5DATA001C.2 Machine Learning and Data Mining Course Work

Name: Dhanuja Udurawana
Student ID: 20212043
UOW ID: W1871491
Course: B.Eng. in Software Engineering
Submission: Course Work

Table of Contents

Machine Learning and Data Mining ... 4
Partition Clustering ... 4
Subtask 01 ... 4
Pre-Processing ... 4
Data Cleaning ... 4
Outliers ... 4
Determining Number of Cluster Centers ... 5
NBclust ... 5
Elbow ... 5
Gap Statistics ... 6
Silhouette ... 6
K-Means Clustering ... 7
Silhouette Plot ... 9
Subtask 02 ... 10
PCA ... 10
Determining Number of clusters (PCA based Centers) ... 10
NBclust ... 10
Elbow ... 11
Gap Statistics ... 11
Silhouette ... 12
K means Clustering (PCA based Dataset) ... 12
Silhouette Plot (PCA based Dataset) ... 14
Calinski- Harabasz Index ... 14
Energy Forecasting ... 15
Input Vector Definition Methods for Electricity Load Forecasting ... 15
I/O Matrix Normalization ... 15
AR Approach ... 16
NARX Approach ... 16
Time Delayed I/O Matrix ... 17
AR Approach ... 17
NARX Approach ... 18
Training and Testing MLP Model ... 18
AR Approach ... 18

NARX Approach 21
Standard Statistical Indices 22
AR Approach 23
NARX Approach 23
Performance Comparison 24
AR Approach 24
NARX Approach 25
NARX Findings 25
Best Performing Model 26
Appendix 27
Partition Clustering 27
Energy Forecasting 34
References 51

Machine Learning and Data Mining

Partition Clustering

Subtask 01

Pre-Processing

Prior to Preprocessing the data set as given for in the coursework requirements we must select the given 18 attributes. Hence the first and last column were removed.

Data Cleaning

In order to ensure the accuracy and reliability of the data, it is important to check for any missing values in the Vehicle Dataset provided for this coursework. Hence as the first task the missing values in the dataset were checked and in the event that there were any of those values were removed using the code below.

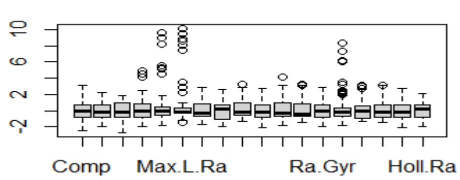
```
vechicles <- vechicles[complete.cases(vechicles), ]     # Check for missing values

vechicles <- na.omit(vechicles)   # If there are missing values, remove them
```

Afterwards the Dataset was normalized using **scale()** function in R. This was done to ensure that all features contribute equally to the clustering, improve performance and interpretability of the algorithm.

```
vechicles <- scale(vechicles)
```

Outliers

Succeeding the normalization of the dataset, the vehicle dataset underwent outlier detection to identify and remove extreme values. Outlier detection and removal must be done due to outliers having significant impact on the dataset such it may produce inaccurate results. In the vehicle dataset outlier detection was done utilizing z scores. Below plot depicts the dataset with outliers.

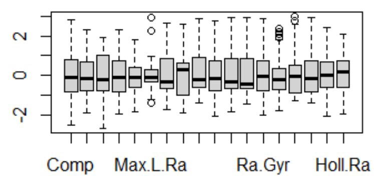


The following code was used to remove the above shown outliers.

```
vechicle_outliers <- apply(vechicles,1,function(x) any(x > 3 | x < -3))
```

```
cleaned_vechicles  <- subset(vechicles,  !vechicle_outliers)
```

After the removal of extreme outliers from the dataset, the newly plotted plot provides a more accurate representation of the data.



Determining Number of Cluster Centers

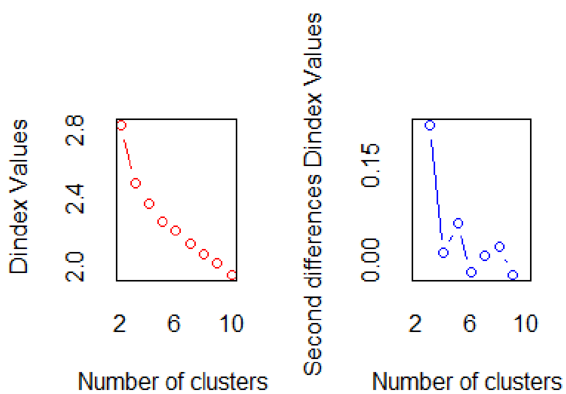
Prior to any sort of cluster analysis, we must first determine the number of cluster centers. When the number of cluster centers is too small the clusters might have the possibility of being too general however, if the number of cluster centers are too large the clusters might be too specific and will be unable to provide any useful information. Hence, we must find the optimal number of clusters. In order to achieve that various clustering solutions will be evaluated using appropriate methods. In this coursework we use NBclust, Elbow, Gap Statistics and Silhouette methods.

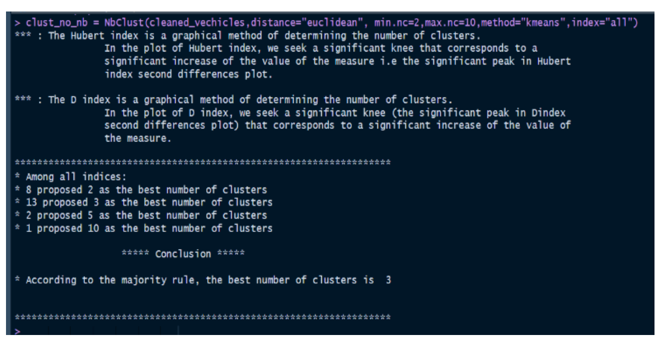
NBclust

NBclust method is an algorithm used in the R NBclust package for evaluating the optimal number of clusters in a dataset using a variety of clustering validation indices. Below code is used in the given coursework.

```
clust_no_nb = NbClust(cleaned_vechicles,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")
```

By running this we can get the following plot.





```
> clust_no_nb = NbClust(cleaned_vechicles,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 corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

*******************************************************************
* Among all indices:
* 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 10 as the best number of clusters

                   ***** Conclusion *****

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


*******************************************************************
>
```

Since the point where the rate of increase slows down significantly is 3 we can come to the conclusion that optimal number of clusters according to NBclust is 3.

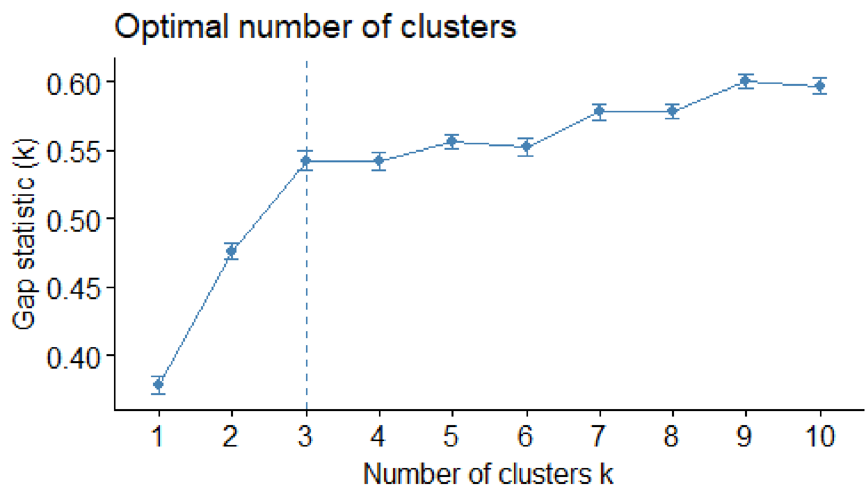
Elbow

Gap Statistics

The gap statistic method compares the within-cluster variation of the original dataset to that of a reference null distribution to determine the optimal number of clusters. Below code can be used to plot a gap stat graph.

fviz_nbclust(cleaned_vechicles,kmeans,method = "gap_stat")

Using the graph we are able to find out the optimal number of clusters.



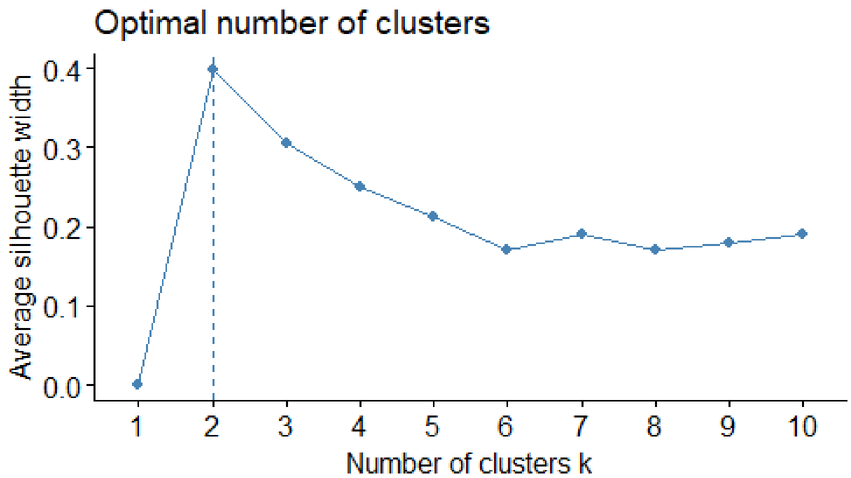
After analyzing the above graph we can determine that the optimal number of clusters using gap stat is 3.

Silhouette

The silhouette method is a technique used to evaluate the quality of clustering results by assigning a silhouette coefficient to each data. The theory behind the silhouette method is that a good clustering solution should have a high average silhouette coefficient. Below code was used to plot the silhouette graph.

fviz_nbclust(cleaned_vechicles, kmeans, method = "silhouette")

The graph drawn is below.



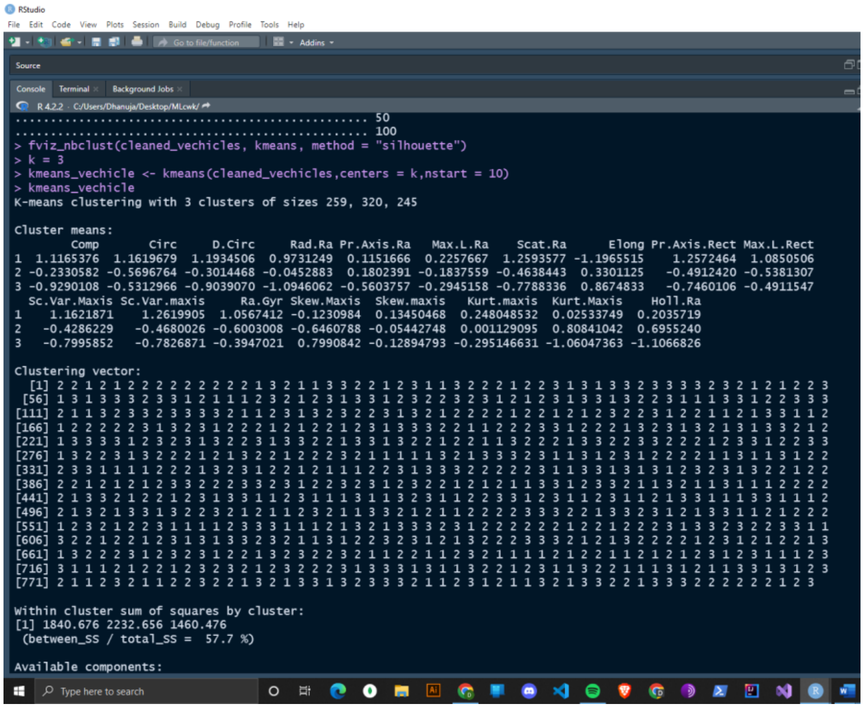
According to the plot we can determine that the optimal number of clusters is 2.

According to the majority rule we can determine that the optimal number of clusters is 3.

K-Means Clustering

After obtaining the optimal among of clusters we can proceed to clustering procedure. For this coursework we intend to use K means clustering a unsupervised learning algorithm used in partition clustering. As we obtain the cleaned dataset and optimal number of clusters the following code can be used to perform K means clustering.

```
k = 3

kmeans_vechicle <- kmeans(cleaned_vechicles,centers = k,nstart = 10)
```



Below Is the plot drafted for the clusters.

Cluster Centers

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

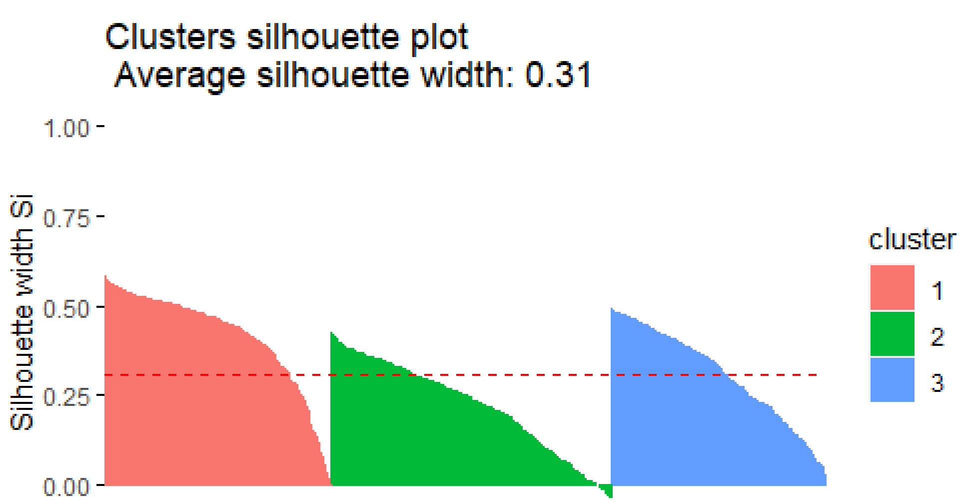
```
  cluster
1       2
2       2
3       1
4       2
5       1
6       2
>
> vechicle_wss = kmeans_vechicle$tot.withinss
> vechicle_wss
[1] 5533.808
>
> vechicle_bss = kmeans_vechicle$betweenss
> vechicle_bss
[1] 7561.925
>
> # Calculating the ratio between WSS and BSS
> vechicle_wss/vechicle_bss
[1] 0.7317989
>
> # Calculate the total sum of squares (TSS)
> TSS <- sum(apply(cleaned_vechicles, 2, var)) * nrow(cleaned_vechicles)
>
> #Calculating the ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_Squares (TSS)
> TSS/vechicle_bss
[1] 1.733903
>
```

Silhouette Plot

silhouette plot which displays how close each data point in one cluster to data points in the neighboring clusters. Below code was used to plot the silhouette plot.

vehicle_sil = silhouette(kmeans_vechicle$cluster,dist(cleaned_vechicles))
fviz_silhouette(vehicle_sil)

Below is the silhouette plot



In the current dataset, the silhouette plot was used to evaluate the quality of the obtained clusters. Overall, the average silhouette coefficient for the clusters was found to be 0.31 indicating that the clustering solution was moderately accomplished the separation of clusters. Moreover, the silhouette plot indicated that there may have been misclassified data due to the plot having a few low silhouette coefficients.

Subtask 02

PCA

Principal Component Analysis is a statistical technique that used to reduce dimensionality. PCA works by finding the directions of maximum variance in the data and projecting the data onto these directions, therefore reducing the dimensionality of the dataset. In the coursework the below code was used to make PCs.

```
v_pca = prcomp(cleaned_vechicles)
```

Afterwards the PC with cumulative score of at least 92% was chosen.

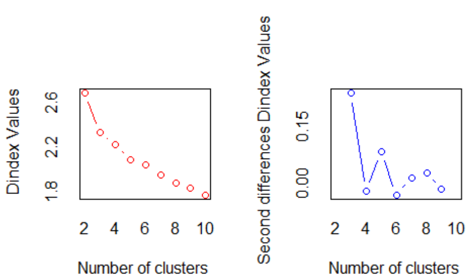
```
#cumulative score per principal components

PVE <- v_pca$sdev^2/sum(v_pca$sdev^2)

PVE <- round(PVE,2)

cum_score = cumsum(PVE)

pc_num = sum(cum_score<0.92) + 1

pc_num


vechicle_pc_dataset = data.frame(vechicle_transformed[, 1:pc_num])
```

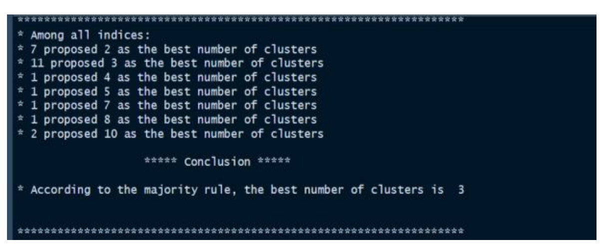
The cumulative score of the first 6 components indicates that these 6 components can explain a significantly high portion of the variability in the dataset. Hence, we can use this component to perform various analysis tasks on the dataset while reducing the dimensionality.

Determining Number of clusters (PCA based Centers)

As previously mentioned, we can find out the optimal number of clusters.

NBclust

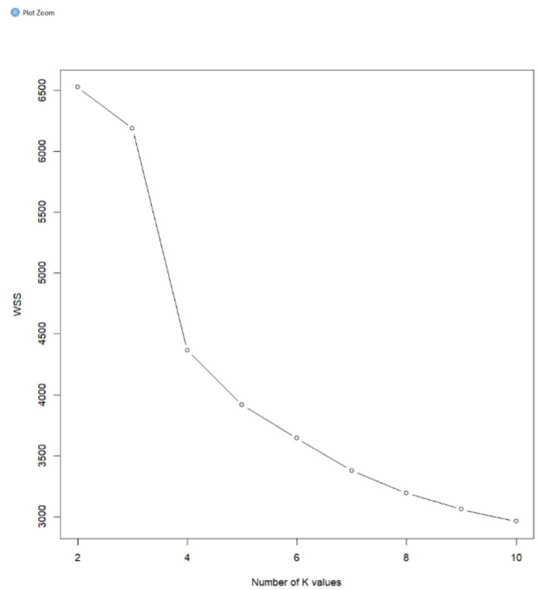




```
*******************************************************************
* Among all indices:
* 7 proposed 2 as the best number of clusters
* 11 proposed 3 as the best number of clusters
* 1 proposed 4 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 1 proposed 7 as the best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

                   ***** Conclusion *****

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


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

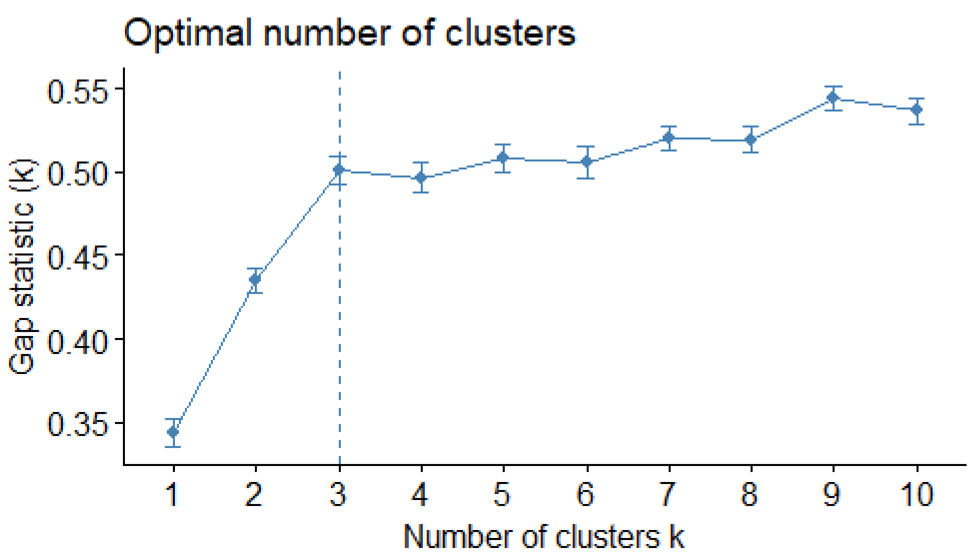
According to NBclust method the optimal number of clusters is 3

Elbow



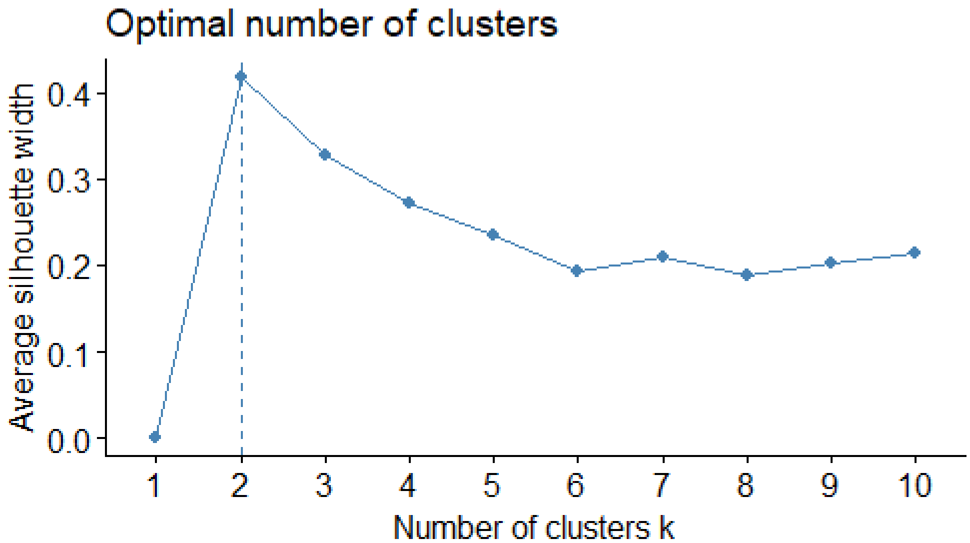
According to elbow method optimal number of clusters is 3

Gap Statistics



According to gap statics method the optimal number of clusters is 3

Silhouette

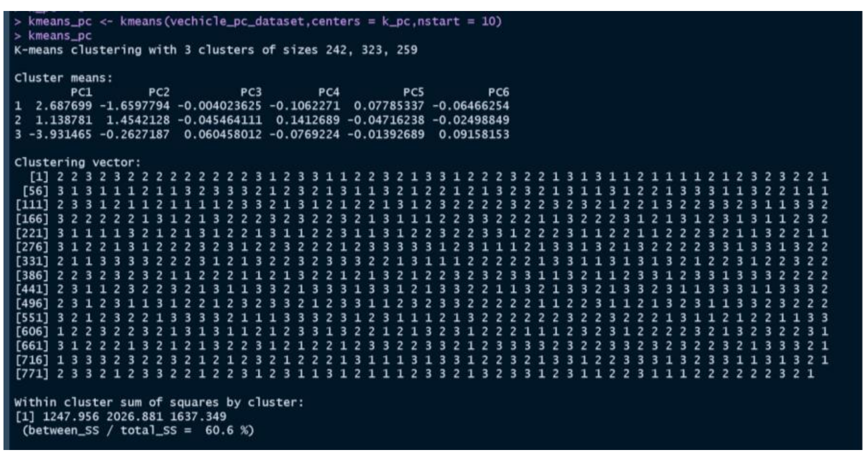


According to the gap statics method the optimal number of clusters is 2.

By applying the majority rule to these values, we can determine that the optimal number of clusters is 3.

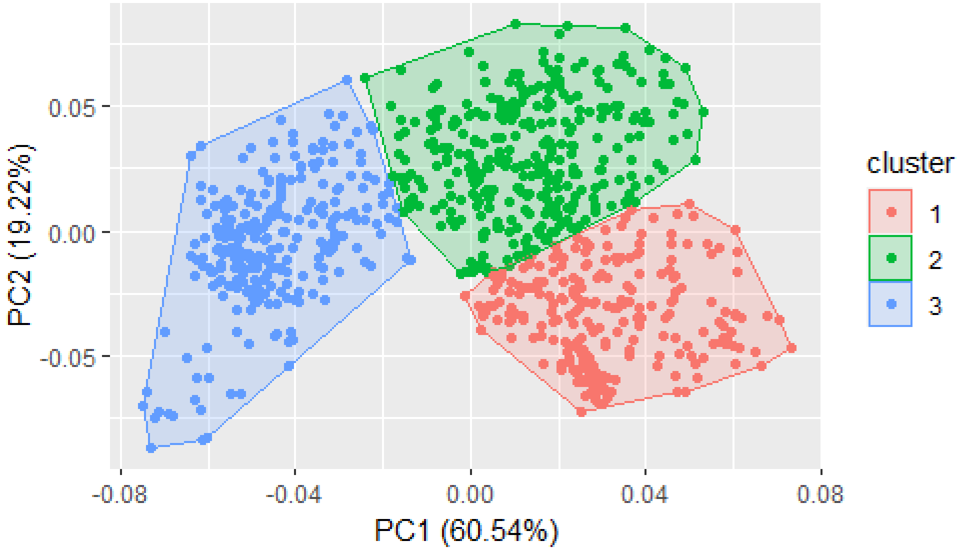
K means Clustering (PCA based Dataset)

Hence, we determine the optimal number of clusters is 3 now we can perform K means analysis on the dataset.



```
> kmeans_pc <- kmeans(vechicle_pc_dataset,centers = k_pc,nstart = 10)
> kmeans_pc
K-means clustering with 3 clusters of sizes 242, 323, 259

Cluster means:
        PC1        PC2          PC3        PC4         PC5         PC6
1  2.687699 -1.6597794 -0.004023625 -0.1062271  0.07785337 -0.06466254
2  1.138781  1.4542128 -0.045464111  0.1412689 -0.04716238 -0.02498849
3 -3.931465 -0.2627187  0.060458012 -0.0769224 -0.01392689  0.09158153

Clustering vector:
  [1] 2 2 3 2 3 2 2 2 2 2 2 2 2 2 3 1 2 3 3 1 1 2 2 3 2 1 3 3 1 2 2 2 3 2 2 1 3 1 3 1 1 2 1 1 1 1 2 1 2 3 2 3 2 2 1
 [56] 3 1 3 1 1 1 2 1 1 3 2 3 3 3 2 1 2 3 2 1 3 1 1 3 2 1 2 2 1 2 1 3 2 3 2 1 3 1 1 3 1 2 2 1 3 3 3 1 1 3 2 2 1 1 1
[111] 2 3 3 1 2 1 1 2 1 1 1 1 2 3 3 2 1 3 1 2 1 2 2 1 3 1 2 3 2 2 2 2 3 2 2 3 2 3 2 1 2 2 1 3 2 2 3 3 2 3 1 1 3 3 2
[166] 3 2 2 2 2 2 1 3 1 2 1 3 2 2 2 3 2 3 2 2 3 2 1 3 1 1 1 2 2 3 3 2 2 2 1 1 3 2 2 2 3 1 2 1 3 1 2 3 1 3 1 1 2 3 2
[221] 3 1 1 1 1 3 2 1 2 1 3 1 2 2 1 3 1 1 2 2 3 1 1 3 1 2 2 3 2 2 3 3 1 2 2 2 3 1 1 2 2 1 1 2 2 2 3 2 1 1 3 2 2 1 1
[276] 3 1 2 2 1 3 1 2 2 2 3 2 3 1 2 2 3 2 2 2 1 2 3 3 3 3 3 1 2 3 1 1 1 2 1 3 3 1 3 2 1 3 2 2 2 2 3 3 1 3 3 1 3 2 2
[331] 2 1 1 3 3 3 3 2 2 2 3 1 2 1 3 2 2 3 2 3 3 3 2 2 1 3 1 1 1 2 2 2 2 2 1 3 3 1 1 3 1 3 1 3 2 1 2 2 3 1 2 2 3 2 2
[386] 2 2 3 2 3 2 3 2 1 1 2 2 2 1 1 2 1 3 2 2 1 2 1 3 2 1 2 2 3 2 3 2 3 3 1 1 3 2 1 1 2 3 3 1 2 3 3 1 3 3 3 2 2 2 2
[441] 2 3 1 1 2 3 2 2 3 2 1 3 1 1 3 3 2 1 3 3 3 1 3 3 1 2 1 3 3 2 2 1 1 3 2 1 3 3 2 1 3 3 2 3 1 1 3 3 3 1 1 3 3 3 2
[496] 2 3 1 2 3 1 1 3 1 2 2 1 2 3 2 3 3 2 1 2 3 3 1 1 2 3 2 3 3 2 2 2 2 2 1 1 2 2 3 1 1 2 1 3 2 3 1 1 3 3 2 3 2 2 2
[551] 3 2 1 2 3 2 2 1 3 3 3 3 2 1 1 1 3 3 3 2 3 1 2 3 1 1 1 2 1 3 2 2 2 2 2 2 3 2 2 3 2 2 2 1 3 1 1 2 1 2 2 1 1 3 3
[606] 1 2 2 3 2 2 3 2 1 3 1 3 1 1 2 1 2 3 3 1 3 2 2 1 2 1 3 2 3 1 2 2 2 1 1 1 2 3 2 3 1 2 2 2 2 3 2 1 3 2 3 2 2 3 1
[661] 3 1 2 2 2 1 3 2 1 2 1 3 2 2 3 1 2 1 2 2 1 2 3 3 2 2 3 3 2 1 2 3 3 3 3 2 3 2 2 3 3 2 3 2 3 2 2 3 2 1 3 3 3 2 1
[716] 1 3 3 3 2 3 2 2 3 2 1 2 1 2 3 2 1 2 2 2 1 3 1 1 1 3 1 3 3 1 2 2 3 2 1 3 3 1 2 2 3 3 3 1 3 2 3 3 1 1 3 1 3 2 1
[771] 2 3 3 2 1 2 3 3 2 2 1 2 2 3 1 2 3 1 1 3 1 2 1 1 1 2 3 3 2 1 3 2 3 3 1 2 3 1 1 2 2 3 1 1 1 2 2 2 2 2 2 3 2 1

Within cluster sum of squares by cluster:
[1] 1247.956 2026.881 1637.349
 (between_SS / total_SS =  60.6 %)
```

Below is the drafted plot.

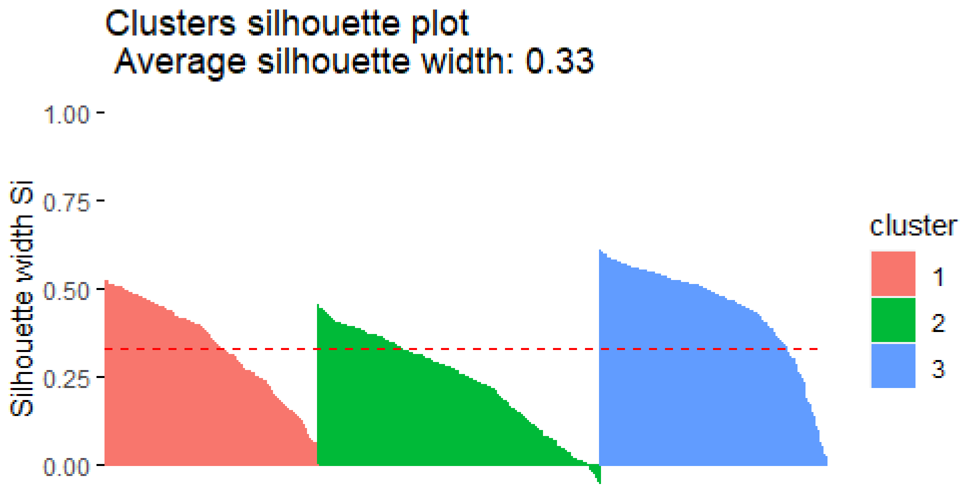


Cluster centers of the PCA dataset

```
Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
> fviz_cluster(kmeans_pc,data=vechicle_pc_dataset)
> kmeans_pc$centers
        PC1        PC2          PC3        PC4         PC5         PC6
1  2.687699 -1.6597794 -0.004023625 -0.1062271  0.07785337 -0.06466254
2  1.138781  1.4542128 -0.045464111  0.1412689 -0.04716238 -0.02498849
3 -3.931465 -0.2627187  0.060458012 -0.0769224 -0.01392689  0.09158153
>
```

```
> vechicle_cluster_pc <- data.frame(vechicle_pc_dataset,cluster = as.factor(kmeans_pc$cluster))
> head(vechicle_cluster_pc)
         PC1        PC2        PC3         PC4        PC5        PC6 cluster
1 -0.4056308  0.2382681 -0.2006852  0.33091947  0.2875534 -1.5750269       2
2  1.5024448  0.4103309 -0.2064772 -0.83587489  0.5467632  0.1011726       2
3 -3.8131027 -0.3246041 -1.1660161 -0.63535494 -0.9079491 -0.1711815       3
4  1.5723187  2.8450209 -0.4215667  0.23561013  0.6577803  0.2614215       2
5 -6.3534175 -4.2537673  0.4564255 -0.13302957 -0.3943263  2.0269191       3
6  0.6353852  2.1784556 -2.1466387  0.02772516 -0.6210915  0.3469327       2
>
> vechicle_wss_pc = kmeans_pc$tot.withinss
> vechicle_bss_pc = kmeans_pc$betweenss
> vechicle_bss_pc
[1] 7555.79
> vechicle_wss_pc
[1] 4912.186
>
> # Calculating the ratio between WSS and BSS
> vechicle_wss_pc/vechicle_bss_pc
[1] 0.6501221
>
> # Calculate the total sum of squares (TSS)
> TSS_pc <- sum(apply(vechicle_pc_dataset, 2, var)) * nrow(vechicle_pc_dataset)
> TSS_pc
[1] 12483.13
>
> #Calculating the ratio of between_cluster_sums_of_squares (BSS) over total_sum_of_Squares (TSS)
> TSS_pc/vechicle_bss_pc
[1] 1.652127
>
```

Silhouette Plot (PCA based Dataset)



In the current dataset, the silhouette plot was used to evaluate the quality of the obtained clusters. Overall, the average silhouette coefficient for the clusters was found to be 0.33 indicating that the clustering solution was moderately accomplished the separation of clusters. Moreover, the silhouette plot indicated that there may have been misclassified data due to the plot having a few low silhouette coefficients. Furthermore, with the PC approach the avg width score has increased hence the clustering results has been improved.

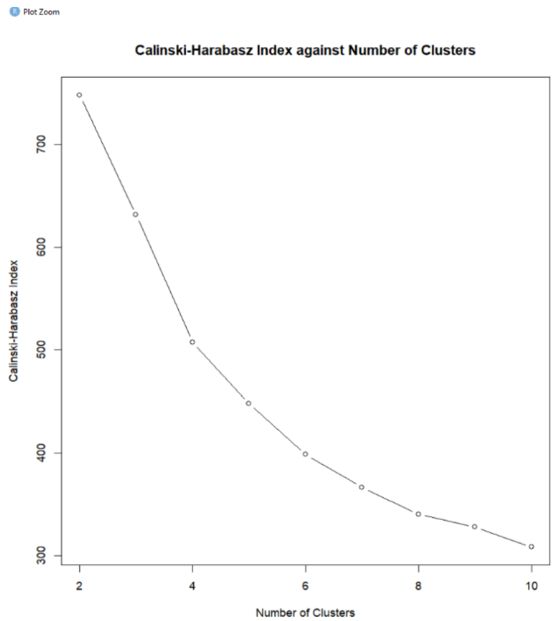
Calinski- Harabasz Index

The Calinski-Harabasz index, also known as the Variance Ratio Criterion, is the ratio of the sum of between-clusters dispersion and of inter-cluster dispersion for all clusters, the higher the score, the better the performance. Hence we aim to have a high Calinski Harbersz index. The below code was used to implement it.

ch_index <- round(calinhara(vechicle_pc_dataset,kmeans_pc$cluster),digits=3)
ch_index

we can compute the calinski – harabasz index for each PC and plot a graph and find optimal number of clusters.

```
R 4.2.2 · C:/Users/Dhanuja/Desktop/MLcwk/
3       3  259          0.44
> # Compute the Calinski-Harabasz Index
> ch_index <- round(calinhara(vechicle_pc_dataset,kmeans_pc$cluster),digits=3)
> ch_index
[1] 631.42
> set.seed(123)
> # Set the number of clusters to evaluate
> k <- 2:10
>
> # Initialize empty vector to store Calinski-Harabasz index values
> ch_scores <- vector("numeric", length(k))
>
> # Compute the Calinski-Harabasz index for each number of clusters
> for (i in 1:length(k)) {
+   km <- kmeans(vechicle_pc_dataset, centers = k[i], nstart = 10)
+   ch_scores[i] <- calinhara(vechicle_pc_dataset, km$cluster)
+ }
>
> ## Visualize the Calinski-Harabasz Index##
> # Plot the Calinski-Harabasz index values
> plot(k, ch_scores, type = "b", xlab = "Number of Clusters", ylab = "Calinski-Harabasz Index", main = "Calinski-Haraba
sz Index against Number of Clusters")
> set.seed(123)
> plot(k, ch_scores, type = "b", xlab = "Number of Clusters", ylab = "Calinski-Harabasz Index", main = "Calinski-Haraba
sz Index against Number of Clusters")
> ch_scores
[1] 747.8705 631.4199 507.4699 448.1050 398.3315 366.4991 340.2745 328.0711 308.2861
>
```



The highest value in the graph is the optimal number of clusters hence the optimal number of clusters according to calinski – harabasz index is 2.

Energy Forecasting

Input Vector Definition Methods for Electricity Load Forecasting

Electricity load forecasting is a common method used to predict future energy demand in the energy industry. Several methods can be utilized to create the input vector in electricity load forecasting. In the input vector of electricity load forecasting can commonly include lagged variables meaning past values of the energy input, weather variables such as temperature, humidity which can affect the energy usage. Moreover, calendar variables can also be used due to alteration of energy usage as days progress. Utilizing these variables electricity load forecasting is done in several schemes. Autoregressive approach is a common schema. It utilizes past input values to forecast future values. The Moving Average approach is an approach that aim to utilize the past forecast errors as inputs to predict the current value. Moreover, the ARIMA models hopes to combine AR and MA approaches. They capture both short term and long-term patterns, however in order to use ARIMA scheme one must need a large amount of input parameters.

I/O Matrix Normalization

Before progressing to build IO matrices firstly, the values must be normalized. This is done in order to ensure the scale of the input and output variables. When the IO matrices contain non

```
# Normalize the data from 0 to 1
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x))) }

# Unnormalize the data
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min )
}
```

scaled data it will have a tendency to be biased towards variables with larger scales. Hence by normalizing we are able to get much more accurate results. Min max normalization is used in this problem statement. The missing values were removed from the dataset as well.

```r
#Cheching for missing values
energyDataSet <- energyDataSet[complete.cases(energyDataSet),]

# If there are missing values, remove them using the na.omit() function
energyDataSet <- na.omit(energyDataSet)
```

AR Approach

During the AR approach the values of 2000h was normalized.

```r
energyDataSet$Date <- ts(energyDataSet$Date)

#Cheching for missing values
energyDataSet <- energyDataSet[complete.cases(energyDataSet),]

# If there are missing values, remove them using the na.omit() function
energyDataSet <- na.omit(energyDataSet)


#Select 20th hour values
targetData <- energyDataSet[,"2000h"]

colnames(targetData) <- "target"

#Noramlizing
targetData <- normalize(targetData)

```

Vales will be denormalized as below,

```r
#Rescale the Data func
target_min <- min(energyDataSet$'2000h')
target_max <- max(energyDataSet$'2000h')

rescale_predict_mlp_v1 <- unnormalize(predict_mlp_v1$net.result,target_min,target_max)
rescale_predict_mlp_v2 <- unnormalize(predict_mlp_v2$net.result,target_min,target_max)
rescale_predict_mlp_v3 <- unnormalize(predict_mlp_v3$net.result,target_min,target_max)
rescale_predict_mlp_v4 <- unnormalize(predict_mlp_v4$net.result,target_min,target_max)
rescale_predict_mlp_v5 <- unnormalize(predict_mlp_v5$net.result,target_min,target_max)
rescale_predict_mlp_v6 <- unnormalize(predict_mlp_v6$net.result,target_min,target_max)
rescale_predict_mlp_v7 <- unnormalize(predict_mlp_v7$net.result,target_min,target_max)
rescale_predict_mlp_v8 <- unnormalize(predict_mlp_v8$net.result,target_min,target_max)
rescale_predict_mlp_v9 <- unnormalize(predict_mlp_v9$net.result,target_min,target_max)
rescale_predict_mlp_v10 <- unnormalize(predict_mlp_v10$net.result,target_min,target_max)
rescale_predict_mlp_v11 <- unnormalize(predict_mlp_v11$net.result,target_min,target_max)
rescale_predict_mlp_v12 <- unnormalize(predict_mlp_v12$net.result,target_min,target_max)
```

NARX Approach

During NARX approach the values of 1800h and 1900h also normalized.

```r

#Creating Loads for 18h and 19h
energyDataSet[,"1800h"] <- normalize(energyDataSet[,"1800h"])
energyDataSet[,"1900h"] <- normalize(energyDataSet[,"1900h"])
```

Vales will be denormalized as below,

```r
#Rescaled Predictions

narx_rescaled_v1 <- unnormalize(narx_prediction_v1$net.result,target_min,target_max)
narx_rescaled_v2 <- unnormalize(narx_prediction_v2$net.result,target_min,target_max)
narx_rescaled_v3 <- unnormalize(narx_prediction_v3$net.result,target_min,target_max)
narx_rescaled_v4 <- unnormalize(narx_prediction_v4$net.result,target_min,target_max)
narx_rescaled_v5 <- unnormalize(narx_prediction_v5$net.result,target_min,target_max)
```

Time Delayed I/O Matrix

Afterwards, we must create time delayed IO matrices. An IO matrix is a matrix of input output values that contain lagged values of the output variable which are used to predict the current value of the input variable. An IO matrix represents the historical relationship between input and output.

AR Approach

In the AR approach we only consider the 2000h values. Hence the IO matrix are created using only 20th hour values. As per the requirement lags of t1 – t4 and t7 were utilized in creating lags for the IO matrices.

```r

|
#creating time delayed loads
load1 <- lag(targetData, 1)
load2 <- lag(targetData, 2)
load3 <- lag(targetData, 3)
load4 <- lag(targetData, 4)
load7 <- lag(targetData, 7)




#creating I/O matrix
io_matrix <- cbind(load1,load2,load3,load4,load7,targetData)
colnames(io_matrix) <- c('t1','t2','t3','t4','t7','target')

io_matrix_v1 <- cbind(load1,load2,load3,load4,targetData)
colnames(io_matrix_v1) <- c('t1','t2','t3','t4','target')


io_matrix_v2 <- cbind(load1,load3,load4,load7,targetData)
colnames(io_matrix_v2) <- c('t1','t3','t4','t7','target')

io_matrix_v3 <- cbind(load2,load3,load4,load7,targetData)
colnames(io_matrix_v3) <- c('t2','t3','t4','t7','target')

```

NARX Approach

In the NARX Approach we utilize the 1800h and 19ooh in addition to the 2000h values. For 19th hour and 18th hour lags prior values of t1-t4 and t7 were used.

```r

####..........................................................................................####
#...............<<<<<<<<<<<<<<<<NARX Approach>>>>>>>>>>>>>>>>>>>.......................................#


#Creating Loads for 18h and 19h
energyDataSet[,"1800h"] <- normalize(energyDataSet[,"1800h"])
energyDataSet[,"1900h"] <- normalize(energyDataSet[,"1900h"])

load6_1 <- lag(energyDataSet[,"1800h"], 1)
load6_2 <- lag(energyDataSet[,"1800h"], 2)
load6_3 <- lag(energyDataSet[,"1800h"], 3)
load6_4 <- lag(energyDataSet[,"1800h"], 4)
load6_7 <- lag(energyDataSet[,"1800h"], 7)


load7_1 <- lag(energyDataSet[,"1900h"], 1)
load7_2 <- lag(energyDataSet[,"1900h"], 2)
load7_3 <- lag(energyDataSet[,"1900h"], 3)
load7_4 <- lag(energyDataSet[,"1900h"], 4)
load7_7 <- lag(energyDataSet[,"1900h"], 7)

#creating I/O matrix
io_matrix_narx <- cbind(load1,load2,load3,load4,load7,load6_1,load6_2,load6_3,load6_4,load6_7,load7_1,load7_2,loac
colnames(io_matrix_narx) <- c('t8_1','t8_2','t8_3','t8_4','t8_7','t6_1','t6_2','t6_3','t6_4','t6_7','t7_1','t7_2'
narx_relation <- as.formula("target~ t8_1 + t8_2 + t8_3 + t8_4 + t8_7 + t6_1 + t6_2 + t6_3 + t6_4 + t6_7 + t7_1 +

io_matrix_v1_narx <- cbind(load3,load4,load7,load6_1,load6_4,load6_7,load7_1,load7_2,load7_7,targetData)
colnames(io_matrix_v1_narx) <- c('t8_3','t8_4','t8_7','t6_1','t6_4','t6_7','t7_1','t7_2','t7_7','target')
narx_relation_v1 <- as.formula("target~ t8_3+t8_4+t8_7+t6_1+t6_4+t6_7+t7_1+t7_2+t7_7")
```

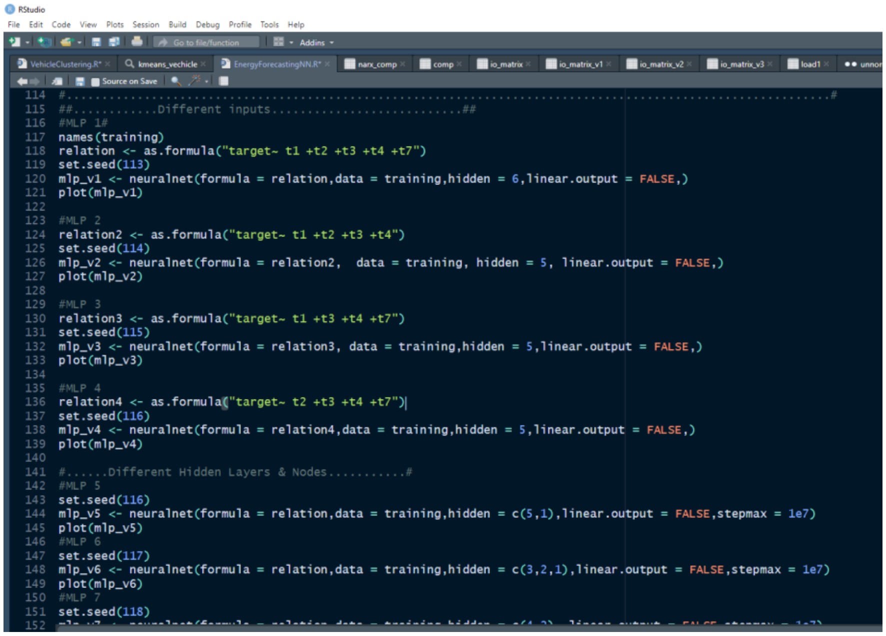
Training and Testing MLP Model

AR Approach

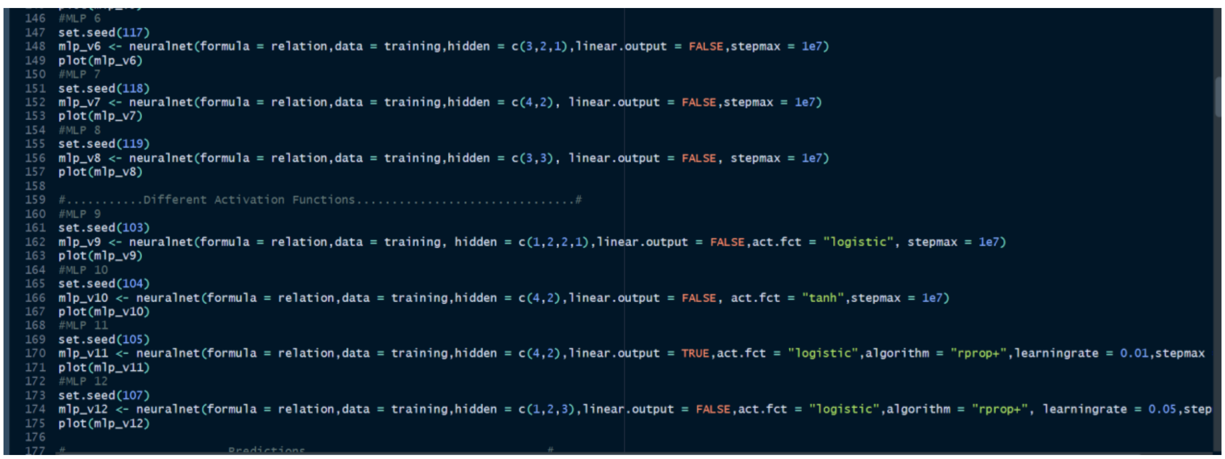
Training and Testing Dataset

```r
#making the testing and training dataset
set.seed(123)

training <- io_matrix[1:380,]

testing <- io_matrix[381:nrow(io_matrix),]

```

MLP Modals



```
#.........................................................................................................#
##...........Different inputs...........................##
#MLP 1#
names(training)
relation <- as.formula("target~ t1 +t2 +t3 +t4 +t7")
set.seed(113)
mlp_v1 <- neuralnet(formula = relation,data = training,hidden = 6,linear.output = FALSE,)
plot(mlp_v1)

#MLP 2
relation2 <- as.formula("target~ t1 +t2 +t3 +t4")
set.seed(114)
mlp_v2 <- neuralnet(formula = relation2,  data = training, hidden = 5, linear.output = FALSE,)
plot(mlp_v2)

#MLP 3
relation3 <- as.formula("target~ t1 +t3 +t4 +t7")
set.seed(115)
mlp_v3 <- neuralnet(formula = relation3, data = training,hidden = 5,linear.output = FALSE,)
plot(mlp_v3)

#MLP 4
relation4 <- as.formula("target~ t2 +t3 +t4 +t7")
set.seed(116)
mlp_v4 <- neuralnet(formula = relation4,data = training,hidden = 5,linear.output = FALSE,)
plot(mlp_v4)

#......Different Hidden Layers & Nodes..........#
#MLP 5
set.seed(116)
mlp_v5 <- neuralnet(formula = relation,data = training,hidden = c(5,1),linear.output = FALSE,stepmax = 1e7)
plot(mlp_v5)
#MLP 6
set.seed(117)
mlp_v6 <- neuralnet(formula = relation,data = training,hidden = c(3,2,1),linear.output = FALSE,stepmax = 1e7)
plot(mlp_v6)
#MLP 7
set.seed(118)
```



```
#MLP 6
set.seed(117)
mlp_v6 <- neuralnet(formula = relation,data = training,hidden = c(3,2,1),linear.output = FALSE,stepmax = 1e7)
plot(mlp_v6)
#MLP 7
set.seed(118)
mlp_v7 <- neuralnet(formula = relation,data = training,hidden = c(4,2), linear.output = FALSE,stepmax = 1e7)
plot(mlp_v7)
#MLP 8
set.seed(119)
mlp_v8 <- neuralnet(formula = relation,data = training,hidden = c(3,3), linear.output = FALSE, stepmax = 1e7)
plot(mlp_v8)

#..........Different Activation Functions..............................#
#MLP 9
set.seed(103)
mlp_v9 <- neuralnet(formula = relation,data = training, hidden = c(1,2,2,1),linear.output = FALSE,act.fct = "logistic", stepmax = 1e7)
plot(mlp_v9)
#MLP 10
set.seed(104)
mlp_v10 <- neuralnet(formula = relation,data = training,hidden = c(4,2),linear.output = FALSE, act.fct = "tanh",stepmax = 1e7)
plot(mlp_v10)
#MLP 11
set.seed(105)
mlp_v11 <- neuralnet(formula = relation,data = training,hidden = c(4,2),linear.output = TRUE,act.fct = "logistic",algorithm = "rprop+",learningrate = 0.01,stepmax
plot(mlp_v11)
#MLP 12
set.seed(107)
mlp_v12 <- neuralnet(formula = relation,data = training,hidden = c(1,2,3),linear.output = FALSE,act.fct = "logistic",algorithm = "rprop+", learningrate = 0.05,step
plot(mlp_v12)
```

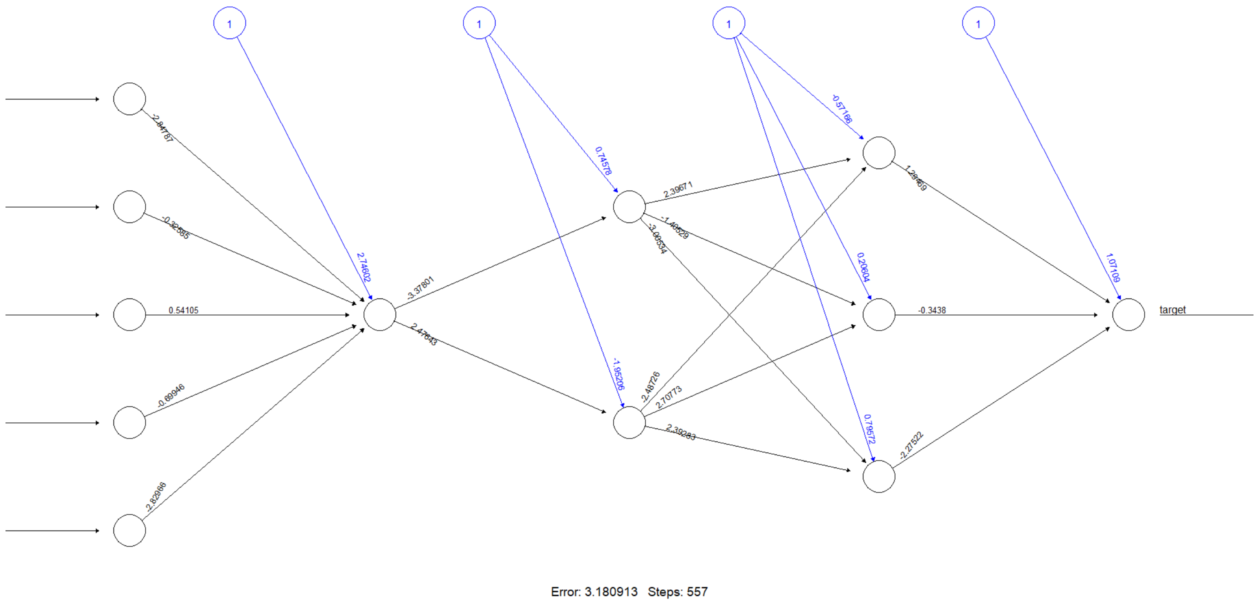


Figure 1: MLP 12

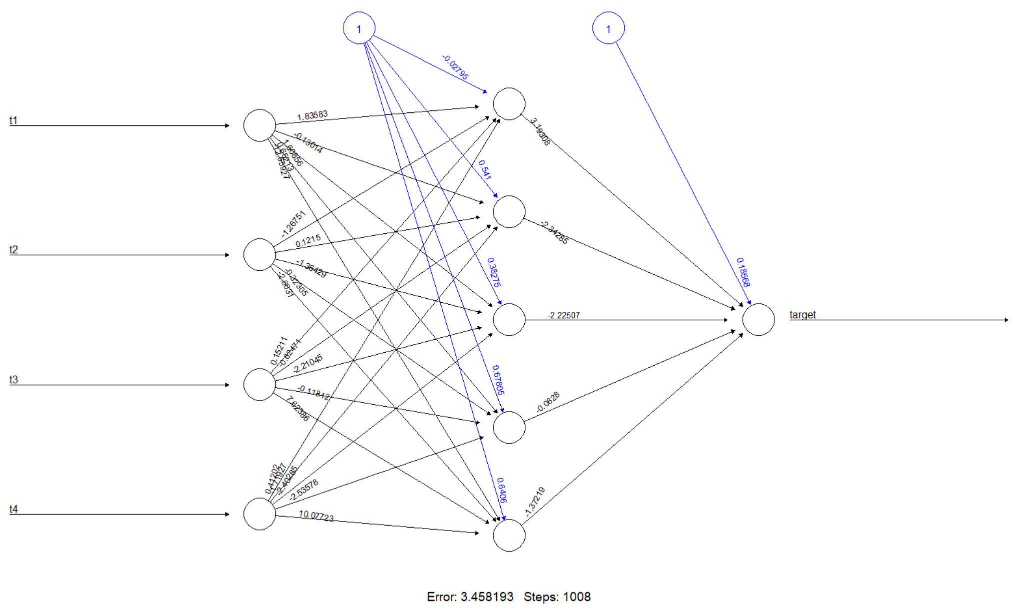


Figure 2: MLP 2

Predictions

```r
mlp_v12 <- neuralnet(formula = relation,data = training,hidden = c(1,2,3),linear.output = FALSE,a
plot(mlp_v12)

#.......................Predictions..................................#

predict_mlp_v1 <- neuralnet::compute(mlp_v1,testing)
predict_mlp_v2 <- neuralnet::compute(mlp_v2,testing)
predict_mlp_v3 <- neuralnet::compute(mlp_v3,testing)
predict_mlp_v4 <- neuralnet::compute(mlp_v4,testing)
predict_mlp_v5 <- neuralnet::compute(mlp_v5,testing)
predict_mlp_v6 <- neuralnet::compute(mlp_v6,testing)
predict_mlp_v7 <- neuralnet::compute(mlp_v7,testing)
predict_mlp_v8 <- neuralnet::compute(mlp_v8,testing)
predict_mlp_v9 <- neuralnet::compute(mlp_v9,testing)
predict_mlp_v10 <- neuralnet::compute(mlp_v10,testing)
predict_mlp_v11 <- neuralnet::compute(mlp_v11,testing)
predict_mlp_v12 <- neuralnet::compute(mlp_v12,testing)

#................Rescled Predictions...............................#
#Rescale the Data func
target_min <- min(energyDataSet$'2000h')
target_max <- max(energyDataSet$'2000h')

rescale_predict_mlp_v1 <- unnormalize(predict_mlp_v1$net.result,target_min,target_max)
rescale_predict_mlp_v2 <- unnormalize(predict_mlp_v2$net.result,target_min,target_max)
rescale_predict_mlp_v3 <- unnormalize(predict_mlp_v3$net.result,target_min,target_max)
rescale_predict_mlp_v4 <- unnormalize(predict_mlp_v4$net.result,target_min,target_max)
rescale_predict_mlp_v5 <- unnormalize(predict_mlp_v5$net.result,target_min,target_max)
rescale_predict_mlp_v6 <- unnormalize(predict_mlp_v6$net.result,target_min,target_max)
rescale_predict_mlp_v7 <- unnormalize(predict_mlp_v7$net.result,target_min,target_max)
rescale_predict_mlp_v8 <- unnormalize(predict_mlp_v8$net.result,target_min,target_max)
rescale_predict_mlp_v9 <- unnormalize(predict_mlp_v9$net.result,target_min,target_max)
rescale_predict_mlp_v10 <- unnormalize(predict_mlp_v10$net.result,target_min,target_max)
rescale_predict_mlp_v11 <- unnormalize(predict_mlp_v11$net.result,target_min,target_max)
rescale_predict_mlp_v12 <- unnormalize(predict_mlp_v12$net.result,target_min,target_max)
```

NARX Approach

Training and Testing Dataset

```r
#Creating Training and testing dataset
narx_training <- io_matrix_narx[1:380,]

narx_testing <- io_matrix_narx[381:nrow(io_matrix_narx),]
```

MLP models

```r
##Create MLP with different parameters
#NARX MLP 1
set.seed(123)
narx_mlp_v1 <- neuralnet(formula = narx_relation,data = narx_training,hidden = 16,linear.output = FALSE,act.fct = "logistic",stepmax = 1e7)
plot(narx_mlp_v1)
#NARX MLP 2
set.seed(123)
narx_mlp_v2 <- neuralnet(formula = narx_relation_v1,data = narx_training, hidden = 10,linear.output = FALSE,stepmax = 1e7)
plot(narx_mlp_v2)
#NARX MLP 3
set.seed(124)
narx_mlp_v3 <- neuralnet(formula = narx_relation_v2,data = narx_training,hidden = c(6,10),linear.output = FALSE,stepmax = 1e7)
plot(narx_mlp_v3)
#NARX MLP 4
set.seed(125)
narx_mlp_v4 <- neuralnet(formula = narx_relation,data = narx_training,hidden = c(1,10,5),linear.output = FALSE, act.fct = "logistic",algorithm = "rprop+",learningr
plot(narx_mlp_v4)
#NARX MLP 5
set.seed(126)
narx_mlp_v5 <- neuralnet(formula = narx_relation,
                         data = narx_training,
                         hidden = c(2,14),
                         linear.output = FALSE,
                         act.fct = "logistic",
                         algorithm = "rprop+",
                         learningrate = 0.05,
                         stepmax = 1e7

)
plot(narx_mlp_v5)
```

Predictions

```r
#Predictions

narx_prediction_v1 <- neuralnet::compute(narx_mlp_v1,narx_testing)

narx_prediction_v2 <- neuralnet::compute(narx_mlp_v2,narx_testing)

narx_prediction_v3 <- neuralnet::compute(narx_mlp_v3,narx_testing)

narx_prediction_v4 <- neuralnet::compute(narx_mlp_v4,narx_testing)

narx_prediction_v5 <- neuralnet::compute(narx_mlp_v5,narx_testing)

#Rescaled Predictions

narx_rescaled_v1 <- unnormalize(narx_prediction_v1$net.result,target_min,target_max)
narx_rescaled_v2 <- unnormalize(narx_prediction_v2$net.result,target_min,target_max)
narx_rescaled_v3 <- unnormalize(narx_prediction_v3$net.result,target_min,target_max)
narx_rescaled_v4 <- unnormalize(narx_prediction_v4$net.result,target_min,target_max)
narx_rescaled_v5 <- unnormalize(narx_prediction_v5$net.result,target_min,target_max)
```

Standard Statistical Indices

During this coursework, in order to evaluate the testing performance, we used 4 different statical indexes namely, MAE, RMSE, MAPE, SMAPE.

MAE: The mean absolute error is a measure of the errors between paired observations used to determine accuracy of the MLP model. It measures the average absolute difference between the predicted value and the actual value. The lower the MAE is the more accurate the model will be.

RMSE: The root mean squared error or root mean squared deviation is another method to determine the accuracy of the MLP model. It calculates square root of the average of squared differences between predicted and actual values square root of the average of squared differences between predicted and actual values. The lower the index is, the higher the accuracy of the modal.

MAPE: Mean Absolute Percentage Error is a metric that calculates the percentage difference between predicted and actual values. It is used for evaluating the accuracy of a model when the error magnitude will be important. However, MAPE can be unreliable when actual values are close to zero. Similarly, to the other indexes, the lower the index is, the higher the accuracy of the modal.

SMAPE: Symmetric Mean Absolute Percentage Error is a variant of MAPE that symmetrically measures the percentage difference between predicted and actual values. It is useful when the magnitude of errors is important, and it is also easy to interpret. Similarly, to the other indexes, the lower the index is, the higher the accuracy of the modal.

```r
# Calculate MAPE
MAPE <- function(actual, predicted) {
  mean(abs((actual - predicted)/actual)) * 100
}


# Calculate SMAPE
smape <- function(actual, forecast){
  n <- length(actual)
  smape <- (1/n) * sum(2 * abs(forecast - actual) / (abs(forecast) + abs(actual))) * 100
  return(smape)
}


eval_list <- function(actual, predicted){
  rmse_mlp <- rmse(actual, predicted)
  mae_mlp <- mae(actual, predicted)
  mape_mlp <- MAPE(actual = actual, predicted = predicted)
  smape_mlp <- smape(actual, predicted)
  return(c(rmse_mlp,mae_mlp,mape_mlp,smape_mlp))
}
```

Matrix library was used for rmse and mae calculation.

AR Approach

```r
#................Testing Performance.............................#


eval_mlp_v1 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v1)
eval_mlp_v2 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v2)
eval_mlp_v3 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v3)
eval_mlp_v4 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v4)
eval_mlp_v5 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v5)
eval_mlp_v6 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v6)
eval_mlp_v7 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v7)
eval_mlp_v8 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v8)
eval_mlp_v9 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v9)
eval_mlp_v10 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v10)
eval_mlp_v11 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v11)
eval_mlp_v12 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v12)








#............Neural Network Comparison................................#

comp <- rbind(eval_mlp_v1,eval_mlp_v2,eval_mlp_v3,eval_mlp_v4,eval_mlp_v5,eval_mlp_v6,eval_mlp_v7,eval_mlp_v8,eval_mlp_v9,eval_mlp_v10,eval_mlp_v11,eval_mlp_v12)
colnames(comp) <- c("RMSE","MAE","MAPE","SMAPE")
rownames(comp) <- c('t1','t2','t3','t4','t7')
comp
```

NARX Approach

```r
#Testing Performance#
narx_eval_mlp_v1 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), narx_rescaled_v1)
narx_eval_mlp_v2 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), narx_rescaled_v2)
narx_eval_mlp_v3 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), narx_rescaled_v3)
narx_eval_mlp_v4 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), narx_rescaled_v4)
narx_eval_mlp_v5 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), narx_rescaled_v5)

narx_comp <- rbind(narx_eval_mlp_v1,narx_eval_mlp_v2,narx_eval_mlp_v3,narx_eval_mlp_v4,narx_eval_mlp_v5)
colnames(narx_comp) <- c("RMSE","MAE","MAPE","SMAPE")
rownames(narx_comp) <- c('t8_1','t8_2','t8_3','t8_4','t8_7','t6_1','t6_2','t6_3','t6_4','t6_7','t7_1','t7_2','t7_3','t7_4','t7_7')
narx_comp
```

Performance Comparison

AR Approach

	RMSE	MAE	MAPE	SMAPE
eval_mlp_v1	3.014562	2.381676	6.041395	6.038595
eval_mlp_v2	3.638082	2.943892	7.466884	7.511863
eval_mlp_v3	2.893804	2.302263	5.820255	5.837522
eval_mlp_v4	3.116282	2.335755	5.873508	5.930763
eval_mlp_v5	3.080851	2.450522	6.195663	6.209950
eval_mlp_v6	2.993246	2.361713	5.977417	6.004999
eval_mlp_v7	2.990735	2.329923	5.904334	5.904998
eval_mlp_v8	2.966622	2.333424	5.904007	5.919353
eval_mlp_v9	4.498110	3.658754	8.981405	9.256153
eval_mlp_v10	2.939125	2.322774	5.841847	5.934291
eval_mlp_v11	2.882224	2.299560	5.855124	5.818495
eval_mlp_v12	2.920287	2.303576	5.835785	5.840468

AR	Inputs	Description
MLP 1	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 6 nodes and 1 hidden layer
MLP 2	target~ t1 +t2 +t3 +t4	A nonlinear NN with 5 nodes and 1 hidden layer
MLP 3	target~ t1 +t3 +t4 +t7	A nonlinear NN with 5 nodes and 1 hidden layer
MLP 4	target~ t2 +t3 +t4 +t7	A nonlinear NN with 5 nodes and 1 hidden layer
MLP 5	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 5,1 nodes and 2 hidden layers
MLP 6	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 3,2,1 nodes and 3 hidden layer
MLP 7	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 4,2 nodes and 2 hidden layers
MLP 8	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 3,3 nodes and 2 hidden layers
MLP 9	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 1,2,2,1 nodes and 4 hidden layer sigmoid activation function
MLP 10	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 4,2 nodes and 2 hidden layers sigmoid activation function
MLP 11	target~ t1 +t2 +t3 +t4 +t7	A linear NN with 4,2 nodes and 2 hidden layers hyperbolic tangent activation function learning rate 0.01
MLP 12	target~ t1 +t2 +t3 +t4 +t7	A nonlinear NN with 1,2,3 nodes and 3 hidden layers sigmoid activation function learning rate 0.05
		Stepmax = 1e7

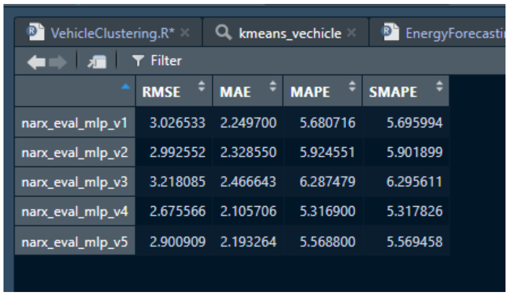
Among the created AR models let's consider AR MLP 11 (best 2-layer model) and AR MLP 4 (best 1 layer model)

Total Number of Weight parameters for MLP 11 = 8

Total Number of weight parameters MLP 4 = 5

The MLP 11 appears to be the most preferred structure. Not only does MLP 11 has more data parameters which allows it to be a much more complex model that fits the data better but also it has lower values for the evaluation compared to MLP 4. Hence, it can be determined that MLP 11 is better than MLP 4.

NARX Approach



	RMSE	MAE	MAPE	SMAPE
narx_eval_mlp_v1	3.026533	2.249700	5.680716	5.695994
narx_eval_mlp_v2	2.992552	2.328550	5.924551	5.901899
narx_eval_mlp_v3	3.218085	2.466643	6.287479	6.295611
narx_eval_mlp_v4	2.675566	2.105706	5.316900	5.317826
narx_eval_mlp_v5	2.900909	2.193264	5.568800	5.569458

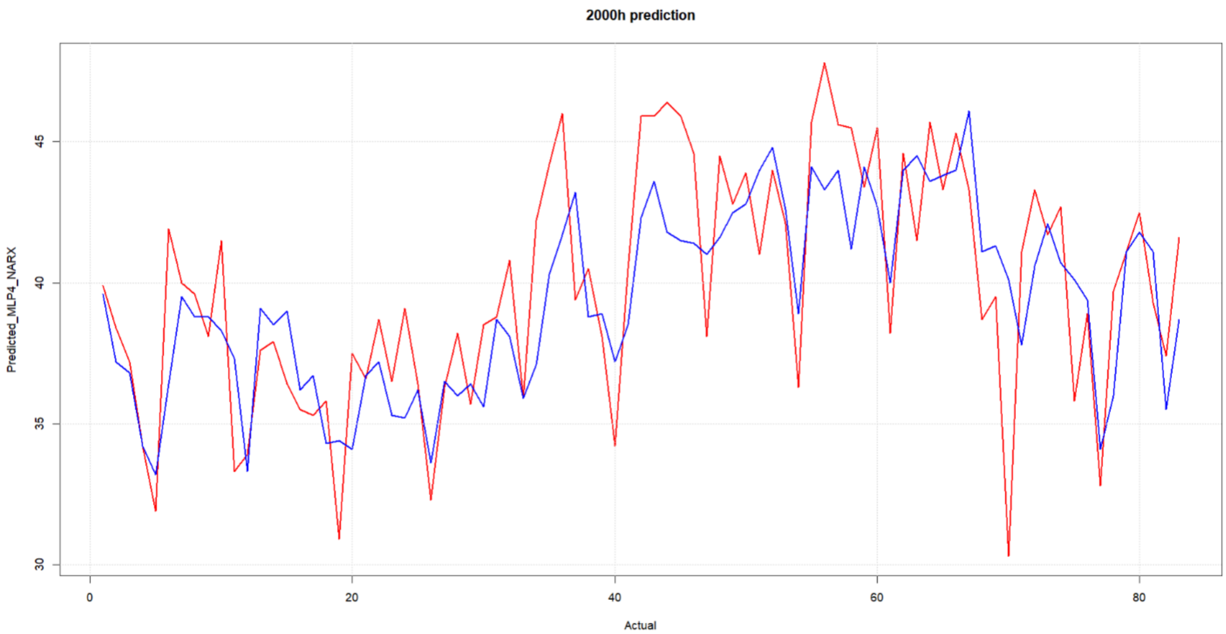
NARX	Inputs	Description
MLP 1	target~ t8_1 + t8_2 + t8_3 + t8_4 + t8_7 + t6_1 + t6_2 + t6_3 + t6_4 + t6_7 + t7_1 + t7_2 + t7_3 +t7_4 +t7_7	A nonlinear NN with 16 nodes and 1 hidden layer's sigmoid activation function
MLP 2	target~ t8_3+t8_4+t8_7+t6_1+t6_4+t6_7+t7_1+t7_2+t7_7	A nonlinear NN with 10nodes and 1 hidden layer
MLP 3	target~ t8_3+t8_4+t8_7+t6_1+t6_4+t6_7+t7_1+t7_2	A nonlinear NN with 6,10 nodes and 2 hidden layers
MLP 4	target~ t8_1 + t8_2 + t8_3 + t8_4 + t8_7 + t6_1 + t6_2 + t6_3 + t6_4 + t6_7 + t7_1 + t7_2 + t7_3 +t7_4 +t7_7	A nonlinear NN with 1,10,5 nodes and 3 hidden layers sigmoid activation function learning rate 0.02
MLP 5	target~ t8_1 + t8_2 + t8_3 + t8_4 + t8_7 + t6_1 + t6_2 + t6_3 + t6_4 + t6_7 + t7_1 + t7_2 + t7_3 +t7_4 +t7_7	A nonlinear NN with 2,14 nodes and 2 hidden layers sigmoid activation function learning rate 0.05

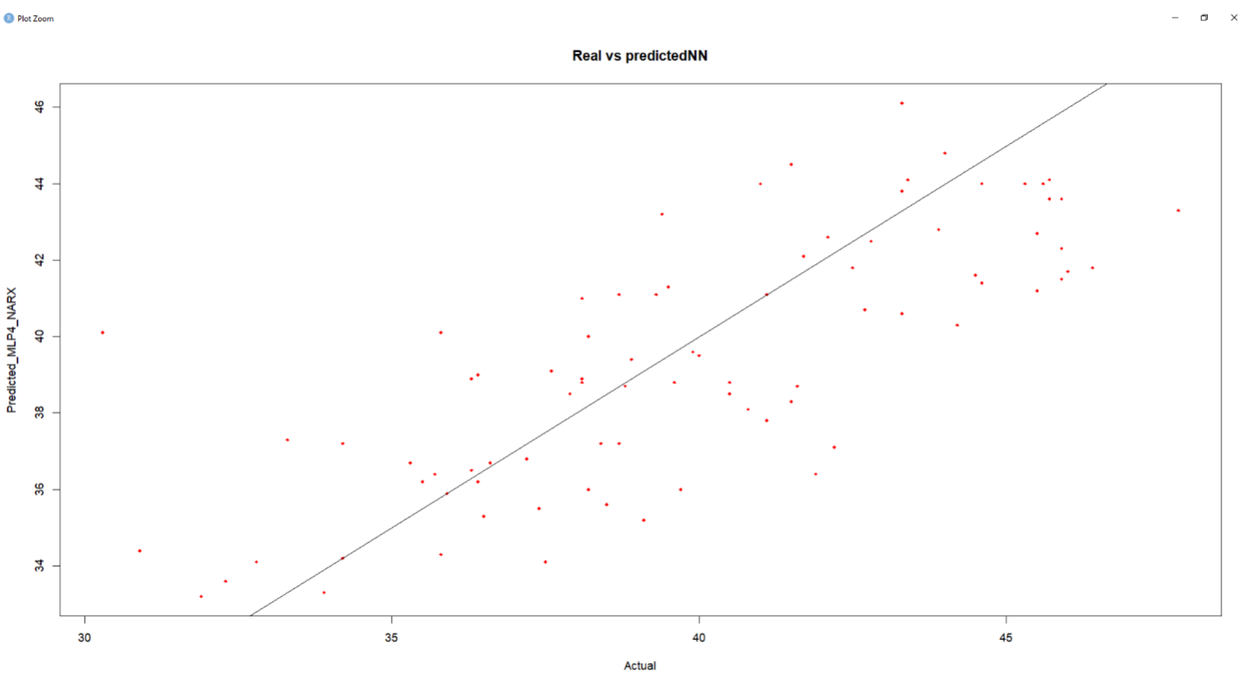
NARX Findings

After carefully analyzing the results of AR and NARX approaches we can come to the conclusion that NARX models generally outperformed AR models in terms of accuracy as proved by the lower values for the MAE, RSME, MAPE, SMAPE statistical indices. Hence, we can determine that NARX models can effectively predict the target results than a AR model for electricity loa forecasting solutions.

Best Performing Model

As we consider the statical indexes MAE, RSME, MAPE, SMAPE we can determine that NARX MLP 4 is the best performing model.






```
> cat("NARX MLP 4 RMSE =", narx_eval_mlp_v4[1], "\n")
NARX MLP 4 RMSE = 2.675566
> cat("NARX MLP 4 MAE =", narx_eval_mlp_v4[2], "\n")
NARX MLP 4 MAE = 2.105706
> cat("NARX MLP 4 MAPE =", narx_eval_mlp_v4[3], "\n")
NARX MLP 4 MAPE = 5.3169
> cat("NARX MLP 4 SMAPE =", narx_eval_mlp_v4[4], "\n")
NARX MLP 4 SMAPE = 5.317826
>
```

Appendix

Partition Clustering

```
# Import libraries
library(readxl)
library(dplyr)
library(NbClust)
library(stats)
library(tidyverse)
library(ggplot2)
library(cluster)
library(factoextra)
library(dataset)
library(fpc)
library(ggfortify)

#Removing existing objects
rm(list = ls())
setwd("C:/Users/Dhanuja/Desktop/MLcwk")

vechicleset <- read_excel("Cwk/vehicles.xlsx")

##PREPROCCESING##
# Remove the final coulum
vechicles <- vechicleset[, -c(1, ncol(vechicleset))]

# data Type
class(vechicleset)

# Check for missing values
```

```
vechicles <- vechicles[complete.cases(vechicles), ]

# If there are missing values, remove them using the na.omit() function
vechicles <- na.omit(vechicles)

summary(vechicles)



# Scale the data using the standardization method
vechicles <- scale(vechicles)
# create a boxplot to visualize distribution in order to find outliers
boxplot(vechicles)

vechicle_outliers <- apply(vechicles,1,function(x) any(x > 3 | x < -3))
cleaned_vechicles <- subset(vechicles, !vechicle_outliers)

boxplot(cleaned_vechicles)
##Finding the Optimal Number of Clusterrs##
#NBClust
Methodhttp://127.0.0.1:10465/graphics/plot_zoom_png?width=1188&height=827
set.seed(26)
clust_no_nb = NbClust(cleaned_vechicles,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")

#Elbow Method
set.seed(28)
k_val <- 2:10
WSS <- sapply(k_val,function(k_val){kmeans(cleaned_vechicles,centers =
k_val)$tot.withinss})
plot(k_val, WSS, type = "b", xlab = "Number of K values", ylab = "WSS")
```

```
#Silouette Method
set.seed(32)
fviz_nbclust(cleaned_vechicles, kmeans, method = "silhouette")

#Gap-Stat Method
set.seed(34)
fviz_nbclust(cleaned_vechicles,kmeans,method = "gap_stat")

##K MEANS ##
k = 3
kmeans_vechicle <- kmeans(cleaned_vechicles,centers = k,nstart = 10)
kmeans_vechicle

#Calculating Centers
kmeans_vechicle$centers

fviz_cluster(kmeans_vechicle,data=cleaned_vechicles)

vechicle_cluster <- data.frame(cleaned_vechicles,cluster =
as.factor(kmeans_vechicle$cluster))
head(vechicle_cluster)

vechicle_wss = kmeans_vechicle$tot.withinss
vechicle_wss

vechicle_bss = kmeans_vechicle$betweenss
vechicle_bss

# Calculating the ratio between WSS and BSS
```

```
vechicle_wss/vechicle_bss

# Calculate the total sum of squares (TSS)
TSS <- sum(apply(cleaned_vechicles, 2, var)) * nrow(cleaned_vechicles)

#Calculating the ratio of between_cluster_sums_of_squares (BSS) over
total_sum_of_Squares (TSS)
TSS/vechicle_bss

#silhouette plot
vehicle_sil = silhouette(kmeans_vechicle$cluster,dist(cleaned_vechicles))
fviz_silhouette(vehicle_sil)

#AVG silhouette width score
silhouette_avg = mean(vehicle_sil[,3])
silhouette_avg

#//………………………………………………………………………………………//#

###PCA###
v_pca = prcomp(cleaned_vechicles)
summary(v_pca)

#EigenValues and Eigenvectors
v_eigenvalues <- v_pca$sdev^2
v_eigenvectors <- v_pca$rotation

v_eigenvalues
v_eigenvectors
```

```
#cumulative score per principal components
fviz_eig(v_pca, addlabels = TRUE, ylim = c(0, 60))

#Create Tranformed Dataset(PCA as attributes)#
vechicle_transformed <- predict(v_pca,cleaned_vechicles)
summary(vechicle_transformed)

#cumulative score per principal components
PVE <- v_pca$sdev^2/sum(v_pca$sdev^2)
PVE <- round(PVE,2)
cum_score = cumsum(PVE)
pc_num = sum(cum_score<0.92) + 1
pc_num

vechicle_pc_dataset = data.frame(vechicle_transformed[, 1:pc_num])

##Finding the Optimal Number of Clusterrs##
#NBClust Method
set.seed(26)
clust_no_pc = NbClust(vechicle_pc_dataset,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")

#Elbow Method
set.seed(28)
k_val_pc <- 2:10
WSS_pc <- sapply(k_val,function(k_val){kmeans(vechicle_pc_dataset,centers =
k_val)$tot.withinss})
plot(k_val_pc, WSS_pc, type = "b", xlab = "Number of K values", ylab = "WSS")
```

```
#Silouette Method
set.seed(32)
fviz_nbclust(vechicle_pc_dataset, kmeans, method = "silhouette")

#Gap-Stat Method
set.seed(34)
fviz_nbclust(vechicle_pc_dataset,kmeans,method = "gap_stat")



##K MEANS PC##
k_pc = 3
kmeans_pc <- kmeans(vechicle_pc_dataset,centers = k_pc,nstart = 10)
kmeans_pc

#Calculating Centers
kmeans_pc$centers

fviz_cluster(kmeans_pc,data=vechicle_pc_dataset)
autoplot(kmeans_pc, vechicle_pc_dataset, frame=TRUE)

vechicle_cluster_pc <- data.frame(vechicle_pc_dataset,cluster =
as.factor(kmeans_pc$cluster))
head(vechicle_cluster_pc)

vechicle_wss_pc = kmeans_pc$tot.withinss
vechicle_bss_pc = kmeans_pc$betweenss
vechicle_bss_pc
vechicle_wss_pc
```

```
# Calculating the ratio between WSS and BSS
vechicle_wss_pc/vechicle_bss_pc

# Calculate the total sum of squares (TSS)
TSS_pc <- sum(apply(vechicle_pc_dataset, 2, var)) * nrow(vechicle_pc_dataset)
TSS_pc

#Calculating the ratio of between_cluster_sums_of_squares (BSS) over
total_sum_of_Squares (TSS)
TSS_pc/vechicle_bss_pc

#silhouette plot
vehicle_sil_pc = silhouette(kmeans_pc$cluster,dist(vechicle_pc_dataset))
fviz_silhouette(vehicle_sil_pc)

#AVG silhouette width score
silhouette_avg_pc = mean(vehicle_sil_pc[,2])

##Calinski-Harabasz Index##

set.seed(40)
# Compute the Calinski-Harabasz Index
ch_index <- round(calinhara(vechicle_pc_dataset,kmeans_pc$cluster),digits=3)
ch_index

set.seed(123)
# Set the number of clusters to evaluate
k <- 2:10

# Initialize empty vector to store Calinski-Harabasz index values
```

```
ch_scores <- vector("numeric", length(k))

# Compute the Calinski-Harabasz index for each number of clusters
for (i in 1:length(k)) {
  km <- kmeans(vechicle_pc_dataset, centers = k[i], nstart = 10)
  ch_scores[i] <- calinhara(vechicle_pc_dataset, km$cluster)
}
ch_scores
## Visualize the Calinski-Harabasz Index##
# Plot the Calinski-Harabasz index values
set.seed(123)
plot(k, ch_scores, type = "b", xlab = "Number of Clusters", ylab = "Calinski-Harabasz Index", main = "Calinski-Harabasz Index against Number of Clusters")
```

Energy Forecasting

```
#import Libaries
library(readxl)
library(caret)
library(dplyr)
library(neuralnet)
library(Metrics)
library(stats)
```

```
#Removing existing objects
rm(list = ls())

# Normalize the data from 0 to 1
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x))) }

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



# Calculate MAPE
MAPE <- function(actual, predicted) {
  mean(abs((actual - predicted)/actual)) * 100
}



# Calculate SMAPE
smape <- function(actual, forecast){
  n <- length(actual)
  smape <- (1/n) * sum(2 * abs(forecast - actual) / (abs(forecast) + abs(actual))) * 100
  return(smape)
}



eval_list <- function(actual, predicted){
  rmse_mlp <- rmse(actual, predicted)
  mae_mlp <- mae(actual, predicted)
```

```
  mape_mlp <- MAPE(actual = actual, predicted = predicted)
  smape_mlp <- smape(actual, predicted)
  return(c(rmse_mlp,mae_mlp,mape_mlp,smape_mlp))
}

#Import Dataset
energyDataSet <- read_excel("Cwk/uow_consumption.xlsx")



#Randomize the dataset
#energyDataSet <- energyDataSet[sample(nrow(energyData)), ]
#Setting col names
colnames(energyDataSet) <- c("Date","1800h","1900h","2000h")

#Convert into time series
energyDataSet$Date <- ts(energyDataSet$Date)

#Cheching for missing values
energyDataSet <- energyDataSet[complete.cases(energyDataSet),]

# If there are missing values, remove them using the na.omit() function
energyDataSet <- na.omit(energyDataSet)



#Select 20th hour values
targetData <- energyDataSet[,"2000h"]

colnames(targetData) <- "target"

#Noramlizing
```

```
targetData <- normalize(targetData)




#creating time delayed loads
load1 <- lag(targetData, 1)
load2 <- lag(targetData, 2)
load3 <- lag(targetData, 3)
load4 <- lag(targetData, 4)
load7 <- lag(targetData, 7)




#creating I/O matrix
io_matrix <- cbind(load1,load2,load3,load4,load7,targetData)
colnames(io_matrix) <- c('t1','t2','t3','t4','t7','target')

io_matrix_v1 <- cbind(load1,load2,load3,load4,targetData)
colnames(io_matrix_v1) <- c('t1','t2','t3','t4','target')



io_matrix_v2 <- cbind(load1,load3,load4,load7,targetData)
colnames(io_matrix_v2) <- c('t1','t3','t4','t7','target')

io_matrix_v3 <- cbind(load2,load3,load4,load7,targetData)
colnames(io_matrix_v3) <- c('t2','t3','t4','t7','target')
```

```
#Removing empty values
io_matrix <- io_matrix[complete.cases(io_matrix),]
io_matrix_v1 <- io_matrix_v1[complete.cases(io_matrix_v1),]
io_matrix_v2 <- io_matrix_v2[complete.cases(io_matrix_v2),]
io_matrix_v3 <- io_matrix_v3[complete.cases(io_matrix_v3),]



#making the testing and training dataset
set.seed(123)

training <- io_matrix[1:380,]

testing <- io_matrix[381:nrow(io_matrix),]

#Create MLP with different parameters
#.............................................................................................................#
##............Different inputs...........................##
#MLP 1#
names(training)
relation <- as.formula("target~ t1 +t2 +t3 +t4 +t7")
set.seed(113)
mlp_v1 <- neuralnet(formula = relation,data = training,hidden = 6,linear.output =
FALSE,)
plot(mlp_v1)

#MLP 2
relation2 <- as.formula("target~ t1 +t2 +t3 +t4")
set.seed(114)
mlp_v2 <- neuralnet(formula = relation2,  data = training, hidden = 5, linear.output =
FALSE,)
```

```
plot(mlp_v2)


#MLP 3
relation3 <- as.formula("target~ t1 +t3 +t4 +t7")
set.seed(115)
mlp_v3 <- neuralnet(formula = relation3, data = training,hidden = 5,linear.output =
FALSE,)
plot(mlp_v3)


#MLP 4
relation4 <- as.formula("target~ t2 +t3 +t4 +t7")
set.seed(116)
mlp_v4 <- neuralnet(formula = relation4,data = training,hidden = 5,linear.output =
FALSE,)
plot(mlp_v4)


#......Different Hidden Layers & Nodes..........#
#MLP 5
set.seed(116)
mlp_v5 <- neuralnet(formula = relation,data = training,hidden = c(5,1),linear.output =
FALSE,stepmax = 1e7)
plot(mlp_v5)
#MLP 6
set.seed(117)
mlp_v6 <- neuralnet(formula = relation,data = training,hidden = c(3,2,1),linear.output =
FALSE,stepmax = 1e7)
plot(mlp_v6)
#MLP 7
set.seed(118)
mlp_v7 <- neuralnet(formula = relation,data = training,hidden = c(4,2), linear.output =
FALSE,stepmax = 1e7)
```

plot(mlp_v7)

#MLP 8

set.seed(119)

mlp_v8 <- neuralnet(formula = relation,data = training,hidden = c(3,3), linear.output = FALSE, stepmax = 1e7)

plot(mlp_v8)

#...........Different Activation Functions..............................#

#MLP 9

set.seed(103)

mlp_v9 <- neuralnet(formula = relation,data = training, hidden = c(1,2,2,1),linear.output = FALSE,act.fct = "logistic", stepmax = 1e7)

plot(mlp_v9)

#MLP 10

set.seed(104)

mlp_v10 <- neuralnet(formula = relation,data = training,hidden = c(4,2),linear.output = FALSE, act.fct = "tanh",stepmax = 1e7)

plot(mlp_v10)

#MLP 11

set.seed(105)

mlp_v11 <- neuralnet(formula = relation,data = training,hidden = c(4,2),linear.output = TRUE,act.fct = "logistic",algorithm = "rprop+",learningrate = 0.01,stepmax = 1e7)

plot(mlp_v11)

#MLP 12

set.seed(107)

mlp_v12 <- neuralnet(formula = relation,data = training,hidden = c(1,2,3),linear.output = FALSE,act.fct = "logistic",algorithm = "rprop+", learningrate = 0.05,stepmax = 1e7)

plot(mlp_v12)

#.......................Predictions.................................#

```
predict_mlp_v1 <- neuralnet::compute(mlp_v1,testing)
predict_mlp_v2 <- neuralnet::compute(mlp_v2,testing)
predict_mlp_v3 <- neuralnet::compute(mlp_v3,testing)
predict_mlp_v4 <- neuralnet::compute(mlp_v4,testing)
predict_mlp_v5 <- neuralnet::compute(mlp_v5,testing)
predict_mlp_v6 <- neuralnet::compute(mlp_v6,testing)
predict_mlp_v7 <- neuralnet::compute(mlp_v7,testing)
predict_mlp_v8 <- neuralnet::compute(mlp_v8,testing)
predict_mlp_v9 <- neuralnet::compute(mlp_v9,testing)
predict_mlp_v10 <- neuralnet::compute(mlp_v10,testing)
predict_mlp_v11 <- neuralnet::compute(mlp_v11,testing)
predict_mlp_v12 <- neuralnet::compute(mlp_v12,testing)

#...............Rescled  Predictions...........................#
#Rescale the Data func
target_min <- min(energyDataSet$'2000h')
target_max <- max(energyDataSet$'2000h')

rescale_predict_mlp_v1 <-
unnormalize(predict_mlp_v1$net.result,target_min,target_max)
rescale_predict_mlp_v2 <-
unnormalize(predict_mlp_v2$net.result,target_min,target_max)
rescale_predict_mlp_v3 <-
unnormalize(predict_mlp_v3$net.result,target_min,target_max)
rescale_predict_mlp_v4 <-
unnormalize(predict_mlp_v4$net.result,target_min,target_max)
rescale_predict_mlp_v5 <-
unnormalize(predict_mlp_v5$net.result,target_min,target_max)
rescale_predict_mlp_v6 <-
unnormalize(predict_mlp_v6$net.result,target_min,target_max)
rescale_predict_mlp_v7 <-
unnormalize(predict_mlp_v7$net.result,target_min,target_max)
```

rescale_predict_mlp_v8 <-
unnormalize(predict_mlp_v8$net.result,target_min,target_max)

rescale_predict_mlp_v9 <-
unnormalize(predict_mlp_v9$net.result,target_min,target_max)

rescale_predict_mlp_v10 <-
unnormalize(predict_mlp_v10$net.result,target_min,target_max)

rescale_predict_mlp_v11 <-
unnormalize(predict_mlp_v11$net.result,target_min,target_max)

rescale_predict_mlp_v12 <-
unnormalize(predict_mlp_v12$net.result,target_min,target_max)

#...............Testing Performance...........................#

eval_mlp_v1 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v1)

eval_mlp_v2 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v2)

eval_mlp_v3 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v3)

eval_mlp_v4 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v4)

eval_mlp_v5 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v5)

eval_mlp_v6 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v6)

eval_mlp_v7 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v7)

eval_mlp_v8 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v8)

eval_mlp_v9 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v9)

eval_mlp_v10 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
rescale_predict_mlp_v10)

eval_mlp_v11 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v11)

eval_mlp_v12 <- eval_list(unnormalize(testing[,"target"],target_min,target_max), rescale_predict_mlp_v12)

#............Neural Network Comparison...............................#

comp <- rbind(eval_mlp_v1,eval_mlp_v2,eval_mlp_v3,eval_mlp_v4,eval_mlp_v5,eval_mlp_v6,eval_mlp_v7,eval_mlp_v8,eval_mlp_v9,eval_mlp_v10,eval_mlp_v11,eval_mlp_v12)

colnames(comp) <- c("RMSE","MAE","MAPE","SMAPE")

rownames(comp) <- c('t1','t2','t3','t4','t7')

comp

```
predictedVSactual <- cbind(
                        round(rescale_predict_mlp_v1,digits = 1),
                        round(rescale_predict_mlp_v2,digits = 1),
                        round(rescale_predict_mlp_v3,digits = 1),
                        round(rescale_predict_mlp_v4,digits = 1),
                        round(rescale_predict_mlp_v5,digits = 1),
                        round(rescale_predict_mlp_v6,digits = 1),
                        round(rescale_predict_mlp_v7,digits = 1),
```

```
                    round(rescale_predict_mlp_v8,digits = 1),
                    round(rescale_predict_mlp_v9,digits = 1),
                    round(rescale_predict_mlp_v10,digits = 1),
                    round(rescale_predict_mlp_v11,digits = 1),
                    round(rescale_predict_mlp_v12,digits = 1),
                    unnormalize(testing[,"target"],target_min,target_max))

colnames(predictedVSactual) <- c("Predicted_MLP1",
                        "Predicted_MLP2",
                        "Predicted_MLP3",
                        "Predicted_MLP4",
                        "Predicted_MLP5",
                        "Predicted_MLP6",
                        "Predicted_MLP7",
                        "Predicted_MLP8",
                        "Predicted_MLP9",
                        "Predicted_MLP10",
                        "Predicted_MLP11",
                        "Predicted_MLP12",
                        "Actual")



####..........................................................................................####

#...............<<<<<<<<<<<<<<<<<<NARX
Approach>>>>>>>>>>>>>>>>>>>..........................................#



#Creating Loads for 18h and 19h
energyDataSet[,"1800h"] <- normalize(energyDataSet[,"1800h"])
```

```
energyDataSet[,"1900h"] <- normalize(energyDataSet[,"1900h"])


load6_1 <- lag(energyDataSet[,"1800h"], 1)
load6_2 <- lag(energyDataSet[,"1800h"], 2)
load6_3 <- lag(energyDataSet[,"1800h"], 3)
load6_4 <- lag(energyDataSet[,"1800h"], 4)
load6_7 <- lag(energyDataSet[,"1800h"], 7)



load7_1 <- lag(energyDataSet[,"1900h"], 1)
load7_2 <- lag(energyDataSet[,"1900h"], 2)
load7_3 <- lag(energyDataSet[,"1900h"], 3)
load7_4 <- lag(energyDataSet[,"1900h"], 4)
load7_7 <- lag(energyDataSet[,"1900h"], 7)


#creating I/O matrix
io_matrix_narx <-
cbind(load1,load2,load3,load4,load7,load6_1,load6_2,load6_3,load6_4,load6_7,load7_1,load7_2,load7_3,load7_4,load7_7,targetData)
colnames(io_matrix_narx) <-
c('t8_1','t8_2','t8_3','t8_4','t8_7','t6_1','t6_2','t6_3','t6_4','t6_7','t7_1','t7_2','t7_3','t7_4','t7_7','target')
narx_relation <- as.formula("target~ t8_1 + t8_2 + t8_3 + t8_4 + t8_7 + t6_1 + t6_2 +
t6_3 + t6_4 + t6_7 + t7_1 + t7_2 + t7_3 +t7_4 +t7_7")


io_matrix_v1_narx <-
cbind(load3,load4,load7,load6_1,load6_4,load6_7,load7_1,load7_2,load7_7,targetData)
colnames(io_matrix_v1_narx) <-
c('t8_3','t8_4','t8_7','t6_1','t6_4','t6_7','t7_1','t7_2','t7_7','target')
narx_relation_v1 <- as.formula("target~
t8_3+t8_4+t8_7+t6_1+t6_4+t6_7+t7_1+t7_2+t7_7")
```

```
io_matrix_v2_narx <-
cbind(load3,load4,load7,load6_1,load6_4,load6_7,load7_1,load7_2,load7_7,targetData)

colnames(io_matrix_v2_narx) <-
c('t8_3','t8_4','t8_7','t6_1','t6_4','t6_7','t7_1','t7_2','target')

narx_relation_v2 <- as.formula("target~ t8_3+t8_4+t8_7+t6_1+t6_4+t6_7+t7_1+t7_2")

#Removing empty values

io_matrix_narx <- io_matrix_narx[complete.cases(io_matrix_narx),]

io_matrix_v1_narx <- io_matrix_v1_narx[complete.cases(io_matrix_v1_narx),]

io_matrix_v2_narx <- io_matrix_v2_narx[complete.cases(io_matrix_v2_narx),]



#Creating Training and testing dataset

narx_training <- io_matrix_narx[1:380,]


narx_testing <- io_matrix_narx[381:nrow(io_matrix_narx),]


##Create MLP with different parameters

#NARX MLP 1

set.seed(123)

narx_mlp_v1 <- neuralnet(formula = narx_relation,data = narx_training,hidden =
16,linear.output = FALSE,act.fct = "logistic",stepmax = 1e7)

plot(narx_mlp_v1)

#NARX MLP 2

set.seed(123)

narx_mlp_v2 <- neuralnet(formula = narx_relation_v1,data = narx_training, hidden =
10,linear.output = FALSE,stepmax = 1e7)

plot(narx_mlp_v2)

#NARX MLP 3

set.seed(124)

narx_mlp_v3 <- neuralnet(formula = narx_relation_v2,data = narx_training,hidden =
c(6,10),linear.output = FALSE,stepmax = 1e7)
```

```
plot(narx_mlp_v3)

#NARX MLP 4

set.seed(125)

narx_mlp_v4 <- neuralnet(formula = narx_relation,data = narx_training,hidden =
c(1,10,5),linear.output = FALSE, act.fct = "logistic",algorithm = "rprop+",learningrate =
0.02,stepmax = 1e7)

plot(narx_mlp_v4)

#NARX MLP 5

set.seed(126)

narx_mlp_v5 <- neuralnet(formula = narx_relation,
                    data = narx_training,
                    hidden = c(2,14),
                    linear.output = FALSE,
                    act.fct = "logistic",
                    algorithm = "rprop+",
                    learningrate = 0.05,
                    stepmax = 1e7

)
plot(narx_mlp_v5)

#Predictions

narx_prediction_v1 <- neuralnet::compute(narx_mlp_v1,narx_testing)

narx_prediction_v2 <- neuralnet::compute(narx_mlp_v2,narx_testing)

narx_prediction_v3 <- neuralnet::compute(narx_mlp_v3,narx_testing)

narx_prediction_v4 <- neuralnet::compute(narx_mlp_v4,narx_testing)
```

```
narx_prediction_v5 <- neuralnet::compute(narx_mlp_v5,narx_testing)

#Rescaled Predictions

narx_rescaled_v1 <- unnormalize(narx_prediction_v1$net.result,target_min,target_max)
narx_rescaled_v2 <- unnormalize(narx_prediction_v2$net.result,target_min,target_max)
narx_rescaled_v3 <- unnormalize(narx_prediction_v3$net.result,target_min,target_max)
narx_rescaled_v4 <- unnormalize(narx_prediction_v4$net.result,target_min,target_max)
narx_rescaled_v5 <- unnormalize(narx_prediction_v5$net.result,target_min,target_max)

#Testing Performance#
narx_eval_mlp_v1 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
narx_rescaled_v1)
narx_eval_mlp_v2 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
narx_rescaled_v2)
narx_eval_mlp_v3 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
narx_rescaled_v3)
narx_eval_mlp_v4 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
narx_rescaled_v4)
narx_eval_mlp_v5 <- eval_list(unnormalize(testing[,"target"],target_min,target_max),
narx_rescaled_v5)

narx_comp <-
rbind(narx_eval_mlp_v1,narx_eval_mlp_v2,narx_eval_mlp_v3,narx_eval_mlp_v4,narx_
eval_mlp_v5)
colnames(narx_comp) <- c("RMSE","MAE","MAPE","SMAPE")
rownames(narx_comp) <-
c('t8_1','t8_2','t8_3','t8_4','t8_7','t6_1','t6_2','t6_3','t6_4','t6_7','t7_1','t7_2','t7_3','t7_4','t7_
7')
narx_comp
```

```
narx_predictedVSactual <- cbind(
                    round(narx_rescaled_v1,digits = 1),
                    round(narx_rescaled_v2,digits = 1),
                    round(narx_rescaled_v3,digits = 1),
                    round(narx_rescaled_v4,digits = 1),
                    round(narx_rescaled_v5,digits = 1),

                    unnormalize(testing[,"target"],target_min,target_max))

colnames(narx_predictedVSactual) <- c("Predicted_MLP1_NARX",
                        "Predicted_MLP2_NARX",
                        "Predicted_MLP3_NARX",
                        "Predicted_MLP4_NARX",
                        "Predicted_MLP5_NARX",
                        "Actual")

#..........Graphical  Charts.............

par(mfrow = c(1,1))
plot(
  unnormalize(testing[,"target"],target_min,target_max),
  round(narx_rescaled_v4,digits = 1),
  col = 'red',
  main = 'Real vs predictedNN',
  pch = 18,
  cex = 0.7,
  xlab = 'Actual',
  ylab = 'Predicted_MLP4_NARX'
)
```

```
abline(a = 0, b = 1, h = 90, v = 90)

x = 1:length(unnormalize(testing[,"target"],target_min,target_max))
plot(
  x,
  unnormalize(testing[,"target"],target_min,target_max),
  col = "red",
  type = "l",
  lwd = 2,
  main = "2000h prediction",
  xlab = 'Actual',
  ylab = 'Predicted_MLP4_NARX'
)
lines(x, round(narx_rescaled_v4,digits = 1), col = "blue", lwd = 2)
grid()

cat("NARX MLP 4 RMSE =", narx_eval_mlp_v4[1], "\n")
cat("NARX MLP 4 MAE =", narx_eval_mlp_v4[2], "\n")
cat("NARX MLP 4 MAPE =", narx_eval_mlp_v4[3], "\n")
cat("NARX MLP 4 SMAPE =", narx_eval_mlp_v4[4], "\n")
```

References

"How to Calculate Mean Absolute Percentage Error (MAPE) in R." 04 Aug. 2021, https://www.r-bloggers.com/2021/08/how-to-calculate-mean-absolute-percentage-error-mape-in-r/.

"How to Calculate Root Mean Square Error (RMSE) in R." 23 Jul. 2021, https://www.r-bloggers.com/2021/07/how-to-calculate-root-mean-square-error-rmse-in-r/.

"How to Calculate SMAPE in R | R-bloggers." 03 Aug. 2021, https://www.r-bloggers.com/2021/08/how-to-calculate-smape-in-r/.

"How to Calculate Mean Absolute Percentage Error (MAPE) in R." 04 Aug. 2021, https://www.r-bloggers.com/2021/08/how-to-calculate-mean-absolute-percentage-error-mape-in-r/.

"How to measure clustering performances when there are no ground truth ...." 02 Jan. 2020, https://medium.com/@haataa/how-to-measure-clustering-performances-when-there-are-no-ground-truth-db027e9a871c.

"Home - RDocumentation." https://www.rdocumentation.org/.

"How to Remove Outliers in R | R-bloggers." 27 Sept. 2021, https://www.r-bloggers.com/2021/09/how-to-remove-outliers-in-r-3/.

"Methods and Models for Electric Load Forecasting: A ... - Sciendo." https://sciendo.com/downloadpdf/journals/jlst/11/1/article-p51.pdf.

"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/.