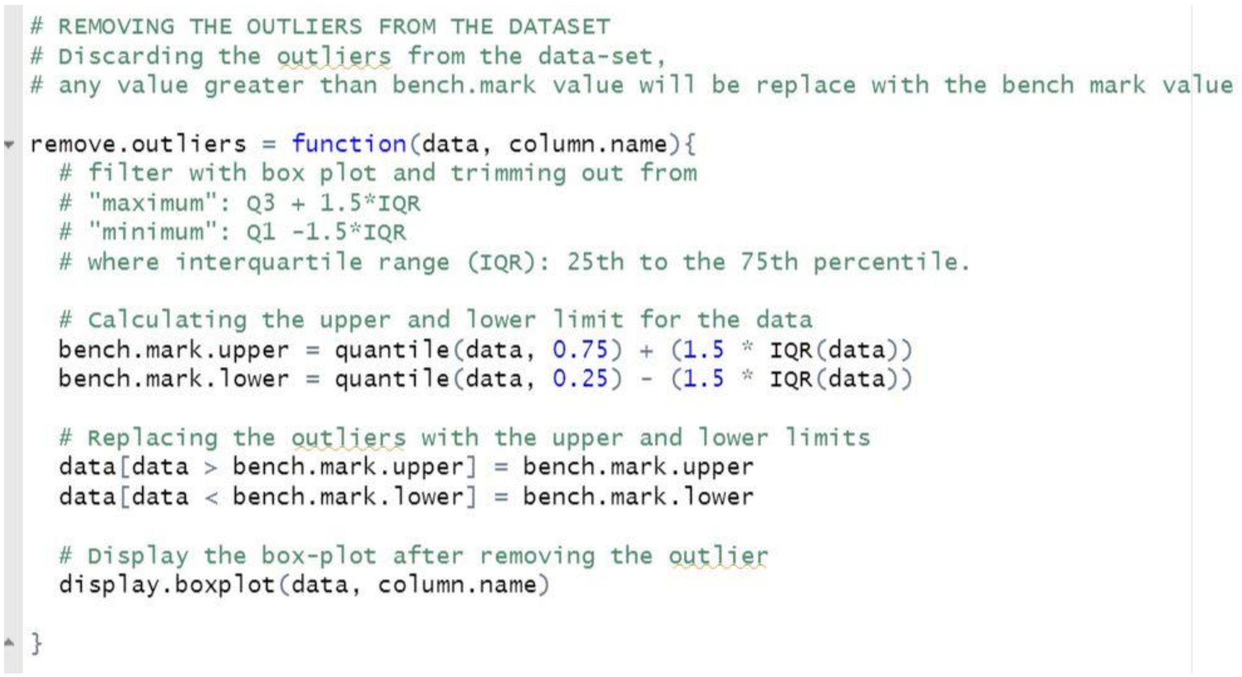
4. **What are outliers?**

- Data points or data values which are completely out of the range from what it's expected to be in are called outliers. In other words, data collected due to some fault or error which makes it to fall way out of the range of data expected.
- Example, consider the height of students and there are 2 outliers present here which is 1ft and 8ft. (In general there are no students with the height of 1ft or 8ft so these are outliers)

Heights: **1ft**, 5ft, 5.1ft, 4.9ft, 5.7ft, 5.9ft, **8ft**

5. **Why do outliers need to be removed from the dataset?**

- The presence of an outlier can affect the accuracy of the model we create. This is because the data is not clean. With clean data only we can create a better accurate model accuracy.
- Hence, outliers need to be removed from the dataset before the training process.



```
# REMOVING THE OUTLIERS FROM THE DATASET
# Discarding the outliers from the data-set,
# any value greater than bench.mark value will be replace with the bench mark value

remove.outliers = function(data, column.name){
  # filter with box plot and trimming out from
  # "maximum": Q3 + 1.5*IQR
  # "minimum": Q1 -1.5*IQR
  # where interquartile range (IQR): 25th to the 75th percentile.

  # Calculating the upper and lower limit for the data
  bench.mark.upper = quantile(data, 0.75) + (1.5 * IQR(data))
  bench.mark.lower = quantile(data, 0.25) - (1.5 * IQR(data))

  # Replacing the outliers with the upper and lower limits
  data[data > bench.mark.upper] = bench.mark.upper
  data[data < bench.mark.lower] = bench.mark.lower

  # Display the box-plot after removing the outlier
  display.boxplot(data, column.name)

}
```

~ This is an image of the outlier removal step taken in Kmeans clustering problem statement ~

6. **How can we find out if there are outliers in our dataset?**

- By drawing up a box plot for each feature of your dataset, you will be able to find data points which go above the maximum range and data points going below the minimum range and these are the outliers and has to be removed. You can remove the outliers by literally removing the data from the dataset or a better approach would be to replace the outlier values with the bench mark values (higher and lower) from the dataset, by doing so we don't loose data from the dataset and also not reduce the accuracy.

NOTE: PCA was not performed because the CW specification method NOT to. However, the code for PCA is commented in the code section for PCA reference

7. **Explain how the Order of scaling and outlier removal is important?**

- Outliers has to be first removed from the dataset before scaling the dataset.
- If the outlier removal is not performed before normalizing the data, then the dataset won't be efficient enough for the modal to predict, this is because the resulting data won't be properly standardized therefore you may end up getting different variables/columns having different standard deviations which is a problem.

8. **Briefly explain the meaning of:**

   1. ***Accuracy***:
      - Accuracy is one an evaluation metric type for classification models.
      - Accuracy is also defined by the following formula:

        Accuracy = Total Number of correct predictions / Total Number of predictions

        Accuracy = (True Positives + True Negatives) / Total Number of predictions

        In other words, accuracy represents how close a measurement comes to its true value.

   2. ***Precision***:
      - Precision is another evaluation metrics which is also referred to as the spread of the measured values.
      - Precision is calculated by the following formula:

        Precision = True Positives / (True Positives + False Positives)

   - The result of this equation lies between 0 to 1, 0.0 indicating that there is no precision and 1.0 for full precision.

   3. ***Recall***:
      - Recall is another evaluation metrics that is used to quantify the number of correct positive predictions made out of all positive predictions that could have been made.
      - Unlike precision, Recall provides an indication of missed positive predictions.
      - Recall is calculated by the following formula:

        Recall = True Positives / (True Positives + False Negatives)

9. **Results and Discussion using the Confusion matrix with respect to the calculation for the accuracy/recall and precision matrices.**

- When automated tools were used inorder to calculate the number of clusters formed these were the following results which were obtained.

Using fviz_nbclust method

Elbow Method (**Automated**) gave **3 clusters** as the BEST.
Silhouette Method (**Automated**) gave **2 clusters** as the BEST.
Gap Statistics Method (**Automated**) gave **3 clusters** as the BEST.

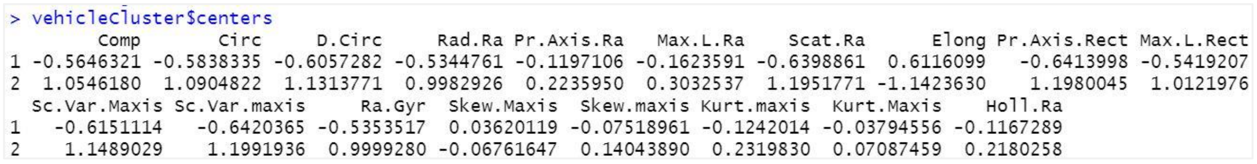
Using NbClust method

By using Euclidean Distance gave **2 clusters** as the BEST.
By using Manhattan Distance gave **2 clusters** as the BEST.

- A **manually** created **elbow method** is used to compute the best number of clusters, by looping through 1 to 10 number of clusters and performing the KMeans clustering and calculating its accuracy using the *confusion matrix*. These are the follow results given for the clusters from 1 to 10 with its accuracy, precision and its recall value.

  - 1 Cluster (Accuracy: 23.5%)
  - 2 Cluster (Accuracy: 36.9%)
  - 3 Cluster (Accuracy: 32.2%)
  - 4 Cluster (Accuracy: 32.7%)
  - 5 Cluster (Accuracy: 13.6%)
  - 6 Cluster (Accuracy: 15.7%)
  - 7 Cluster (Accuracy: 11.7%)
  - 8 Cluster (Accuracy: 14.4%)
  - 9 Cluster (Accuracy: 8.2%)
  - 10 Cluster (Accuracy: 11.2%)

- Since the 2-cluster have the highest accuracy, **2 clusters** are considered and taken as the optimal number of clusters for this problem statement.
- These are the following coordinates of the centers of the clusters.

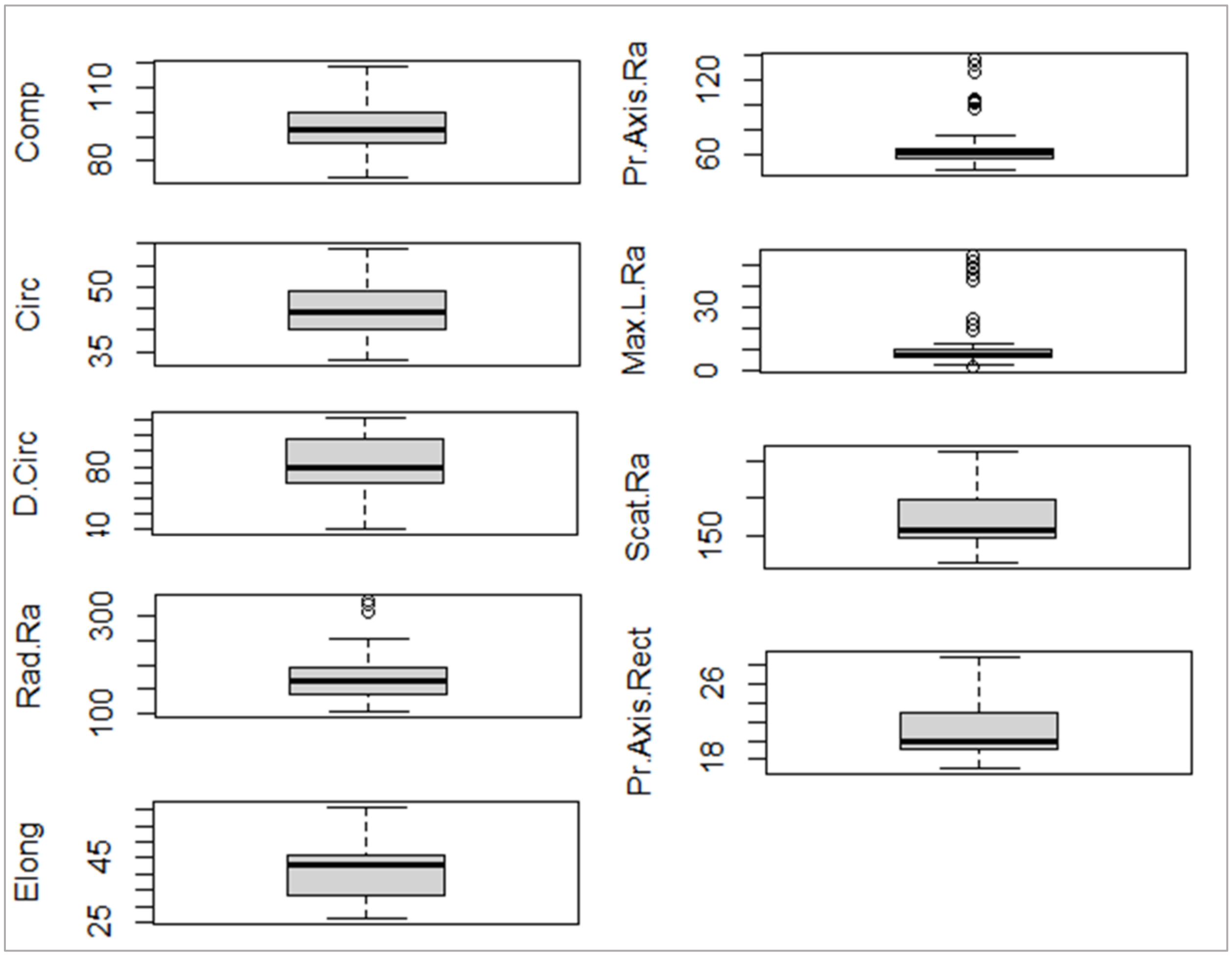


```
> vehicleCluster$centers
        Comp       Circ     D.Circ     Rad.Ra Pr.Axis.Ra   Max.L.Ra    Scat.Ra      Elong Pr.Axis.Rect Max.L.Rect
1 -0.5646321 -0.5838335 -0.6057282 -0.5344761 -0.1197106 -0.1623591 -0.6398861  0.6116099   -0.6413998 -0.5419207
2  1.0546180  1.0904822  1.1313771  0.9982926  0.2235950  0.3032537  1.1951771 -1.1423630    1.1980045  1.0121976
  Sc.Var.Maxis Sc.Var.maxis     Ra.Gyr  Skew.Maxis  Skew.maxis Kurt.maxis  Kurt.Maxis    Holl.Ra
1   -0.6151114   -0.6420365 -0.5353517  0.03620119 -0.07518961 -0.1242014 -0.03794556 -0.1167289
2    1.1489029    1.1991936  0.9999280 -0.06761647  0.14043890  0.2319830  0.07087459  0.2180258
```

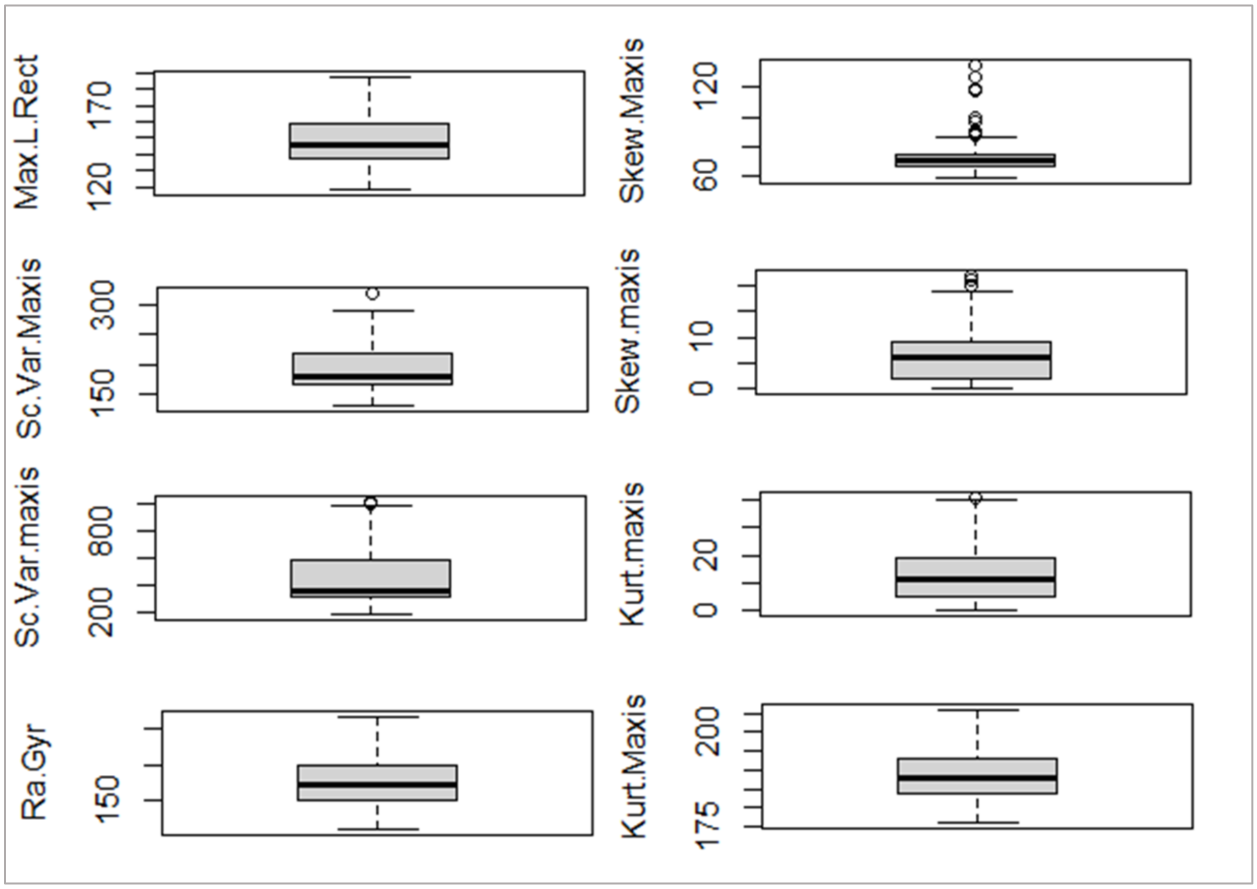
~ Cluster centers coordinates ~

10. **The following are graphs obtained and its respective description about the graph**

- The graphs below represent boxplots for all columns or features of the dataset which consist of outliers

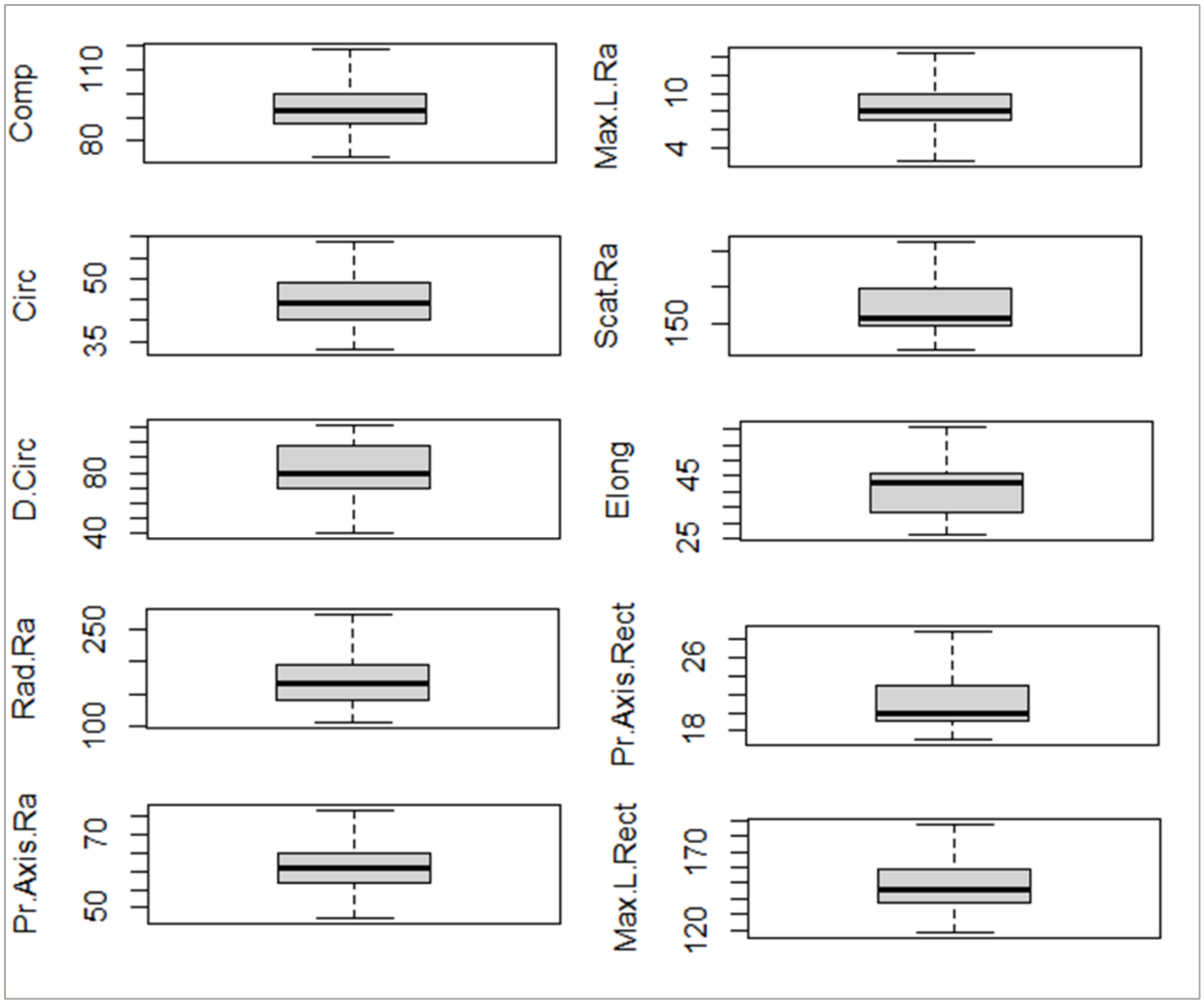


~ Displaying outliers of the features of the dataset ~

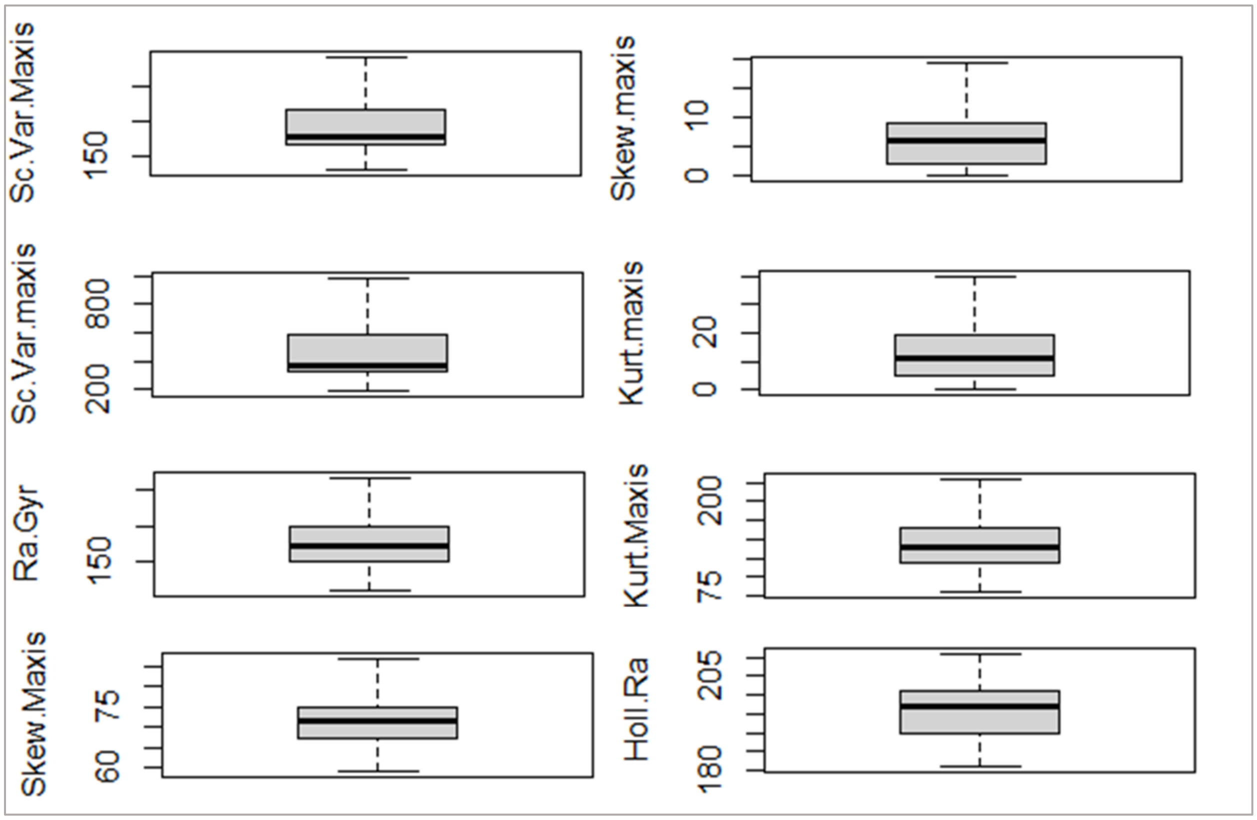


~ Displaying outliers of the features of the dataset ~

- The graphs below represent boxplots for all columns or features of the dataset where the outliers are removed.

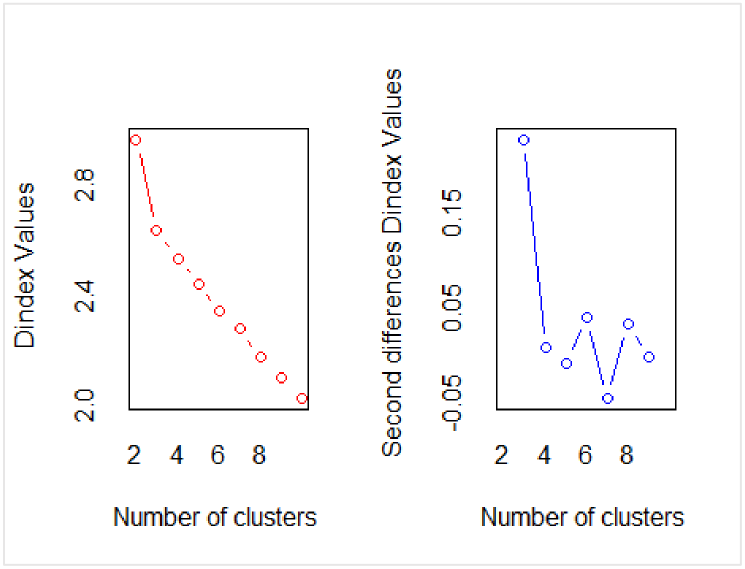


~ Removing outliers of the features of the dataset ~

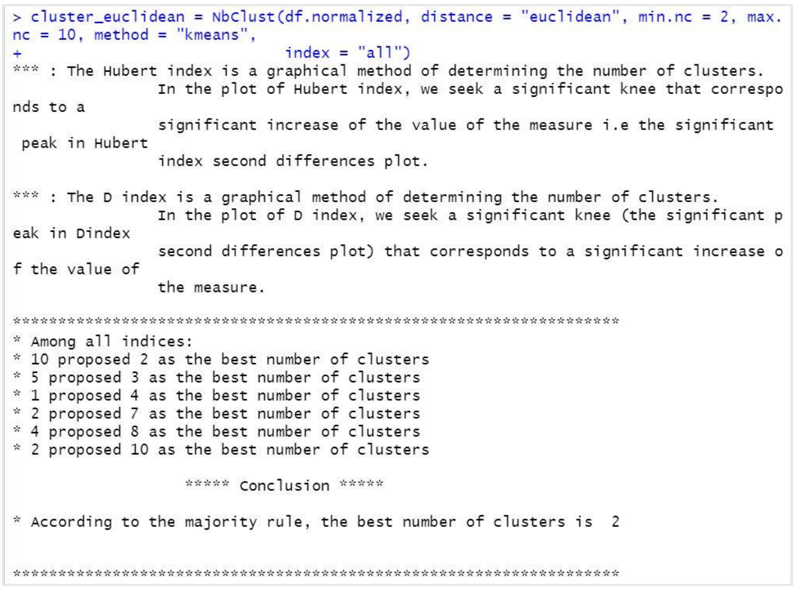


~ Removing outliers of the features of the dataset ~

- Using the **automated tool NbClust** for performing KMeans clustering using Euclidean Distance. (finding the best number of clusters)



~ Related graphs for using Euclidean distance for KMeans clustering ~

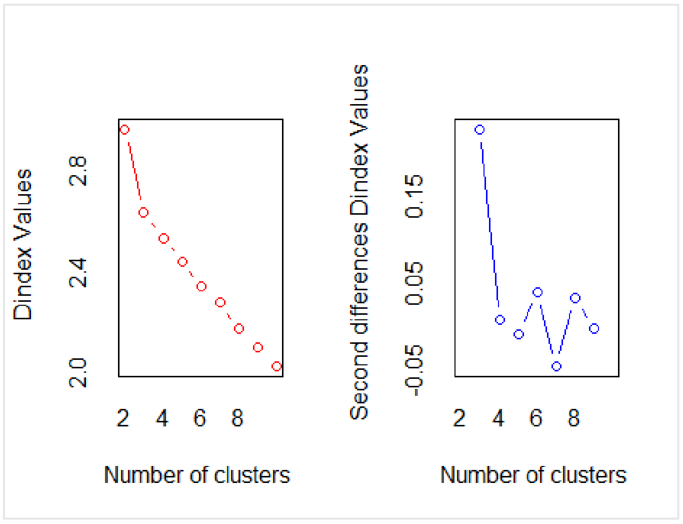


```
> cluster_euclidean = NbClust(df.normalized, distance = "euclidean", min.nc = 2, max.
nc = 10, method = "kmeans",
+                              index = "all")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that correspo
nds 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 p
eak in Dindex
                second differences plot) that corresponds to a significant increase o
f the value of
                the measure.

*******************************************************************
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 5 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 2 proposed 7 as the best number of clusters
* 4 proposed 8 as the best number of clusters
* 2 proposed 10 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2


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

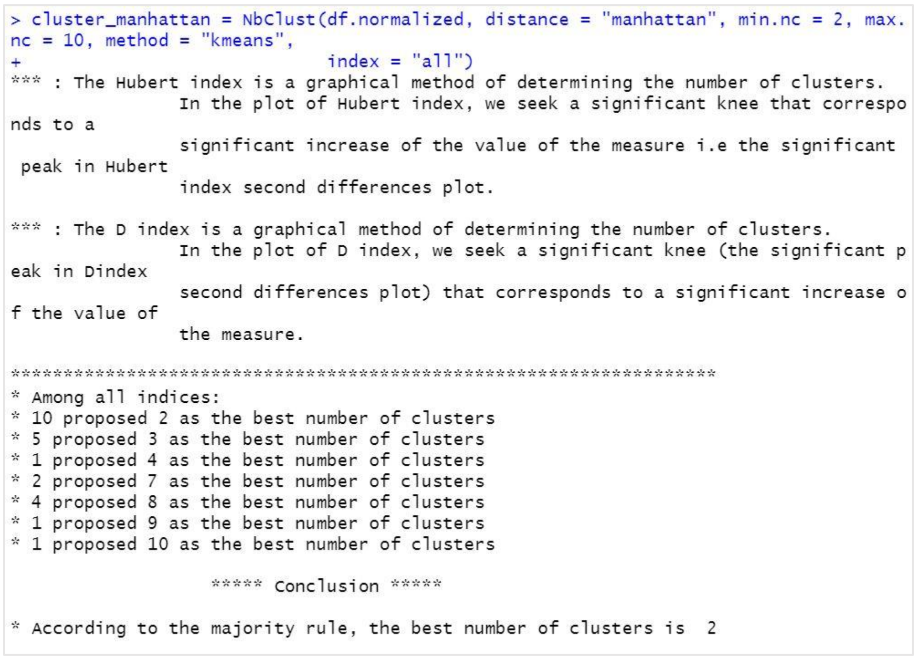
~ Conclusion result for using Euclidean distance for KMeans clustering ~

- Using the **automated tool NbClust** for performing KMeans clustering using Manhattan Distance.

(finding the best number of clusters)



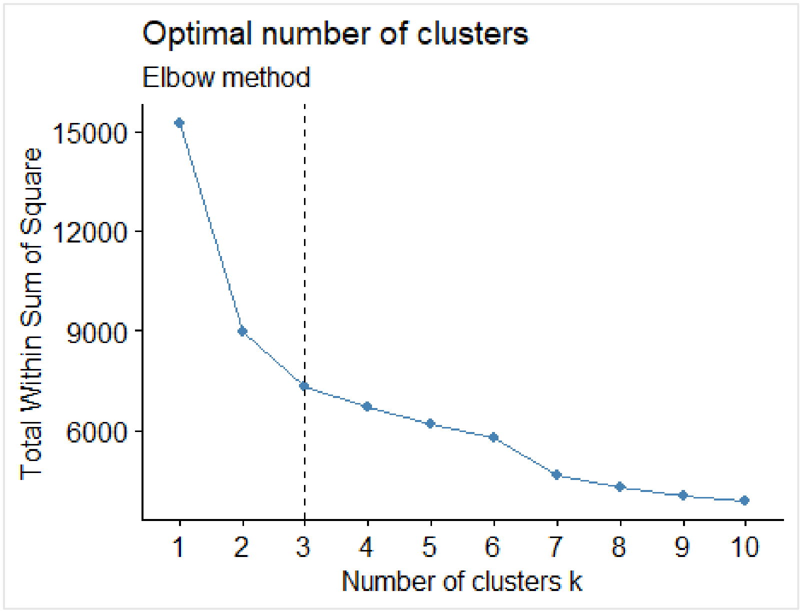
~ Related graphs for using Manhattan distance for KMeans clustering ~



```
> cluster_manhattan = NbClust(df.normalized, distance = "manhattan", min.nc = 2, max.
nc = 10, method = "kmeans",
+                              index = "all")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that correspo
nds 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 p
eak in Dindex
                second differences plot) that corresponds to a significant increase o
f the value of
                the measure.

*******************************************************************
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 5 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 2 proposed 7 as the best number of clusters
* 4 proposed 8 as the best number of clusters
* 1 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  2
```

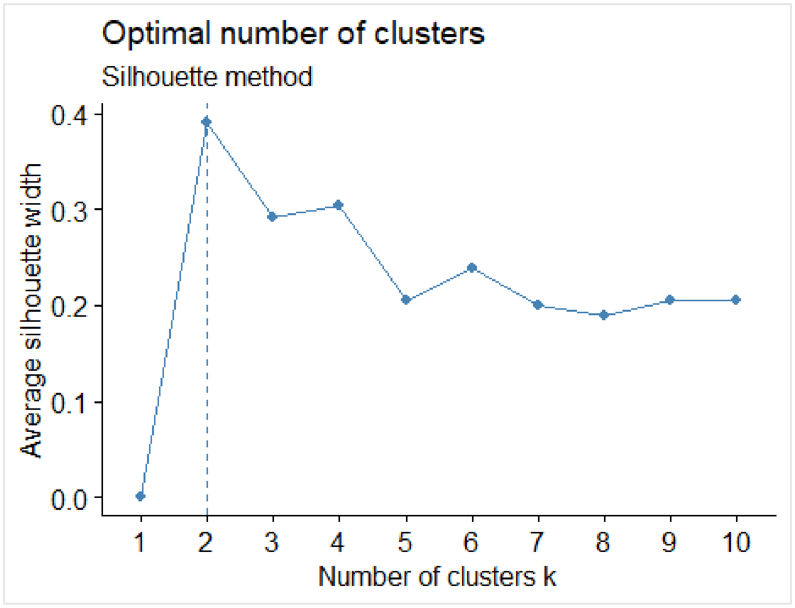
~ Conclusion result for using Manhattan distance for KMeans clustering ~

- Using the **automated tools** to find the best number of centroids (using the **ELBOW** method)



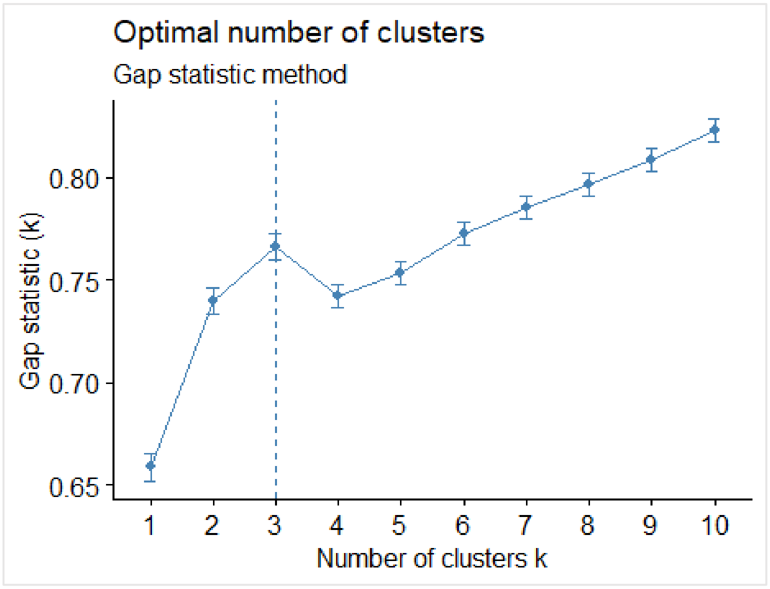
~ Automated elbow method ~

- Using the **automated tools** to find the best **number** of **centroids** (using the **SILHOUETTE** method)



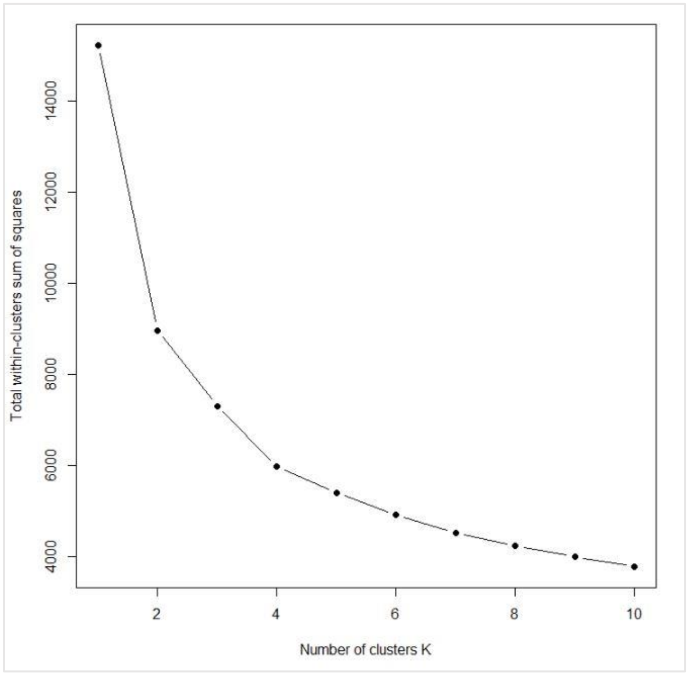
~ Automated Silhouette method ~

- Using the **automated** tools to find the **best** number of **centroids** (using the **GAP STATISTIC** method)



~ Automated Gap Statistic method ~

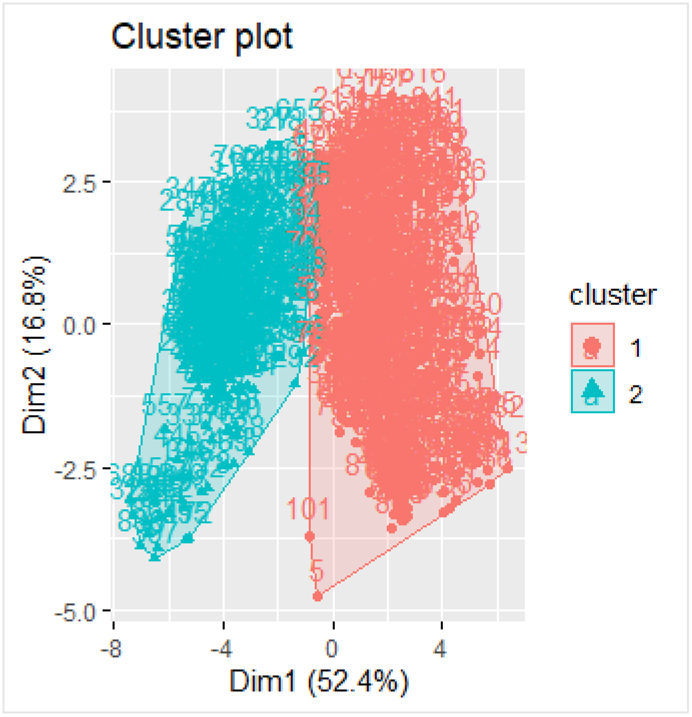
- Using the **manual** method to find the best number of centroids (using the ELBOW method)



~ Manual Elbow method to find optimal clusters~

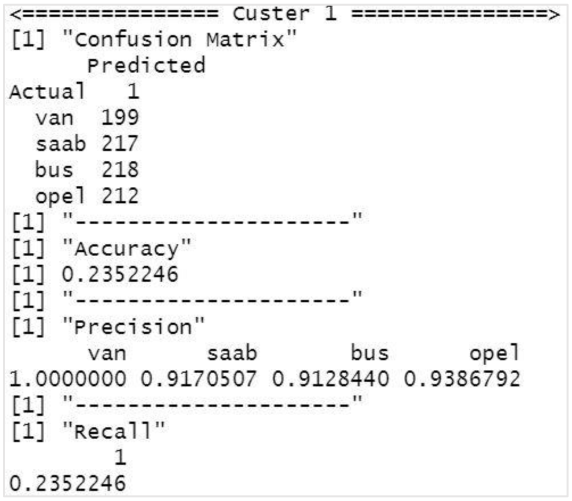
In average by using the automated tools for finding the optimal cluster, out of the 5 methods 3 of the gave **2 clusters** as the optimal *(this is the result from Silhouette Method, Kmeans using Euclidean and Manhattan)* moreover when k means clustering was also performed manually it was found that 2 clusters gave the highest accuracy, hence making 2 clusters the optimal number of clusters for this problem.

A cluster plot related to the clustering of data is given below for visualization (for 2 clusters).



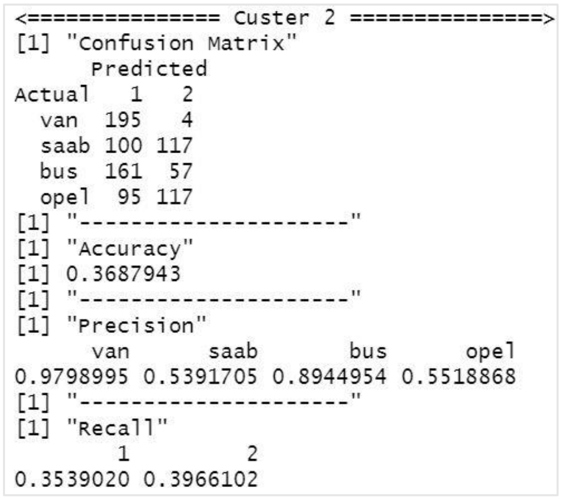
~ Cluster Plot Visualization ~

Given below are the evaluation results obtained for the first 10 clusters inorder to find the optimal cluster centroid number. (using a confusion matrix these evaluations are made)



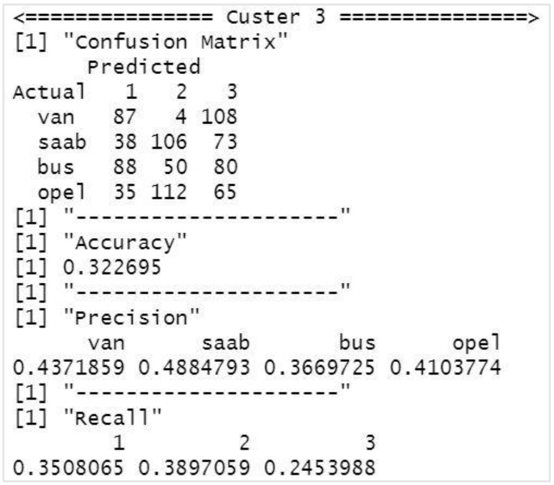
```
<=============== Custer 1 ===============>
[1] "Confusion Matrix"
      Predicted
Actual   1
  van  199
  saab 217
  bus  218
  opel 212
[1] "---------------------"
[1] "Accuracy"
[1] 0.2352246
[1] "---------------------"
[1] "Precision"
      van      saab       bus      opel
1.0000000 0.9170507 0.9128440 0.9386792
[1] "---------------------"
[1] "Recall"
        1
0.2352246
```

~ Cluster 01 evaluation ~



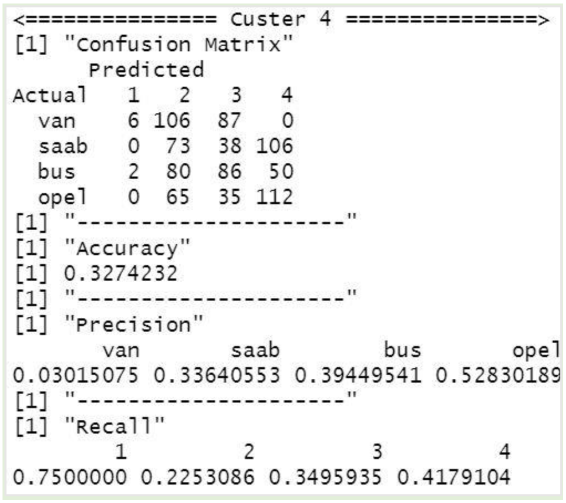
```
<=============== Custer 2 ===============>
[1] "Confusion Matrix"
      Predicted
Actual   1   2
  van  195   4
  saab 100 117
  bus  161  57
  opel  95 117
[1] "---------------------"
[1] "Accuracy"
[1] 0.3687943
[1] "---------------------"
[1] "Precision"
      van      saab       bus      opel
0.9798995 0.5391705 0.8944954 0.5518868
[1] "---------------------"
[1] "Recall"
        1         2
0.3539020 0.3966102
```

~ Cluster 02 evaluation ~



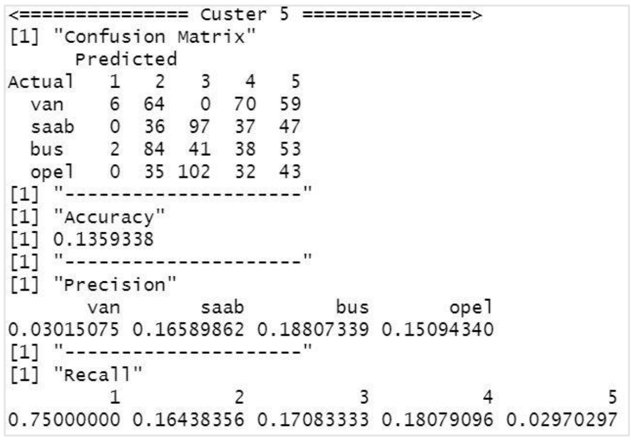
```
<=============== Custer 3 ===============>
[1] "Confusion Matrix"
      Predicted
Actual   1   2   3
  van   87   4 108
  saab  38 106  73
  bus   88  50  80
  opel  35 112  65
[1] "---------------------"
[1] "Accuracy"
[1] 0.322695
[1] "---------------------"
[1] "Precision"
      van      saab       bus      opel
0.4371859 0.4884793 0.3669725 0.4103774
[1] "---------------------"
[1] "Recall"
        1         2         3
0.3508065 0.3897059 0.2453988
```

~ Cluster 03 evaluation ~



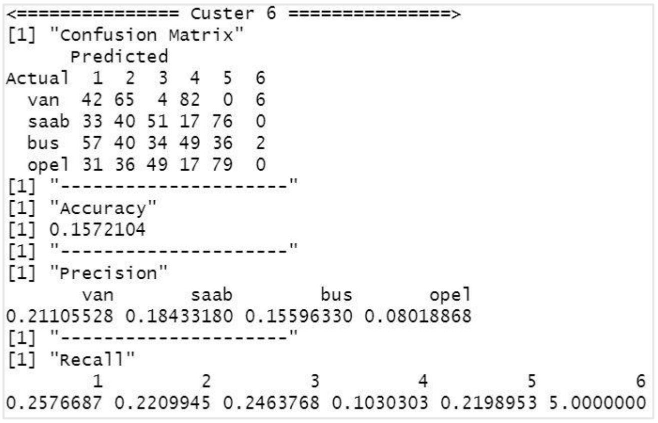
```
<=============== Custer 4 ===============>
[1] "Confusion Matrix"
      Predicted
Actual   1   2   3   4
  van    6 106  87   0
  saab   0  73  38 106
  bus    2  80  86  50
  opel   0  65  35 112
[1] "---------------------"
[1] "Accuracy"
[1] 0.3274232
[1] "---------------------"
[1] "Precision"
       van       saab        bus       opel
0.03015075 0.33640553 0.39449541 0.52830189
[1] "---------------------"
[1] "Recall"
        1         2         3         4
0.7500000 0.2253086 0.3495935 0.4179104
```

~ Cluster 04 evaluation ~



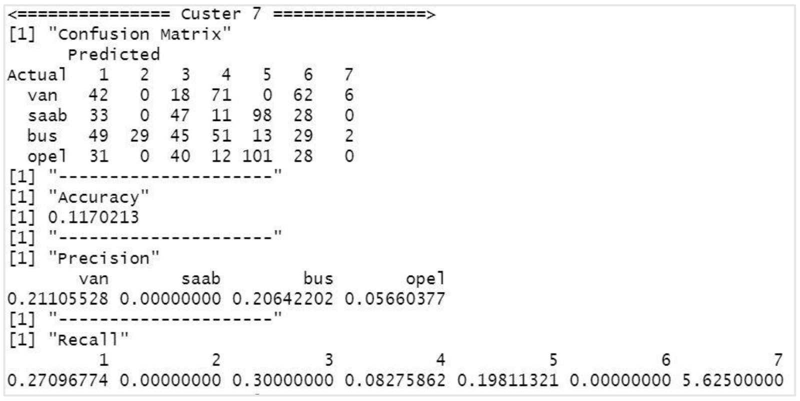
```
<=============== Custer 5 ===============>
[1] "Confusion Matrix"
      Predicted
Actual   1   2   3   4   5
  van    6  64   0  70  59
  saab   0  36  97  37  47
  bus    2  84  41  38  53
  opel   0  35 102  32  43
[1] "---------------------"
[1] "Accuracy"
[1] 0.1359338
[1] "---------------------"
[1] "Precision"
       van       saab        bus       opel
0.03015075 0.16589862 0.18807339 0.15094340
[1] "---------------------"
[1] "Recall"
         1          2          3          4          5
0.75000000 0.16438356 0.17083333 0.18079096 0.02970297
```

~ Cluster 05 evaluation ~



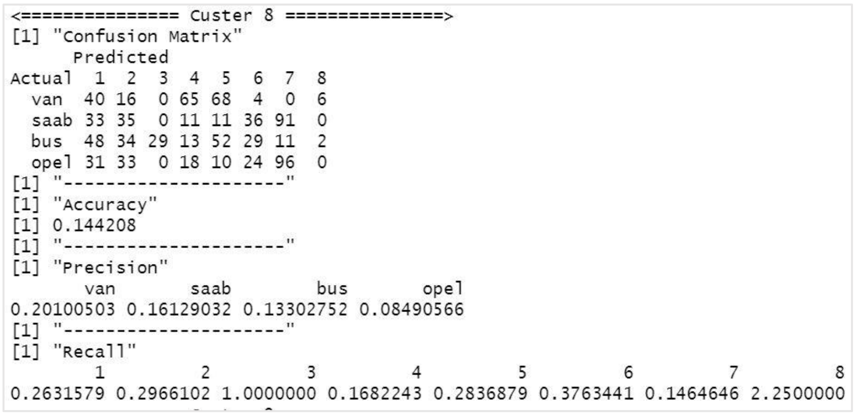
```
<=============== Custer 6 ===============>
[1] "Confusion Matrix"
      Predicted
Actual  1  2  3  4  5  6
  van  42 65  4 82  0  6
  saab 33 40 51 17 76  0
  bus  57 40 34 49 36  2
  opel 31 36 49 17 79  0
[1] "---------------------"
[1] "Accuracy"
[1] 0.1572104
[1] "---------------------"
[1] "Precision"
       van       saab        bus       opel 
0.21105528 0.18433180 0.15596330 0.08018868 
[1] "---------------------"
[1] "Recall"
        1         2         3         4         5         6 
0.2576687 0.2209945 0.2463768 0.1030303 0.2198953 5.0000000 
```

~ Cluster 06 evaluation ~



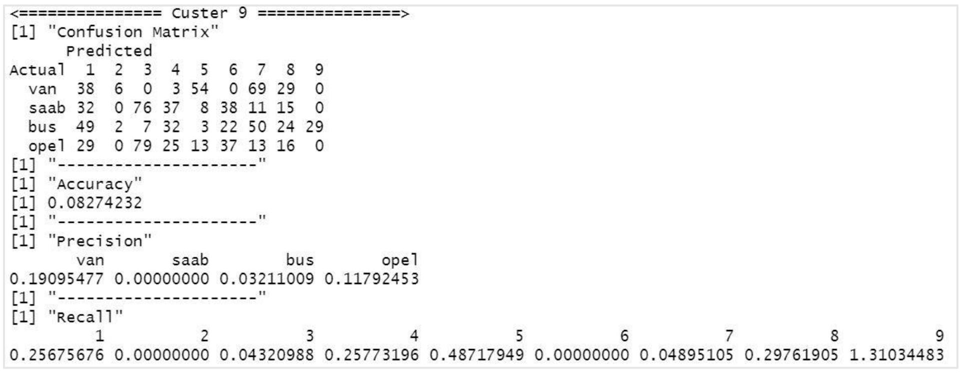
```
<=============== Custer 7 ===============>
[1] "Confusion Matrix"
      Predicted
Actual   1   2   3   4   5   6   7
  van   42   0  18  71   0  62   6
  saab  33   0  47  11  98  28   0
  bus   49  29  45  51  13  29   2
  opel  31   0  40  12 101  28   0
[1] "---------------------"
[1] "Accuracy"
[1] 0.1170213
[1] "---------------------"
[1] "Precision"
       van       saab        bus       opel 
0.21105528 0.00000000 0.20642202 0.05660377 
[1] "---------------------"
[1] "Recall"
         1          2          3          4          5          6          7 
0.27096774 0.00000000 0.30000000 0.08275862 0.19811321 0.00000000 5.62500000 
```

~ Cluster 07 evaluation ~



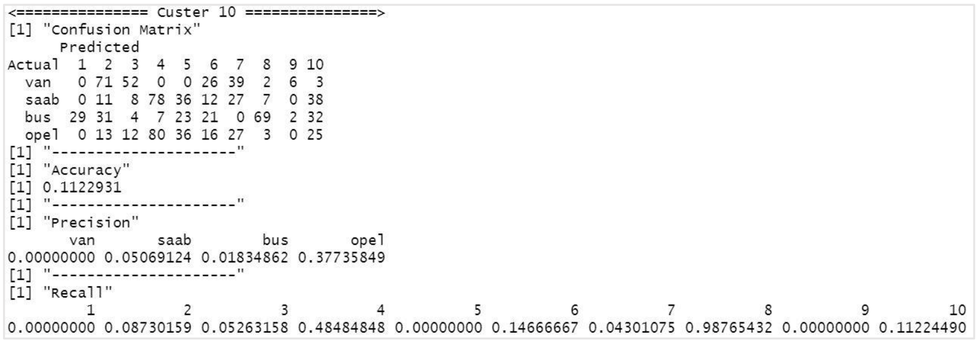
```
<=============== Custer 8 ===============>
[1] "Confusion Matrix"
      Predicted
Actual  1  2  3  4  5  6  7  8
  van  40 16  0 65 68  4  0  6
  saab 33 35  0 11 11 36 91  0
  bus  48 34 29 13 52 29 11  2
  opel 31 33  0 18 10 24 96  0
[1] "---------------------"
[1] "Accuracy"
[1] 0.144208
[1] "---------------------"
[1] "Precision"
       van       saab        bus       opel 
0.20100503 0.16129032 0.13302752 0.08490566 
[1] "---------------------"
[1] "Recall"
        1         2         3         4         5         6         7         8 
0.2631579 0.2966102 1.0000000 0.1682243 0.2836879 0.3763441 0.1464646 2.2500000 
```

~ Cluster 08 evaluation ~



```
<=============== Custer 9 ===============>
[1] "Confusion Matrix"
      Predicted
Actual  1  2  3  4  5  6  7  8  9
  van  38  6  0  3 54  0 69 29  0
  saab 32  0 76 37  8 38 11 15  0
  bus  49  2  7 32  3 22 50 24 29
  opel 29  0 79 25 13 37 13 16  0
[1] "---------------------"
[1] "Accuracy"
[1] 0.08274232
[1] "---------------------"
[1] "Precision"
       van       saab        bus       opel
0.19095477 0.00000000 0.03211009 0.11792453
[1] "---------------------"
[1] "Recall"
         1          2          3          4          5          6          7          8          9
0.25675676 0.00000000 0.04320988 0.25773196 0.48717949 0.00000000 0.04895105 0.29761905 1.31034483
```

~ Cluster 09 evaluation ~



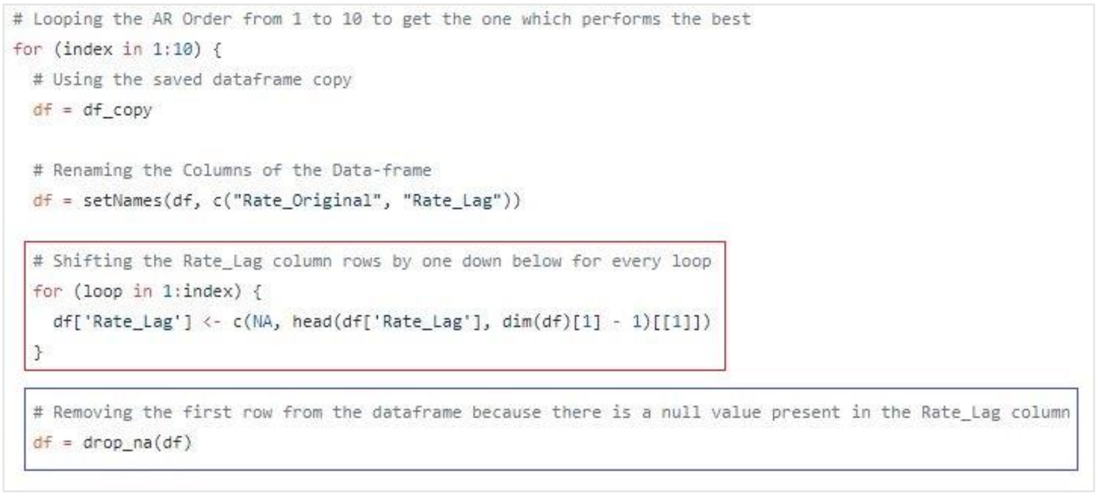
```
<=============== Custer 10 ===============>
[1] "Confusion Matrix"
      Predicted
Actual  1  2  3  4  5  6  7  8  9 10
  van   0 71 52  0  0 26 39  2  6  3
  saab  0 11  8 78 36 12 27  7  0 38
  bus  29 31  4  7 23 21  0 69  2 32
  opel  0 13 12 80 36 16 27  3  0 25
[1] "---------------------"
[1] "Accuracy"
[1] 0.1122931
[1] "---------------------"
[1] "Precision"
       van       saab        bus       opel
0.00000000 0.05069124 0.01834862 0.37735849
[1] "---------------------"
[1] "Recall"
         1          2          3          4          5          6          7          8          9         10
0.00000000 0.08730159 0.05263158 0.48484848 0.00000000 0.14666667 0.04301075 0.98765432 0.00000000 0.11224490
```

~ Cluster 10 evaluation ~

FORECASTING PART

1. **Discussion of the various schemes used to define the input vector and related evidence.**

   Since this problem statements requires the "autoregressive" (AR) approach, the input matrix for the model will change or vary depending on the order of AR considered when training. Suppose if AR 1 approach is considered to be applied for the model, then another duplicated column of the exchange rate column is created but the rows are shifted down by one, this is called the **lag**. Now using the lag column rates as the input for the model and output as the original rate column we perform the training process. This is repeated from AR1 to AR10 in order to find which is the best order of AR. Likewise, the input vector which is the rates lag will change when the order of AR changes (rows will get shifted with respect to the AR order). (NOTE: once rows are shifted the NULL value rows are removed before training).

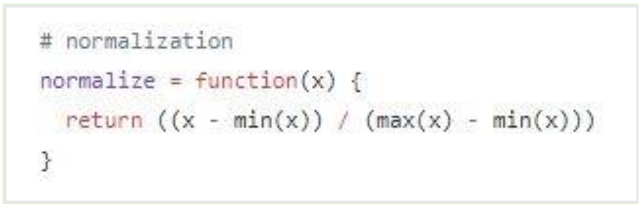


```
# Looping the AR Order from 1 to 10 to get the one which performs the best
for (index in 1:10) {
  # Using the saved dataframe copy
  df = df_copy

  # Renaming the Columns of the Data-frame
  df = setNames(df, c("Rate_Original", "Rate_Lag"))

  # Shifting the Rate_Lag column rows by one down below for every loop
  for (loop in 1:index) {
    df['Rate_Lag'] <- c(NA, head(df['Rate_Lag'], dim(df)[1] - 1)[[1]])
  }

  # Removing the first row from the dataframe because there is a null value present in the Rate_Lag column
  df = drop_na(df)
```

   In the above diagram it clearly visible that there is a looping going on to get the best order of AR from 1 to 10. In the **RED** colored box indicates that the shifting of the columns happens in order to get the lag rates and this changes as the order of AR changes. Moreover, the **BLUE** box indicates that once the shift rows are completed the top rows which contain NULL values after shifting is removed.

2. **Explain why normalization procedure is necessary for this specific type of NN.**

   Since neural networks deals with a large number of layers and a large number of nodes then the TIME TAKEN for the model to train properly will take a lot of time in general, but by normalizing the data input to the neural network it SPEEDS up the learning process to a higher rate leading to faster or quicker convergence or increase the chances to reach the global minima quickly. Moreover, normalization is important give a consistence scale range for all the features there by making it more accurate for training and not making it bias to a specific feature due to its input size.



```
# normalization
normalize = function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
```

~ Normalizing the data using MIN MAX normalization ~

3. Comparision table for the testing performance based on the statistical indices.

Input data/vector used is the AR1 rate lag data

Performance Table Results for 1 hidden layer

PARAMETERS	MAE	RMSE	MAPE
hidden=c(1), act.fct = "logistic", learningrate = 0.1	**2.204 %**	**2.9626 %**	**4.3369 %**
hidden=c(2), act.fct = "logistic", learningrate = 0.1	**2.1872 %**	**2.9313 %**	**4.3196 %**
hidden=c(3), act.fct = "logistic", learningrate = 0.1	**2.202 %**	**2.9201 %**	**4.3281 %**
hidden=c(4), act.fct = "logistic", learningrate = 0.1	**2.2107 %**	**2.9423 %**	**4.3616 %**
hidden=c(5), act.fct = "logistic", learningrate = 0.1	**2.206 %**	**2.9204 %**	**4.3337 %**
hidden=c(6), act.fct = "logistic", learningrate = 0.1	**2.1999 %**	**2.9036 %**	**4.2842 %**
hidden=c(7), act.fct = "logistic", learningrate = 0.1	**2.2146 %**	**2.9217 %**	**4.3451 %**
hidden=c(8), act.fct = "logistic", learningrate = 0.1	**2.2227 %**	**2.925 %**	**4.3579 %**
hidden=c(9),	**2.2332 %**	**2.9339 %**	**4.3808 %**

act.fct = "logistic",
learningrate = 0.1			

hidden=c(10),
act.fct = "logistic",
learningrate = 0.1
	2.2386 %	**2.9238 %**	**4.3693 %**

The optimal number of nodes found for the single hidden layer MLP NN is 6.

Now finding the optimal activation function for this MLP NN structure

hidden=c(6),
act.fct = "tanh",
learningrate = 0.1
	2.1921 %	**2.9261 %**	**4.3228 %**

hidden=c(6),
act.fct = "logistic",
learningrate = 0.1
	2.1999 %	**2.9036 %**	**4.2842 %**

It can be clearly seen that "logistic" performs much better.

Now finding the optimal learning rate for this MLP NN structure

hidden=c(6),
act.fct = "logistic",
learningrate = 0.02
	2.2184 %	**2.9172 %**	**4.3628 %**

hidden=c(6),
act.fct = "logistic",
learningrate = 0.04
	2.2484 %	**2.9625 %**	**4.4272 %**

hidden=c(6),
act.fct = "logistic",
learningrate = 0.06
	2.2147 %	**2.9211 %**	**4.3637 %**

hidden=c(6),
act.fct = "logistic",
learningrate = 0.08
	2.2284 %	**2.9325 %**	**4.3672 %**

hidden=c(6),
act.fct = "logistic",
learningrate = 0.1
	2.1999 %	**2.9036 %**	**4.2842 %**

From the above results learning rate of 0.1 gave the best result, hence the best of set of parameters for 1 hidden layer MLP NN is as follows

HIDDEN_LAYERS = c(6)

ACTIVATION_FUNCTION = "logistic"

LEARNING_RATE = 0.1

Performance Table Results for AR1 with 2 hidden layer, for the first layer 6 nodes will be used since that was the best when calculated for 1 hidden layer MLP NN, the nodes for the 2nd hidden layer will be calculated along with the learning rate, and activation function

PARAMETERS	MAE	RMSE	MAPE
hidden=c(6,1), act.fct = "logistic", learningrate = 0.1	**2.420 %**	**2.956 %**	**4.3451 %**
hidden=c(6,2), act.fct = "logistic", learningrate = 0.1	**2.1845 %**	**2.9546 %**	**4.3125 %**
hidden=c(6,3), act.fct = "logistic", learningrate = 0.1	**2.236 %**	**2.9236 %**	**4.3459 %**
hidden=c(6,4), act.fct = "logistic", learningrate = 0.1	**2.2512 %**	**2.9633 %**	**4.3617 %**
hidden=c(6,5), act.fct = "logistic", learningrate = 0.1	**2.226 %**	**2.9234 %**	**4.3457 %**
hidden=c(6,6), act.fct = "logistic", learningrate = 0.1	**2.1246 %**	**2.7469 %**	**4.2123 %**
hidden=c(6,7), act.fct = "logistic", learningrate = 0.1	**2.2246 %**	**2.9257 %**	**4.3421 %**
hidden=c(6,8), act.fct = "logistic", learningrate = 0.1	**2.2327 %**	**2.965 %**	**4.3539 %**
hidden=c(6,9), act.fct = "logistic", learningrate = 0.1	**2.2532 %**	**2.9379 %**	**4.3848 %**
hidden=c(6,10),	**2.2686 %**	**2.9258 %**	**4.3653 %**

act.fct = "logistic", learningrate = 0.1			

The optimal number of nodes found for the 2nd hidden layer is 6.

Now finding the optimal activation function for this MLP NN structure

hidden=c(6,6), act.fct = "tanh", learningrate = 0.1	**2.2547 %**	**2.9311 %**	**4.3437 %**
hidden=c(6,6), act.fct = "logistic", learningrate = 0.1	**2.2184 %**	**2.9172 %**	**4.3328 %**

It can be clearly seen that "logistic" performs much better.

Now finding the optimal learning rate for this MLP NN structure

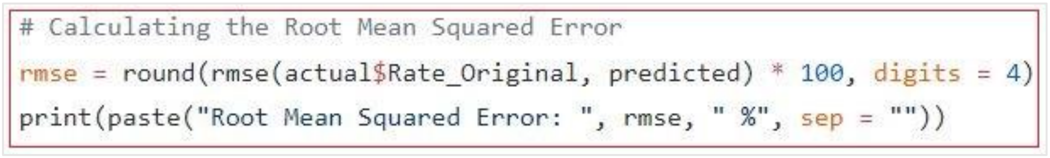
hidden=c(6,6), act.fct = "logistic", learningrate = 0.02	**2.2184 %**	**2.9172 %**	**4.3428 %**
hidden=c(6,6), act.fct = "logistic", learningrate = 0.04	**2.2484 %**	**2.9625 %**	**4.4272 %**
hidden=c(6,6), act.fct = "logistic", learningrate = 0.06	**2.2147 %**	**2.9311 %**	**4.3437 %**
hidden=c(6,6), act.fct = "logistic", learningrate = 0.08	**2.182 %**	**2.9139 %**	**4.2749 %**
hidden=c(6,6), act.fct = "logistic", learningrate = 0.1	**2.2484 %**	**2.9211 %**	**4.333 %**

From the above results learning rate of 0.08 gave the best result, hence the best of set of parameters for 2 hidden layer MLP NN is as follows

```
HIDDEN_LAYERS = c(6,6)

ACTIVATION_FUNCTION = "logistic"

LEARNING_RATE = 0.08
```

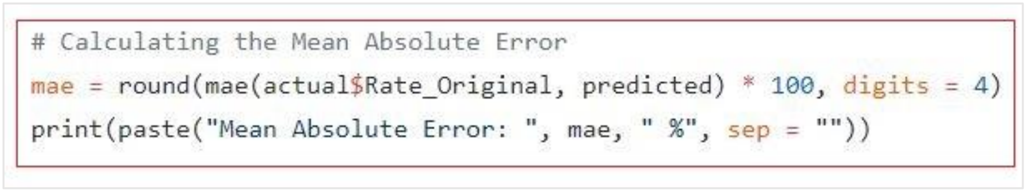
4. **Brief explanation on the statistical indices (RMSE, MAE and MAPE)**

- **RMSE (Root Mean Square Error):** This is a statistical measure of the error of the model in predicting the respective data. Lower the RMSE value better the model is, better the prediction output is as well.



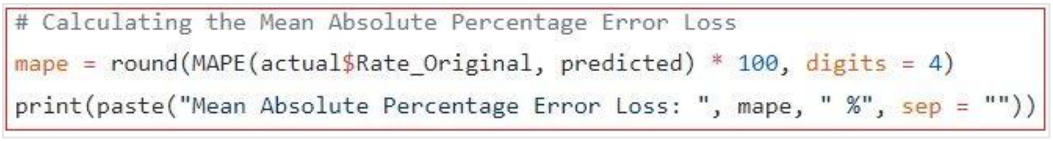
```
# Calculating the Root Mean Squared Error
rmse = round(rmse(actual$Rate_Original, predicted) * 100, digits = 4)
print(paste("Root Mean Squared Error: ", rmse, " %", sep = ""))
```

- **MAE (Mean Absolute Error):** This is basically the sum or total of average of the absolute difference between the predicted and actual values. So, what basically this means is that using the MAE we can get a clear picture as to how wrong the predictions are. Lower the MAE better the model



```
# Calculating the Mean Absolute Error
mae = round(mae(actual$Rate_Original, predicted) * 100, digits = 4)
print(paste("Mean Absolute Error: ", mae, " %", sep = ""))
```

- **MAPE (Mean Absolute Percentage Error):** This is a statistical measure to define the accuracy of a model on a particular dataset or in other terms it is the percentage of average of absolute difference between predicted values and true values, divided by the true values. Lower the MAPE better the model again.

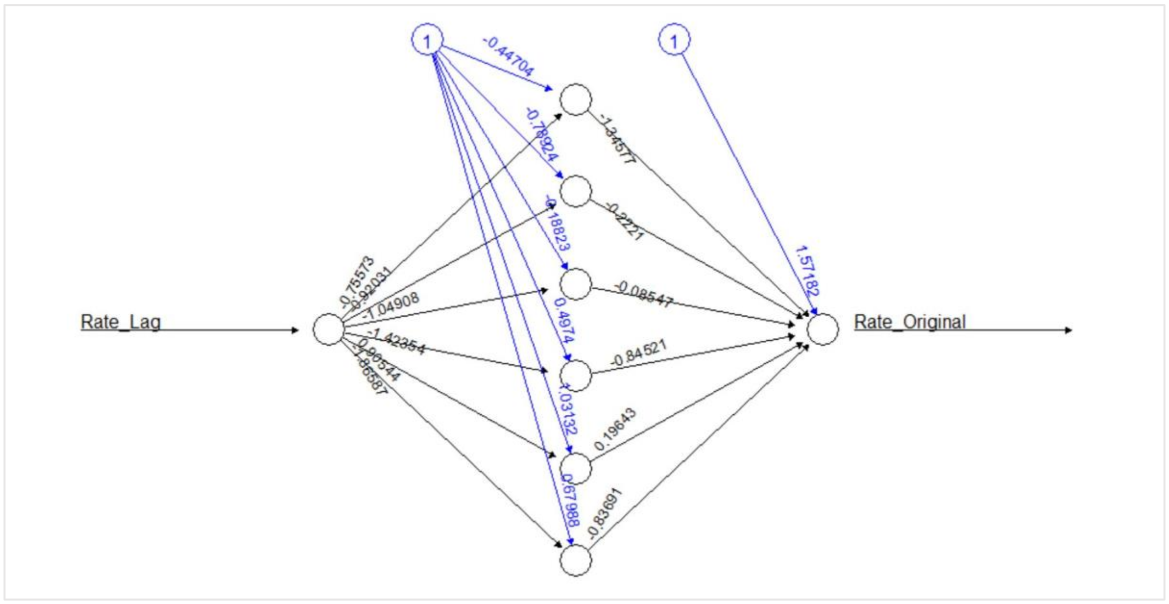


```
# Calculating the Mean Absolute Percentage Error Loss
mape = round(MAPE(actual$Rate_Original, predicted) * 100, digits = 4)
print(paste("Mean Absolute Percentage Error Loss: ", mape, " %", sep = ""))
```

5. **Efficiency result from the best one hidden layer and two hidden layer network with their respective parameters.**

The BEST one hidden layer network

```
HIDDEN_LAYERS = c(6)
ACTIVATION_FUNCTION = "logistic"
LEARNING_RATE = 0.1
```



Number of weights between input and hidden layer:

Number of inputs x Number of Neurons in the hidden layer = 1 x 6 = 6

Number of weights between hidden layer and output layer:

Number of Neurons in the hidden layer x Number of Neurons in output layer = 6 x 1 = 6

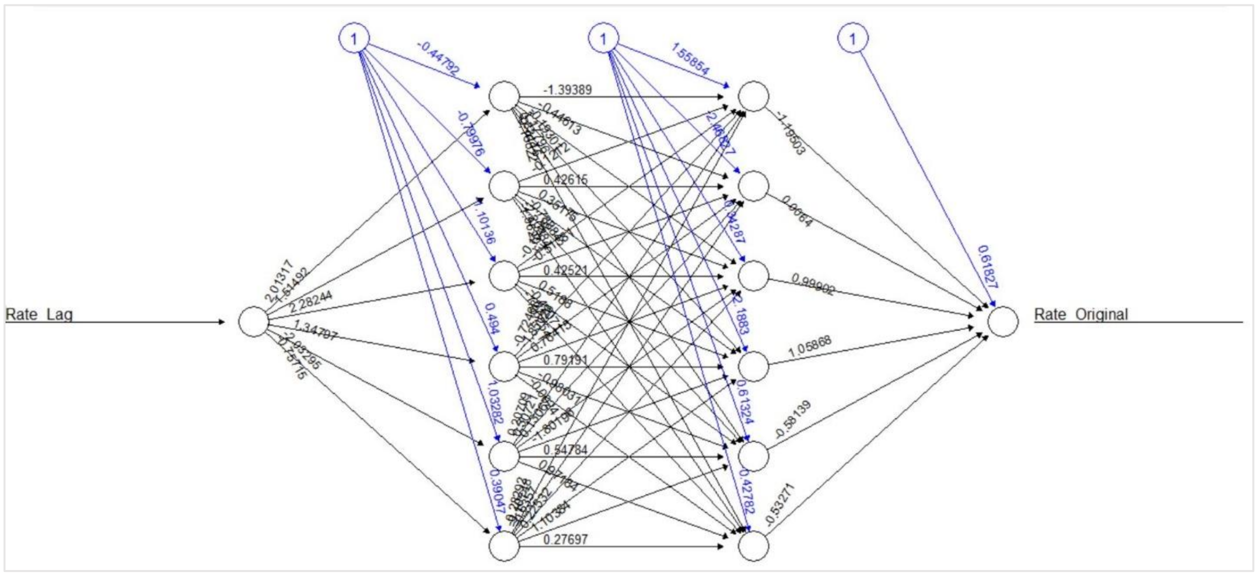
Total Number of weight parameters = 6 + 6 = 12

Statistics Result (Based on AR1 input vector)

MAE	RMSE	MAPE
2.1999 %	**2.9036 %**	**4.2842 %**

The BEST two hidden layer network

```
HIDDEN_LAYERS = c(6,6)

ACTIVATION_FUNCTION = "logistic"

LEARNING_RATE = 0.08
```



Number of weights between input and first hidden layer:

Number of inputs x Number of Neurons in the hidden layer = 1 x 6 = 6

Number of weights between the first hidden layer and the second hidden layer:

Number of neurons in the first hidden layer x Number of neurons in the second hidden layer

= 6 x 6 = 36

Number of weights between second hidden layer and output layer:

Number of Neurons in the hidden layer x Number of Neurons in output layer = 6 x 1 = 6

Total Number of weight parameters = 6 + 36 + 6 = 48

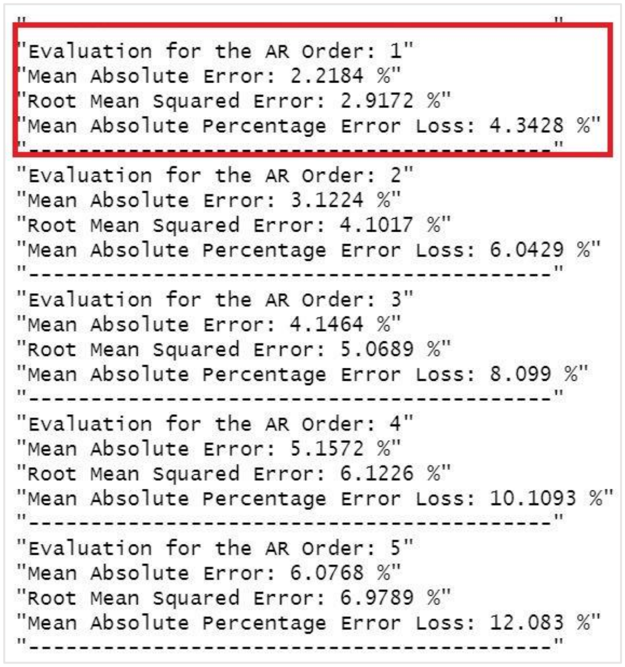
Since there is a big difference in the weight parameters between 1 hidden layer and 2 hidden layer by checking the stats of each of this model a better conclusion can be made.

Statistics Result (Based on AR1 input vector)

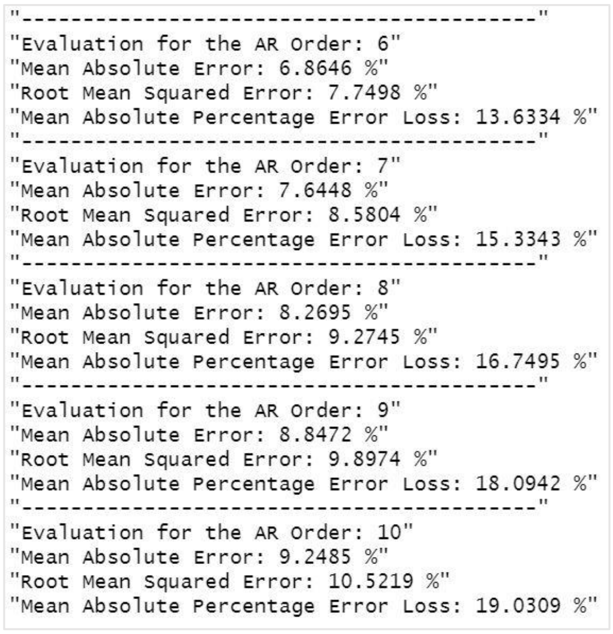
MAE	RMSE	MAPE
2.182 %	**2.9139 %**	**4.2749 %**

When compared both of the Statistics Result table there is only roughly 0.1 difference between the MAE, RMSE and MAPE of the single hidden layer MLP NN and 2 hidden layered MLP NN. Even though the 2 hidden layer MLP NN has a larger number of weight parameters compared to the single hidden layer, it only contributed a very small percentage to the result statistics making the 1 hidden layer MLP NN the efficient Neural Network.

6. **Discussion about the best final** MLP NN **Network by checking with the evaluation stats and with predicted VS actual result graph**



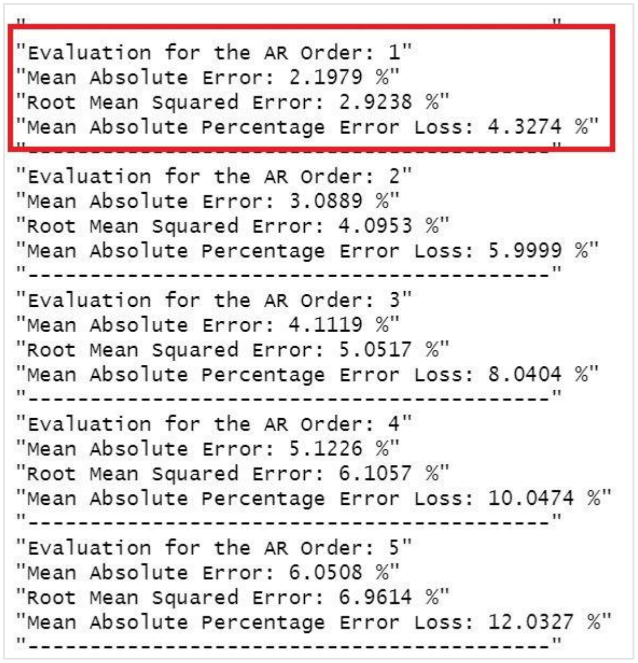
```
"                                           "
"Evaluation for the AR Order: 1"
"Mean Absolute Error: 2.2184 %"
"Root Mean Squared Error: 2.9172 %"
"Mean Absolute Percentage Error Loss: 4.3428 %"
"-------------------------------------------"
"Evaluation for the AR Order: 2"
"Mean Absolute Error: 3.1224 %"
"Root Mean Squared Error: 4.1017 %"
"Mean Absolute Percentage Error Loss: 6.0429 %"
"-------------------------------------------"
"Evaluation for the AR Order: 3"
"Mean Absolute Error: 4.1464 %"
"Root Mean Squared Error: 5.0689 %"
"Mean Absolute Percentage Error Loss: 8.099 %"
"-------------------------------------------"
"Evaluation for the AR Order: 4"
"Mean Absolute Error: 5.1572 %"
"Root Mean Squared Error: 6.1226 %"
"Mean Absolute Percentage Error Loss: 10.1093 %"
"-------------------------------------------"
"Evaluation for the AR Order: 5"
"Mean Absolute Error: 6.0768 %"
"Root Mean Squared Error: 6.9789 %"
"Mean Absolute Percentage Error Loss: 12.083 %"
"-------------------------------------------"
```



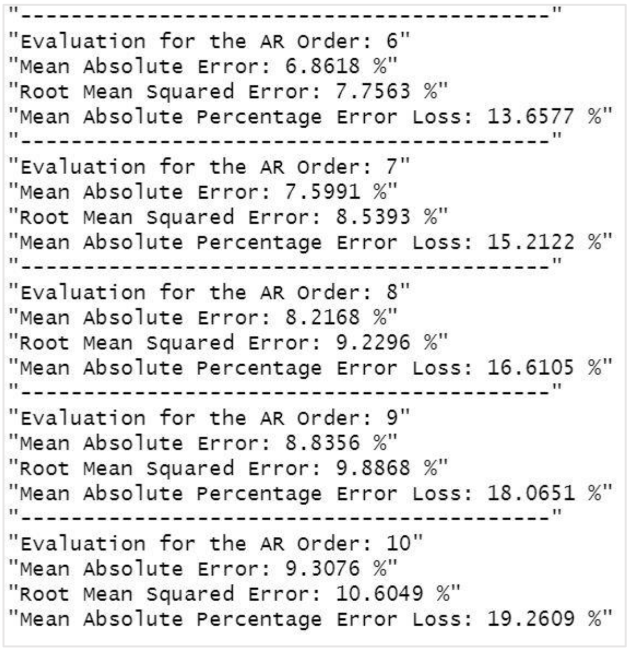
```
"-------------------------------------------"
"Evaluation for the AR Order: 6"
"Mean Absolute Error: 6.8646 %"
"Root Mean Squared Error: 7.7498 %"
"Mean Absolute Percentage Error Loss: 13.6334 %"
"-------------------------------------------"
"Evaluation for the AR Order: 7"
"Mean Absolute Error: 7.6448 %"
"Root Mean Squared Error: 8.5804 %"
"Mean Absolute Percentage Error Loss: 15.3343 %"
"-------------------------------------------"
"Evaluation for the AR Order: 8"
"Mean Absolute Error: 8.2695 %"
"Root Mean Squared Error: 9.2745 %"
"Mean Absolute Percentage Error Loss: 16.7495 %"
"-------------------------------------------"
"Evaluation for the AR Order: 9"
"Mean Absolute Error: 8.8472 %"
"Root Mean Squared Error: 9.8974 %"
"Mean Absolute Percentage Error Loss: 18.0942 %"
"-------------------------------------------"
"Evaluation for the AR Order: 10"
"Mean Absolute Error: 9.2485 %"
"Root Mean Squared Error: 10.5219 %"
"Mean Absolute Percentage Error Loss: 19.0309 %"
```

~ AR order accuracy using the best 1 hidden layer MLP NN **~**

Using the BEST set of parameters for the 1 hidden layer MLP NN on input vectors varying from AR1 to AR10, we are able to see that AR1 or the 1st order of AR gives the best accuracy.



```
"                                          "
"Evaluation for the AR Order: 1"
"Mean Absolute Error: 2.1979 %"
"Root Mean Squared Error: 2.9238 %"
"Mean Absolute Percentage Error Loss: 4.3274 %"
"------------------------------------------"
"Evaluation for the AR Order: 2"
"Mean Absolute Error: 3.0889 %"
"Root Mean Squared Error: 4.0953 %"
"Mean Absolute Percentage Error Loss: 5.9999 %"
"------------------------------------------"
"Evaluation for the AR Order: 3"
"Mean Absolute Error: 4.1119 %"
"Root Mean Squared Error: 5.0517 %"
"Mean Absolute Percentage Error Loss: 8.0404 %"
"------------------------------------------"
"Evaluation for the AR Order: 4"
"Mean Absolute Error: 5.1226 %"
"Root Mean Squared Error: 6.1057 %"
"Mean Absolute Percentage Error Loss: 10.0474 %"
"------------------------------------------"
"Evaluation for the AR Order: 5"
"Mean Absolute Error: 6.0508 %"
"Root Mean Squared Error: 6.9614 %"
"Mean Absolute Percentage Error Loss: 12.0327 %"
"------------------------------------------"
```

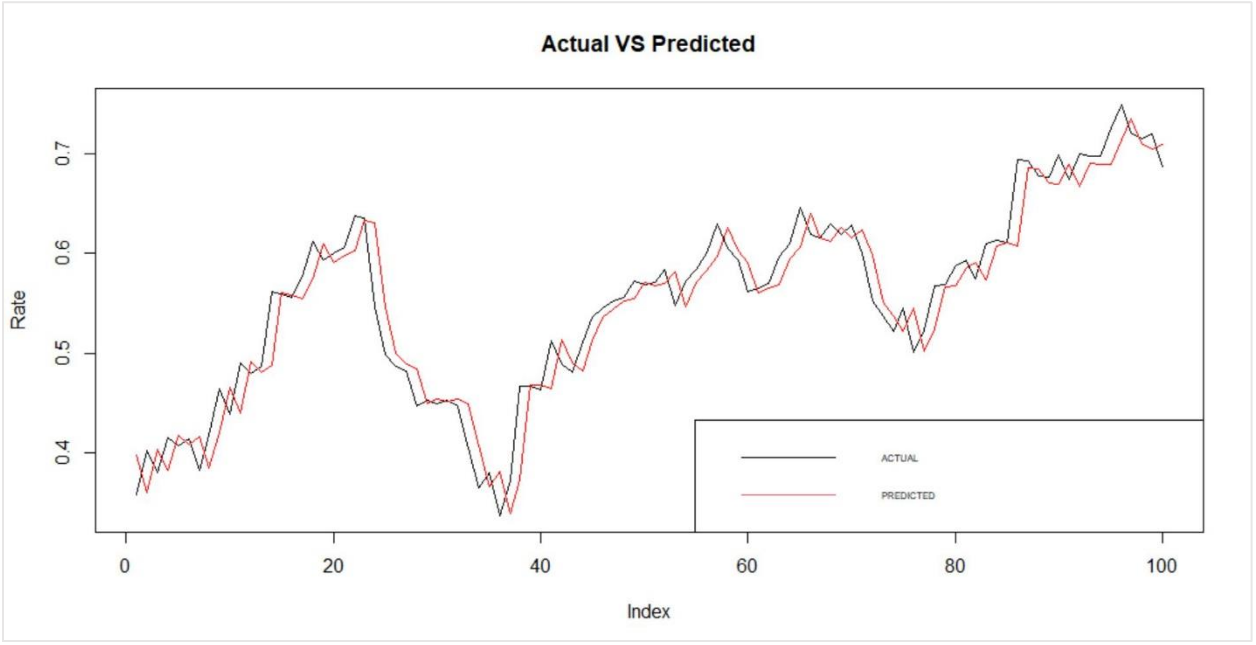


```
"------------------------------------------"
"Evaluation for the AR Order: 6"
"Mean Absolute Error: 6.8618 %"
"Root Mean Squared Error: 7.7563 %"
"Mean Absolute Percentage Error Loss: 13.6577 %"
"------------------------------------------"
"Evaluation for the AR Order: 7"
"Mean Absolute Error: 7.5991 %"
"Root Mean Squared Error: 8.5393 %"
"Mean Absolute Percentage Error Loss: 15.2122 %"
"------------------------------------------"
"Evaluation for the AR Order: 8"
"Mean Absolute Error: 8.2168 %"
"Root Mean Squared Error: 9.2296 %"
"Mean Absolute Percentage Error Loss: 16.6105 %"
"------------------------------------------"
"Evaluation for the AR Order: 9"
"Mean Absolute Error: 8.8356 %"
"Root Mean Squared Error: 9.8868 %"
"Mean Absolute Percentage Error Loss: 18.0651 %"
"------------------------------------------"
"Evaluation for the AR Order: 10"
"Mean Absolute Error: 9.3076 %"
"Root Mean Squared Error: 10.6049 %"
"Mean Absolute Percentage Error Loss: 19.2609 %"
```

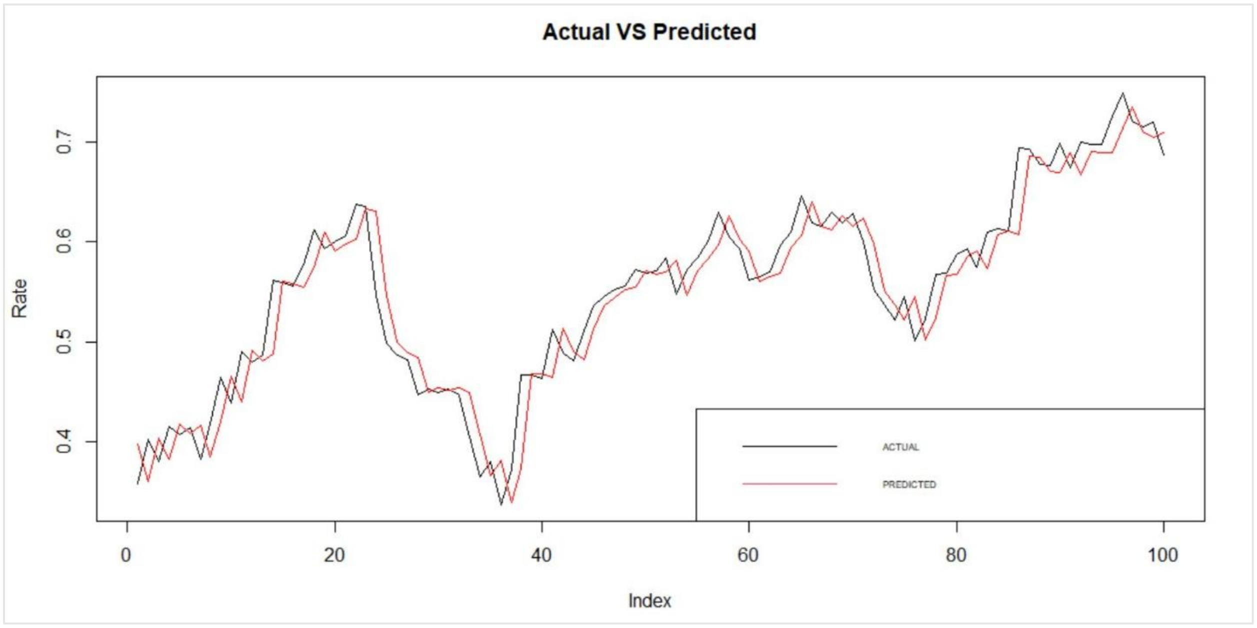
~ AR order accuracy using the best 2 hidden layer MLP NN **~**

Using the BEST set of parameters for the 2 hidden layer MLP NN on input vectors varying from AR1 to AR10, we are able to see that AR1 or the 1st order of AR gives the best accuracy.

Now we can come to a conclusion that AR1 is the best order, now which one out of the 2 models (neural networks) is the best to be considered is the question. Now the statistics and the graphs has to be considered in detail.



~ *Prediction Output VS Desired Output for 1 hidden layer* MLP NN ~



~ Prediction Output VS Desired Output for 2 hidden layer MLP NN *~*

RED LINE = PREDICTED OUTPUT

BLACK LINE = DESIRED OUTPUT

Using the below performance stats we can come to a conclusion which MLP NN **is the best.**

MAE	RMSE	MAPE
2.1999 %	**2.9036 %**	**4.2842 %**

~ Performance statistics for 1 hidden layer MLP NN~

MAE	RMSE	MAPE
2.182 %	**2.9139 %**	**4.2749 %**

~ Performance statistics for 2 hidden layer MLP NN~

Its very clear that there is only a very small difference in the error but however the 2 hidden layer MLP NN has a lower error rate overall when MAE and MAPE of 2 hidden layer MLP NN compared to 1 hidden layer MLP NN.

CONCLUSION

2 HIDDEN LAYER MLP NN GAVE THE BEST ACCURACY (LOWEST ERROR) FOR THIS PROBLEM

APPENDIX

CODE FOR PART 01 (CLUSTERING QUESTION)

```
# --------------------------------------------------
# Name: Mohammed Nazhim Kalam
# Student ID: 2019281
# UoW ID: W1761265
# --------------------------------------------------

# CLUSTERING PART

# Installing package to read Excel Data-set
# install.packages("readxl")     # used to read excel data files
# install.packages("factoextra") # used to determine the optimal number clusters
# install.packages("NbClust")    # used to compute about multiple methods at once,
# # in order to find the optimal number of clusters.
# install.packages("factoextra") # used to plot the clusters out

# Loading the package
library(readxl)
library(knitr)
library(tidymodels)
library(janitor)
library(flexclust)
library(dplyr)
library(factoextra)
library(NbClust)
library(haven)
library(factoextra)

# Reading the data-set "vehicles.xlsx"
```

```
df = read_excel("./vehicles.xlsx")
View(df)

# converting the class column into factors because we are not able to get its count
df = mutate(df, Class = as_factor(df$Class))

# Displaying the types of unique classes present in the data-set
summary(df)

# Removing the Sample index column and the class column from the data-set
df.filtered = subset(df, select = -c(Samples, Class))

# Viewing the filtered data-set
View(df.filtered)

# [PRE-PROCESSING DATA] PERFORMING SCALING AND OUTLIERS REMOVAL

# Checking for any null values present in the data-set (returned 0 so no null values present)
print(sum(is.na(df.filtered)))

# Checking the Summary of the data-set (We can see the stats of the data columns eg: mean, median
etc.)
df.summary = summary(df.filtered)

# Plotting a box plot graph (Box plots are useful to detect potential outliers from the data-set)
display.boxplot = function(data, column.name){
  boxplot(data, ylab = column.name)
}

# calling the display.boxplot function to display the boxplot data representation for each column.
```

```
display.boxplot(df.filtered$Comp, "Comp")
display.boxplot(df.filtered$Circ, "Circ")
display.boxplot(df.filtered$D.Circ, "D.Circ")
display.boxplot(df.filtered$Rad.Ra, "Rad.Ra")
display.boxplot(df.filtered$Pr.Axis.Ra, "Pr.Axis.Ra")
display.boxplot(df.filtered$Max.L.Ra, "Max.L.Ra")
display.boxplot(df.filtered$Scat.Ra, "Scat.Ra")
display.boxplot(df.filtered$Elong, "Elong")
display.boxplot(df.filtered$Pr.Axis.Rect, "Pr.Axis.Rect")
display.boxplot(df.filtered$Max.L.Rect, "Max.L.Rect")
display.boxplot(df.filtered$Sc.Var.Maxis, "Sc.Var.Maxis")
display.boxplot(df.filtered$Sc.Var.maxis, "Sc.Var.maxis")
display.boxplot(df.filtered$Ra.Gyr, "Ra.Gyr")
display.boxplot(df.filtered$Skew.Maxis, "Skew.Maxis")
display.boxplot(df.filtered$Skew.maxis, "Skew.maxis")
display.boxplot(df.filtered$Kurt.maxis, "Kurt.maxis")
display.boxplot(df.filtered$Kurt.Maxis, "Kurt.Maxis")
display.boxplot(df.filtered$Holl.Ra, "Holl.Ra")

# REMOVING THE OUTLIERS FROM THE DATASET
# Discarding the outliers from the data-set,
# any value greater than bench.mark value will be replace with the bench mark value

remove.outliers = function(data, column.name){
  # filter with box plot and trimming out from
  # "maximum": Q3 + 1.5*IQR
  # "minimum": Q1 -1.5*IQR
  # where interquartile range (IQR): 25th to the 75th percentile.

  # Calculating the upper and lower limit for the data
  bench.mark.upper = quantile(data, 0.75) + (1.5 * IQR(data))
  bench.mark.lower = quantile(data, 0.25) - (1.5 * IQR(data))

  # Replacing the outliers with the upper and lower limits
  data[data > bench.mark.upper] = bench.mark.upper
  data[data < bench.mark.lower] = bench.mark.lower

  # Display the box-plot after removing the outlier
  display.boxplot(data, column.name)

}

# calling the remove.outliers function to remove all the outliers from each column of the data
remove.outliers(df.filtered$Comp, "Comp")
remove.outliers(df.filtered$Circ, "Circ")
remove.outliers(df.filtered$D.Circ, "D.Circ")
remove.outliers(df.filtered$Rad.Ra, "Rad.Ra")
remove.outliers(df.filtered$Pr.Axis.Ra, "Pr.Axis.Ra")
remove.outliers(df.filtered$Max.L.Ra, "Max.L.Ra")
remove.outliers(df.filtered$Scat.Ra, "Scat.Ra")
remove.outliers(df.filtered$Elong, "Elong")
remove.outliers(df.filtered$Pr.Axis.Rect, "Pr.Axis.Rect")
remove.outliers(df.filtered$Max.L.Rect, "Max.L.Rect")
remove.outliers(df.filtered$Sc.Var.Maxis, "Sc.Var.Maxis")
remove.outliers(df.filtered$Sc.Var.maxis, "Sc.Var.maxis")
remove.outliers(df.filtered$Ra.Gyr, "Ra.Gyr")
remove.outliers(df.filtered$Skew.Maxis, "Skew.Maxis")
remove.outliers(df.filtered$Skew.maxis, "Skew.maxis")
remove.outliers(df.filtered$Kurt.maxis, "Kurt.maxis")
remove.outliers(df.filtered$Kurt.Maxis, "Kurt.Maxis")
remove.outliers(df.filtered$Holl.Ra, "Holl.Ra")


# NORMALIZING THE DATASET (BRINGING ALL THE DATA INTO A SINGLE UNQIUE SCALE)
# Performing normalization using the Z-Score Standardization [STANDARD NORMALIZATION]
df.normalized = as.data.frame(scale(df.filtered))
View(df.normalized)


# THIS SECTION IS COMMENTED OUT BECAUSE THE CW SPECIFICATION MENTIONED THAT
# ASSUME PCA IS NOT USED  FOR THIS PROBLEM (BY TAKING ALL THE INITAIL FEATURES
# FOR PREDICTION).
#------------------------------BEGINING OF PERFORMING PCA-----------------------
# PERFORMING PCA (PRINCIPAL COMPONENT ANALYSIS) / DIMENSIONALITY REDUCTION
# df.pca = prcomp(df.normalized)
# summary(df.pca)
#
# # Plotting the PCA data to find the best number of Principal Components.
# # Using the elbow method of the plot below we can get the number of components which
# # explain 85% or greater of the variation (BEST SET OF COMPONENTS TO TAKE)
# # In this case the first 4 components are the best, because it covers the greatest
# # area of the graph and has the sudden decrease after the 4th component
# plot(df.pca)
# plot(df.pca, type='l')
#
# # comp.data contains the BEST PCA Component data extract
# comp.data = data.frame(df.pca$x[,1:4])
# View(comp.data)
#------------------------------END OF PERFORMING PCA----------------------------


# DETERMINE THE NUMBER OF CLUSTERS CENTERS (CENTROIDS) (via MANUAL and AUTOMATED
TOOLS)
```

```
# AUTOMATED TOOLS TO FIND THE CENTROIDS


# Using NbClust()

# Using Euclidean for distance (Gave 2)

cluster_euclidean = NbClust(df.normalized, distance = "euclidean", min.nc = 2, max.nc = 10, method =
"kmeans",

                    index = "all")


# Using Manhattan for distance (Gave 2)

cluster_manhattan = NbClust(df.normalized, distance = "manhattan", min.nc = 2, max.nc = 10, method =
"kmeans",

                    index = "all")

# Using fviz_nbclust()

# USING ELBOW METHOD (Gave 3)

# The below method points out that 3 is the optimal number of centroids/clusters to be taken

fviz_nbclust(df.normalized, kmeans, method = "wss") +

  geom_vline(xintercept = 3, linetype = 2) +

  labs(subtitle = "Elbow method")


# USING THE SILHOUETTE METHOD (Gave 2)

# The below method points out that 2 is the optimal number of centroids/clusters to be taken

fviz_nbclust(df.normalized, kmeans, method = "silhouette")+

  labs(subtitle = "Silhouette method")


# USING GAP STATISTIC ( nboot = 50 to keep the function speedy

# recommended value: nboot= 500 for your analysis.

# Use verbose = FALSE to hide computing progression.)

# (Gave 3)

# The below method points out that 3 is the optimal number of centroids/clusters to be taken

fviz_nbclust(df.normalized, kmeans, nstart = 50,  method = "gap_stat", nboot = 50)+
  labs(subtitle = "Gap statistic method")



# MANUALLY FIND THE CENTROIDS / CLUSTERS


# USING ELBOW METHOD
tot.withinss = vector(mode = "character", length = 10)


# Classification Report Function
classification_report <- function(comparison_table, dp = 2) {
  #total counts
  counts <- sum(comparison_table)

  #total sums for each column
  column_sums <- colSums(comparison_table)

  #total sums for each row
  row_sums <- rowSums(comparison_table)

  #true positive value
  tp <- diag(comparison_table)

  #true negative
  tn <- counts - (column_sums + row_sums - tp)

  #false positive
  fp <- row_sums - tp

  #false negative
  fn <- column_sums - tp
  #precision
  pr <- tp / (tp + fp)

  #recall
  re <- tp / (tp + fn)

  #accuracy
  ac <- sum(tp) / counts

  # Displaying the Accuracy, Precision and Recall values
  print("---------------------")
  print("Accuracy")
  print(ac)

  print("---------------------")
  print("Precision")
  print(pr)

  print("---------------------")
  print("Recall")
  print(re)

}

# Manual Finding clusters
# Looping from 1 to the max optimal cluster to find its evaluation result
for (i in 1:10){
  cat("<=============== ", "Custer ", i, " ===============>\n", sep = "")
  set.seed(150)
  # Performing Kmeans clustering
  vehicleCluster = kmeans(df.normalized, centers = i, nstart = 20)

  set.seed(50)# This is the confusion matrix
  cm = as.matrix(table(Actual = df$Class, Predicted = vehicleCluster$cluster))
  print("Confusion Matrix")
  print(cm)

  # Display Classification report
  classification_report(comparison_table = cm)

  # Total within-cluster sum of squares
  tot.withinss[i] = vehicleCluster$tot.withinss
}

# plot to find the best number of clusters to be taken
plot(1:10,
    tot.withinss,
    type="b",
    pch=19,
    xlab = "Number of clusters K",
    ylab = "Total within-clusters sum of squares")

# from the accuracy result we can see that we got the highest accuracy result for
# 2 clusters (36%)
set.seed(150)
vehicleCluster = kmeans(df.normalized, centers = 2, nstart = 20)
fviz_cluster(vehicleCluster, data = df.normalized)
```

```
# Displaying the sizes(number of observations in each cluster) of each cluster

vehicleCluster$size


# Displaying the cluster distribution

vehicleCluster$cluster


# Getting the centers (A matrix of cluster centers).

vehicleCluster$centers

View(vehicleCluster$centers)




# References used

# https://www.r-bloggers.com/2014/06/pca-and-k-means-clustering-of-delta-aircraft/

# https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-
methods/

# https://rpubs.com/Nitika/kmeans_Iris

# https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/

# https://uc-r.github.io/kmeans_clustering

# https://www.guru99.com/r-k-means-clustering.html
```

CODE FOR PART 02 (FORECASTING QUESTION)

```
# ------------------------------------------------

# Name: Mohammed Nazhim Kalam

# Student ID: 2019281

# UoW ID: W1761265

# ------------------------------------------------


# FORECASTING PART

# Installing package to read Excel Data-set

# install.packages("fpp")

# install.packages("MASS")

# install.packages("readxl")

# install.packages("neuralnet")

# install.packages("ggplot2")

# install.packages("reshape2")

# install.packages("gridExtra")

# install.packages("fpp2")

# install.packages("e1071")

# install.packages("openxlsx")

# install.packages("MLmetrics")

# install.packages("lubridate")

# install.packages("Metrics")

# install.packages("tidyr")

# install.packages("graphics")


# Loading the package

library(fpp)

library(MASS)

library(readxl)

library(neuralnet)
library(ggplot2)

library(reshape2)

library(gridExtra)

library(fpp2)

library(e1071)

library(openxlsx)

library(MLmetrics)

library(lubridate)

library(Metrics)

library(tidyr)

library(graphics)



# Reading the data-set "vehicles.xlsx"

df = read_excel("./ExchangeUSD.xlsx")

View(df)


# Checking for null values present from the dataset

print(sum(is.na(df)))


# Checking for the summary of the data

print(summary(df))


# Dropping unwanted columns

df = subset(df, select = -c(Wdy, `YYYY/MM/DD`))

df$Rate = df$`USD/EUR`

df_copy = df


# Renaming the Columns of the Data-frame

df = setNames(df, c("Rate_Original", "Rate_Lag"))
```

```
View(df)

# Variables used for 1 hidden and 2 hidden layers for the neural network
# NOTE: One of the two sets have to be commented while the other set is used for the model training

# # THIS SET OF VARIABLES IS USED FOR 1 HIDDEN LAYER MLP
# HIDDEN_LAYERS = c(6)
# ACTIVATION_FUNCTION = "logistic"
# LEARNING_RATE = 0.1

# THIS SET OF VARIABLES ARE USED FOR THE 2 HIDDEN LAYER MLP
HIDDEN_LAYERS = c(6,6)
ACTIVATION_FUNCTION = "logistic"
LEARNING_RATE = 0.08

# Looping the AR Order from 1 to 10 to get the one which performs the best
for (index in 1:10) {
  # Using the saved dataframe copy
  df = df_copy

  # Renaming the Columns of the Data-frame
  df = setNames(df, c("Rate_Original", "Rate_Lag"))

  # Shifting the Rate_Lag column rows by one down below for every loop
  for (loop in 1:index) {
    df['Rate_Lag'] <- c(NA, head(df['Rate_Lag'], dim(df)[1] - 1)[[1]])
  }

  # Removing the first row from the dataframe because there is a null value present in the Rate_Lag
column
```

```
df = drop_na(df)


# normalization
normalize = function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}


# normalized data
df.normalized = data.frame(lapply(df, normalize))
View(df.normalized)


# Creating the Training Data
training_data = df.normalized[1:400,]


# Creating the Testing Data
testing_data = df.normalized[401:500-index,]
View(testing_data)


# Training a model on the data
set.seed(101)


# USED FOR TUNING THE NEURAL NET PARAMETERS IN ORDER TO GET THE BEST SET OF PARAMETERS
FOR THE MODEL
# PLEASE DONT UN-COMMENT AND RUN THIS BECAUSE THIS WAS USED ONLY FOR GETTING THE BEST
SET OF NODES
# Loop from 1 to 10 node for one and two hidden layer to get the best node number for each layer
# for (x in 1:10) {
#   print("-----------------------------------------")
#   model <- neuralnet(Rate_Original~Rate_Lag,
#                   hidden=c(x),
#                   # hidden=c(6,x),
#                   data = training_data,
#                   act.fct = "logistic",
#                   linear.output = TRUE,
#                   err.fct = "sse",
#                   learningrate = 0.1)
#
#   predict_result = predict(model, testing_data)
#   View(predict_result)
#
#   # Evaluating the model
#   actual = data.frame(testing_data)
#   predicted = predict_result
#
#   print(paste("Nodes:", x))
#   # Calculating the Mean Absolute Error
#   mae = round(mae(actual$Rate_Original, predicted) * 100, digits = 4)
#   print(paste("Mean Absolute Error: ", mae, " %", sep = ""))
#
#   # Calculating the Root Mean Squared Error
#   rmse = round(rmse(actual$Rate_Original, predicted) * 100, digits = 4)
#   print(paste("Root Mean Squared Error: ", rmse, " %", sep = ""))
#
#   # Calculating the Mean Absolute Percentage Error Loss
#   mape = round(MAPE(actual$Rate_Original, predicted) * 100, digits = 4)
#   print(paste("Mean Absolute Percentage Error Loss: ", mape, " %", sep = ""))
# }


 # PLEASE DONT UN-COMMENT AND RUN THIS BECAUSE THIS WAS USED ONLY TO GET THE BEST
LEARNING RATE

 # Looping to get the best learning rate which gives the least amount of error rate for the
# model (this includes 1 layer and 2 layer model with 6 nodes for both layers since that
# is the optimal number of nodes found previously)
# learning_rate = 0
# while (learning_rate <= 0.1) {
#   learning_rate = learning_rate + 0.02
#   print(learning_rate)
#   model <- neuralnet(Rate_Original~Rate_Lag,
#                      hidden=c(6),
#                      # hidden=c(6,6),
#                      data = training_data,
#                      act.fct = "logistic",
#                      linear.output = TRUE,
#                      err.fct = "sse",
#                      learningrate = learning_rate)
#
#   predict_result = predict(model, testing_data)
#   View(predict_result)
#
#   # Evaluating the model
#   actual = data.frame(testing_data)
#   predicted = predict_result
#
#   print(paste("learning_rate:", learning_rate))
#   # Calculating the Mean Absolute Error
#   mae = round(mae(actual$Rate_Original, predicted) * 100, digits = 4)
#   print(paste("Mean Absolute Error: ", mae, " %", sep = ""))
#
#   # Calculating the Root Mean Squared Error
#   rmse = round(rmse(actual$Rate_Original, predicted) * 100, digits = 4)
#   print(paste("Root Mean Squared Error: ", rmse, " %", sep = ""))
#

#   # Calculating the Mean Absolute Percentage Error Loss

#   mape = round(MAPE(actual$Rate_Original, predicted) * 100, digits = 4)

#   print(paste("Mean Absolute Percentage Error Loss: ", mape, " %", sep = ""))

# }


# PLEASE DONT UN-COMMENT AND RUN THIS BECAUSE THIS WAS USED ONLY TO CHECK WHICH
ACTIVATION FUNCTION

# PERFORMS BEST WITH THE FOUND BEST NODES AND LEARNING RATE VALUES

# activationFunction = "logistic"

# activationFunction = "tanh"

# model <- neuralnet(Rate~.,

#           # hidden=c(6),

#           hidden=c(6,6),

#           data = training_data,

#           act.fct = activationFunction,

#           linear.output = TRUE,

#           err.fct = "sse",

#           lifesign = "full",

#           learningrate = 0.1)

# }


# with one hidden layer and 6 nodes & two hidden layer with 6,6 nodes for the layers


# OPTIMUM RESULT FOR 1 HIDDEN LAYER MLP

# 6 NODES

# LEARNING RATE = 0.1

# ACTIVATION FUNCTION = LOGISTIC


# OPTIMUM RESULT FOR 2 HIDDEN LAYER MLP
# 6 6 NODES
# LEARNING RATE = 0.08
# ACTIVATION FUNCTION = LOGISTIC

# Training the model
model <- neuralnet(Rate_Original~Rate_Lag,
            hidden = HIDDEN_LAYERS,
            data = training_data,
            act.fct = ACTIVATION_FUNCTION,
            linear.output = TRUE,
            err.fct = "sse",
            learningrate = LEARNING_RATE)

# testing_data_actual_rate = data.frame(testing_data)
predict_result = predict(model, testing_data)
View(predict_result)

# Evaluating the model
actual = data.frame(testing_data)
predicted = predict_result

# Evaluation for the AR order number
print("------------------------------------------")
print(paste("Evaluation for the AR Order:", index))

# Calculating the Mean Absolute Error
mae = round(mae(actual$Rate_Original, predicted) * 100, digits = 4)
print(paste("Mean Absolute Error: ", mae, " %", sep = ""))

# Calculating the Root Mean Squared Error
```

```
  rmse = round(rmse(actual$Rate_Original, predicted) * 100, digits = 4)
  print(paste("Root Mean Squared Error: ", rmse, " %", sep = ""))

  # Calculating the Mean Absolute Percentage Error Loss
  mape = round(MAPE(actual$Rate_Original, predicted) * 100, digits = 4)
  print(paste("Mean Absolute Percentage Error Loss: ", mape, " %", sep = ""))
}

# Since AR1 gave the best result we will be using that
# Using the saved dataframe copy
df = df_copy

# Renaming the Columns of the Data-frame
df = setNames(df, c("Rate_Original", "Rate_Lag"))

# Shifting the Rate_Lag column rows by one down below
df['Rate_Lag'] <- c(NA, head(df['Rate_Lag'], dim(df)[1] - 1)[[1]])

# Removing the first row from the dataframe because there is a null value present in the Rate_Lag
column
df = drop_na(df)

# normalization
normalize = function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# normalized data
df.normalized = data.frame(lapply(df, normalize))
View(df.normalized)
```

```
# Creating the Training Data
training_data = df.normalized[1:400,]

# Creating the Testing Data
testing_data = df.normalized[401:499,]
View(testing_data)

# Training a model on the data
set.seed(101)

# Training the model
model <- neuralnet(Rate_Original~Rate_Lag,
            hidden = HIDDEN_LAYERS,
            data = training_data,
            act.fct = ACTIVATION_FUNCTION,
            linear.output = TRUE,
            err.fct = "sse",
            learningrate = LEARNING_RATE)

# Plotting the model network structure
plot(model)

# testing_data_actual_rate = data.frame(testing_data)
predict_result = predict(model, testing_data)

# Adding the index column
testing_data$Index = seq.int(nrow(testing_data))

# Plotting the graph
```

```
plot(testing_data$Index,testing_data$Rate_Original,
    main = "Actual VS Predicted", xlab = "Index",
    ylab = "Rate", col = "black", type = "l")
lines(testing_data$Index, predict_result, col="red")
legend("bottomright",                  # Add legend to plot
      legend = c("ACTUAL", "PREDICTED"),
      col = 1:2,
      lty = 1,
      cex = 0.50)

# Evaluating the model
actual = data.frame(testing_data)
predicted = predict_result

# Displaying the predicted and the desired output
predicted_desired_output = data.frame(testing_data$Rate_Original, predicted)
predicted_desired_output = setNames(predicted_desired_output, c("Desired_Output",
"Predicted_Output"))
View(predicted_desired_output)

# Calculating the Mean Absolute Error
mae = round(mae(actual$Rate_Original, predicted) * 100, digits = 4)
print(paste("Mean Absolute Error: ", mae, " %", sep = ""))

# Calculating the Root Mean Squared Error
rmse = round(rmse(actual$Rate_Original, predicted) * 100, digits = 4)
print(paste("Root Mean Squared Error: ", rmse, " %", sep = ""))

# Calculating the Mean Absolute Percentage Error Loss
mape = round(MAPE(actual$Rate_Original, predicted) * 100, digits = 4)
```

```
print(paste("Mean Absolute Percentage Error Loss: ", mape, " %", sep = ""))



# Best total number of nodes for hidden layer 1 is 6

# Best total number nodes for hidden layer 2 is 6



# References Used

# https://github.com/EviSfn/MLP-neural-network/blob/master/neuralnetwork.R

# https://github.com/alexsnow348/Exchange-Rate-Forecasting-Using-Ensemble-ANN-
Model/blob/master/MLP.R

# https://github.com/cran/nnfor/blob/master/R/mlp.R

# https://www.rdocumentation.org/packages/lubridate/versions/1.7.10/topics/decimal_date

# https://www.gormanalysis.com/blog/dates-and-times-in-r-without-losing-your-sanity/
```

REFERENCES

Brownlee, Jason. ‘How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification’. Machine Learning Mastery, 2 Jan. 2020, https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/.

Dimensionality Reduction Is Good or Bad’. Data Science, Analytics and Big Data Discussions, 24 July 2015, https://discuss.analyticsvidhya.com/t/dimensionality-reduction-is-good-or-bad/2444.

Principal Component Analysis - Javatpoint’. Www.Javatpoint.Com, https://www.javatpoint.com/principal-component-analysis. Accessed 12 Apr. 2021

Stöttner, Timo. ‘Why Data Should Be Normalized before Training a Neural Network’. Medium, 16 May 2019, https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d

Harrison, Myles. PCA and K-Means Clustering of Delta Aircraft | R-Bloggers. 23 June 2014, https://www.r-bloggers.com/2014/06/pca-and-k-means-clustering-of-delta-aircraft/

Determining The Optimal Number Of Clusters: 3 Must Know Methods’. Datanovia, https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/. Accessed 12 Apr. 2021

RPubs - k-Means Clustering in R for Beginners. https://rpubs.com/Nitika/kmeans_Iris. Accessed 12 Apr. 2021

K-Means Clustering in R: Algorithm and Practical Examples’. Datanovia, https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/. Accessed 12 Apr. 2021

K-Means Clustering in R with Example. https://www.guru99.com/r-k-means-clustering.html. Accessed 12 Apr. 2021